

FME® Readers and Writers

(formats supported by the Esri® ArcGIS® Data Interoperability extension)

About Quick Facts Tables

Each format's chapter overview includes high-level information about the format's characteristics:

Format Type Identifier

Every format supported by FME is uniquely identified by an uppercase alphanumeric string. This row lists the unique identifier for this format. For example, the identifier for IGDS Design is IGDS. The format type identifier is used in mapping files on **READER_TYPE** and **WRITER_TYPE** lines to define the reader and writer to be used for translation. In FME Objects, when a reader or writer is created, the format type identifier is used to specify its type.

Reader/Writer

FME typically allows reading from and writing to a supported data format. However, some formats support either reading *or* writing, but not both. This row specifies whether reading only, writing only, or reading and writing is available for this format.

Licensing Level

Some FME-supported formats require specific licensing. For example, certain formats are not supported in FME Base Edition. This row lists the minimum licensing level required to read from or write to the format.

Dependencies

Some formats require the installation of the application associated with the format, or may require an extra-cost plugin.

Dataset Type

FME reads features from datasets and writes features to datasets. The definition of a dataset varies between different readers and writers, depending nature of the format. This row specifies the type of dataset used by this format.

File Dataset: The most basic type of dataset is the file dataset: a single file whose extension is included in the list of supported file extensions. A file dataset reader opens the specified file and reads its features. A file dataset writer creates the specified file if it doesn't exist and writes features into it. In general, file dataset writers overwrite existing files, but some can append to existing files.

Directory Dataset: A directory dataset consists of a directory specification. A directory dataset reader examines the name of all the files in the specified directory and reads those files whose extensions are included in the list of supported file extensions. The order in which files are read is not documented. Many directory dataset readers provide an option to explicitly name a subset of the directory's files that are to be read. A directory dataset writer creates the specified directory if it does not exist. In general, if the directory does exist, any files in the directory that match the name of output files will be overwritten; however, some directory dataset writers can append to existing files.

Database Dataset: A database dataset consists of a set of information needed to connect to a database schema. A database dataset reader reads all the tables in the specified database schema. The order in which tables are read is not documented. Many database dataset readers provide an option to explicitly name a subset of the schema's tables that are to be read. Any tables in the specified database that match the name of output tables will be overwritten.

URL Dataset: A URL dataset consists of a uniform resource locator. A URL dataset reader connects over a network to a remote server and retrieves the data provided by that resource.

Typical File Extensions

Most file dataset readers will read data from any legally-named file, independent of extension.

Most formats are associated with one or more file extensions. This row lists the file extensions typically associated with this format. When a format uses several files with the same basename but different extensions, the primary extension is listed first and the ancillary ones are listed in parentheses.

Automated Translation Support

The FME typically allows automated translation to and from a supported data format. However, some formats support either automated reading *or* automated writing, but not both. This row specifies whether this format supports automated reading, writing, or both. When automated translation is not available, a custom mapping file must be used.

Feature Type

Every format supported by FME identifies the features in its datasets according to a well-defined data classification scheme. This primary classification is known as the feature's type, which serves as the main handle to a feature. This row describes the format's classification scheme. For example, a simple classification scheme is to identify features according to the file or table they reside in (the feature's type is the file base name or table name, respectively). For level-based formats, features are typically grouped by level (the feature's type is the level in which it resides). Another common classification is to group features according to the thematic layer to which they belong – for example, roads, railways, and rivers. Although these are the most common classification schemes, the list of possible schemes is quite broad because the classification chosen is individual to the format.

User-Defined Attributes

An FME feature consists of geometry and attributes. While many of a feature's attributes are predefined by FME and the feature's format (such attributes are constant from one dataset to another), some formats allow users to define custom attributes. These user-defined attributes give the format flexibility to store arbitrary amounts of domain-specific information in addition to geometry. This row specifies whether this format supports user-defined attributes.

Coordinate System Support

This row indicates whether datasets of this format can store coordinate system information, and if so, whether the reader extracts this information.

Generic Color Support

The `fme_color` and `fme_fill_color` feature attributes represent the red, green, and blue intensities of a feature. Intensities can each vary between 0.0 and 1.0, and are calculated by taking the color intensity and dividing it by the total intensity range. This row indicates whether or not the format supports the generic `fme_color` and `fme_fill_color` attributes.

If a format has generic color support, the reader will add both the generic FME color attributes and the format-specific color attributes to features. *Writers* that support generic color will give precedence to the format-specific color attributes if they are present in addition to the generic attributes.

Since both generic FME attributes and format-specific attributes exist on workspace features, it is important to note that the co-existence of the two types of attributes can sometimes cause a conflict, and the format-specific attribute will be deleted. For more information, see [Format-Specific Attributes and Generic FME Attributes in Workbench](#).

Spatial Index

This row applies only to readers: it indicates whether or not the native reader supports spatial indexing. The possible values are:

Never: This reader never provides a spatial index.

Optional: This reader may or may not provide a spatial index, depending on whether or not one is available for the specific dataset being read.

Always: This reader always provides a spatial index.

Schema Required

This row applies only to writers. For mapping files, it indicates whether or not DEF lines are required. For FME Objects applications, it indicates whether or not schema features need to be provided to the writer before data features can be written to a dataset.

Encoding Support

This row indicates whether a format supports character encoding schemes for attribute values. Users who need to translate their international data will benefit from this enhanced support as they will no longer need to set their default system language (on Windows, this is set through the "Regional and Language Options" dialog) to match the encoding of the data. Depending on the format, the character encoding may need to be specified by the user.

Transaction Support

This row applies only to readers: it indicates whether or not the native reader supports transaction processing. For mapping files, it indicates whether or not the starting transaction number and the transaction interval can be specified. For FME Objects applications, it indicates whether or not the universal writer object honors the **startTransaction**, **commitTransaction**, and **rollbackTransaction** methods.

Enhanced Geometry

Indicates whether the reader/writer supports the enhanced geometry model. The addition of enhanced geometry model support allows lines and polygons containing arcs to be maintained, rather than stroked or the geometry split up into multiple segments. It also provides the ability to truly hold measures.

Go to http://www.fmepedia.com/index.php/Geometry_Model for more information.

For more information on the global directive that sets the usage of enhanced geometry in readers and writers (as well as functions, factories and transformers), see the *FME Fundamentals* on-line help, in FME Configuration > Geometry Handling. (The FME Fundamentals help is available as a link from any help menu.)

Geometry Type Attribute

The name of the feature attribute that contains the feature's format-specific geometry type.

Supported Geometry

This table indicates which geometry types the format supports.

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	yes/no	point	yes/no
circles	yes/no	polygon	yes/no
circular arc	yes/no	raster	yes/no
donut polygon	yes/no	solid	yes/no
elliptical arc	yes/no	surface	yes/no
ellipses	yes/no	text	yes/no
line	yes/no	z values	yes/no
none	yes/no		

Raster-Specific Information

In any raster format, this table provides additional raster-specific information. See About FME Rasters for more information on each entry.

Band Interpretations	Red8, Green8, Blue8
Palette Key Interpretations	not applicable
Palette Value Interpretations	not applicable
Nodata Value	0,0,0
Cell Origin (x, y)	0.5, 0.5
Rotation Support	No
GCP Support	No
World File Support	No
TAB File Support	Yes

About Feature Attributes

Feature attributes are categorized into one of the following three groups:

1. format-specific attributes
2. user attributes
3. generic FME attributes

A feature may have one or more associated *format attributes*. A format attribute represents an attribute that is specific to a format. Some examples are: `autocad_block_name` and `sde30_justification`.

A feature is also associated with one or more associated *user attributes*, which represent custom attributes that hold domain information about a feature, such as: `parcel_identifier`, `owner_name`, `date_surveyed`, etc.

Generic FME attribute names are prefixed by `fme_`. The benefit of using generic attributes over format-specific attributes is that they have the same meaning in all the readers and writers that support them. For example, the `fme_color` and `fme_fill_color` feature attributes represent the red, green, and blue intensities of a feature. If a format has generic color support, the reader will add both the generic FME color attributes and the format-specific color attributes to features. Writers that support generic color will give precedence to the format-specific color attributes if they are present in addition to the generic attributes.

The most important generic attributes are `fme_type` and `fme_geometry`. Both of these relate to the geometry of a feature. The Geometry class represents a feature's positional information. Feature geometry may consist of points, lines, or areas. Features that contain multiple geometric parts are said to have an aggregate geometry. Features with no geometry are also supported.

The distinction between `fme_type` and `fme_geometry` is an important one: `fme_geometry` indicates the geometry type of the actual coordinates, whereas `fme_type` specifies how that geometry is to be interpreted. For example, a point geometry type can be interpreted as one of the following FME types: point, arc, ellipse, or text. The valid combinations of `fme_type` and `fme_geometry` are shown in the table below:

	<code>fme_geometry</code>	<code>fme_point</code>	<code>fme_line</code>	<code>fme_polygon</code>	<code>fme_donut</code>	<code>fme_aggregate</code>	<code>fme_no_geom</code>
<code>fme_type</code>							
<code>fme_point</code>		X				X	
<code>fme_arc</code>		X					
<code>fme_ellipse</code>		X					
<code>fme_text</code>		X					
<code>fme_line</code>			X			X	
<code>fme_area</code>				X	X	X	
<code>fme_undefined</code>							X

Format-Specific Attributes and Generic FME Attributes in Workbench

Since both generic FME attributes and format-specific attributes exist on workspace features, it is important to note that the co-existence of the two types of attributes can sometimes cause a conflict. If this happens between a reader and a writer, the generic fme attribute will take precedence.

For example, if a feature contains a format-specific color specification, and the optional `fme_color` attribute is changed *between the reader and the writer*, `fme_color` will take precedence and the format-specific color spec-

ification will be deleted from the workspace. (However, if a feature within a *writer* contains a format-specific color specification, then that will supersede `fme_color`. See the section on `fme_color` in the *FME Fundamentals* manual.)

This possible conflict also applies if you alter a feature's geometry in a workspace that has the same source and destination format. If you alter the geometry from the reader to the writer, then the generic `fme_type` will be used, and the format-specific geometry type will be deleted.

See the List of Format-Specific Attributes and Corresponding FME Generic Attributes.

List of Format-Specific Attributes and Corresponding FME Generic Attributes

The applicable format attributes and corresponding FME attributes are listed below. Any change (including deletion) to the `fme_*` attribute between a reader and writer will cause the format-specific attribute to be deleted.

Format	Format-Specific Attribute	Corresponding Generic FME Attribute
AutoCAD		<code>fme_color</code>
CGDEF	<code>cgdef_color.red</code> <code>cgdef_color.green</code> <code>cgdef_color.blue</code>	<code>fme_color</code>
Design (v7 and v8)	<code>igds_color</code> <code>igds_color.red</code> <code>igds_color.green</code> <code>igds_color.blue</code>	<code>fme_color</code>
	<code>igds_fill_color</code>	<code>fme_fill_color</code>
	<code>igds_fill_color.red</code> <code>igds_fill_color.green</code> <code>igds_fill_color.blue</code>	<code>igds_fill_color</code>
PenMetrics GRD	<code>grd_pen_color</code> <code>grd_layers_pen_color</code> <code>grd_blocks_pen_color</code>	<code>fme_color</code>
	<code>grd_brush_color</code> <code>grd_layers_brush_color</code> <code>grd_blocks_brush_color</code>	<code>fme_fill_color</code>
IDEX	<code>idex_database_color</code>	<code>fme_color</code>
	<code>idex_database_hatch_color</code>	<code>fme_fill_color</code>
MapInfo Native format (MAPINFO and MITAB)	<code>mapinfo_brush_foreground</code>	<code>fme_fill_color</code>
	<code>mapinfo_pen_color</code> <code>mapinfo_symbol_color</code>	<code>fme_color</code>
	<code>mapinfo_text_fontfgcolor</code>	

Format	Format-Specific Attribute	Corresponding Generic FME Attribute
MIF/MID (MapInfo Data Interchange Format)	mapinfo_brush_foreground	fme_fill_color
	mapinfo_pen_color mapinfo_symbol_color	fme_color
	mapinfo_text_fontfgcolor	
StruMap	strumap_red strumap_green strumap_blue strumap_color	fme_color

Simple Geometries

Simple geometries are geometries that are not composed of other geometries, and do not have associated generic FME attributes that affect their positional representation. More concretely, features with simple geometries are those with the following combinations of `fme_geometry` and `fme_type`: `{fme_line, fme_line}`, `{fme_point, fme_point}`, `{fme_polygon, fme_area}`.

Arc and Ellipsoid Geometries

Features with ellipsoid geometry are those with `fme_geometry` equal to `fme_point` and `fme_type` equal to `fme_ellipse`.

Donut Geometries

Donut Geometries are used to represent area features with holes, such as lakes with islands. A proper Donut Geometry contains only polygons that do not overlap each other or share common edges; all of the inner polygons are disjoint and fully contained within the outer polygon. However, FME does not guarantee that all features with donut geometry follow these rules. For example, when data is read from a data source that supports donut geometries but does not enforce non-intersecting and non-overlapping donuts, FME will respect the original geometry. The figure below shows an example donut geometry.

Features with donut geometry are those with an `fme_geometry` equal to `fme_donut` and an `fme_type` value equal to `fme_area`.

Aggregate Geometries

A feature with aggregate geometry has an `fme_geometry` value equal to `fme_aggregate` and an `fme_type` value equal to either `fme_point`, `fme_line`, or `fme_area`. FME uses aggregates to represent features with multi-part geometries: geometries that are composed of several disjoint pieces. In most situations, the components of an aggregate are homogeneous. That is, if `fme_type` is `fme_point`, then the aggregate contains point geometries; if `fme_type` is `fme_line` then the aggregate contains line geometries; and so on. However, your application should be designed to handle non-homogeneous aggregates gracefully since it is possible that some data sources may contain such features. Non-homogeneous aggregates have no value for `fme_type`.

OpenGIS Geometries

The OpenGIS Consortium defines a Well-Known Text (WKT) representation for feature geometries. The `FME-OFeature` object allows your application to import a geometry from WKT and export a geometry to WKT using the `importGeometryFromOGCWKT` and `exportGeometryFromOGCWKT` methods respectively.

About FME Rasters

Overview

Raster data in FME is represented by features with raster geometry. Raster data differs in several key ways from vector data, and is handled uniquely in FME. A raster can be considered as a grid of values organized into rows and columns, with the relative size of its cells determining its resolution, or level of detail. Each row and column intersection in raster grid is called a cell or pixel. Vector point geometries can often be thought of as analogous to cells, while linear geometries like roads are represented as contiguous cells. Vector features tend to be more meaningful when taken together as a group, while a single raster feature can convey the same spatial information in a non-modular fashion.

Often a raster serves as a backdrop for overlaying specific vector information. For example, you can place vector lines and polygons that represent streets and buildings on a raster image that is an aerial photograph of a city. Conversely raster may be used as a backdrop for creating vector maps. For example, one might use satellite or aerial imagery to map particular features of an urban environment for city planning or geologic structures for locating natural resources.

Rasters can be represented as either image or numeric data. Images are commonly derived from satellite data or photography, while numeric data often represents elevations, temperatures, and other quantitative information.

Raster data is stored in one or more bands whose properties may or may not be homogeneous. Imagery data often contains several bands of data. This data may come from either Optical or SAR (Synthetic Aperture RADAR) type sensors and may contain any number of image bands of data relating to a wide range of spectral bands or polarizations. Though a band may have its own specific band properties, all bands on a raster must share a common set of raster properties such as the number of rows and columns, the cell size and the ground extents.

A band may optionally have one or more palettes, also called colormaps or Look-Up Tables (LUTs), associated with it. A palette is essentially a lookup table of discrete keys to color or string values. These rasters are often referred to as classified because of the discrete data ranges. Classified rasters often serve the purpose of providing a visual representation or providing additional descriptive information for specific areas of a raster.

About FME Rasters

Overview

Raster data in FME is represented by features with raster geometry. Raster data differs in several key ways from vector data, and is handled uniquely in FME. A raster can be considered as a grid of values organized into rows and columns, with the relative size of its cells determining its resolution, or level of detail. Each row and column intersection in raster grid is called a cell or pixel. Vector point geometries can often be thought of as analogous to cells, while linear geometries like roads are represented as contiguous cells. Vector features tend to be more meaningful when taken together as a group, while a single raster feature can convey the same spatial information in a non-modular fashion.

Often a raster serves as a backdrop for overlaying specific vector information. For example, you can place vector lines and polygons that represent streets and buildings on a raster image that is an aerial photograph of a city. Conversely raster may be used as a backdrop for creating vector maps. For example, one might use satellite or aerial imagery to map particular features of an urban environment for city planning or geologic structures for locating natural resources.

Rasters can be represented as either image or numeric data. Images are commonly derived from satellite data or photography, while numeric data often represents elevations, temperatures, and other quantitative information.

Raster data is stored in one or more bands whose properties may or may not be homogeneous. Imagery data often contains several bands of data. This data may come from either Optical or SAR (Synthetic Aperture RADAR) type sensors and may contain any number of image bands of data relating to a wide range of spectral bands or polarizations. Though a band may have its own specific band properties, all bands on a raster must share a common set of raster properties such as the number of rows and columns, the cell size and the ground extents.

A band may optionally have one or more palettes, also called colormaps or Look-Up Tables (LUTs), associated with it. A palette is essentially a lookup table of discrete keys to color or string values. These rasters are often referred to as classified because of the discrete data ranges. Classified rasters often serve the purpose of providing a visual representation or providing additional descriptive information for specific areas of a raster.

Raster Properties

Rasters contain a set of metadata that defines the properties for the raster as a whole. These properties include

- number of bands (channels or layers)
- number of rows and columns (lines and pixels)
- cell size (spacing)
- cell origin
- extents
- rotation
- Ground Control Point (GCPs)

A cell is the rectangular area created in the x and y dimensions by the spacing of pixels from the raster origin.

Spacing or cell size is the fixed distance in the x and y dimensions between each pixel in the raster. Some formats store only one spacing value, meaning that it must be the same for both the x and y dimensions – this is often referred to as square cells.

The raster origin is the lower left x and y of the raster at which the coverage of the data sample begins. It contains the minimal x and minimal y values for the raster. In FME, the raster origin is the lower left corner of the lowest and left-most cell in the raster.

Cell origin is the point within each cell of a raster from which the pixel for that cell is derived. The lower left corner of the cell in the x or y dimension is 0.0, while the upper right corner is 1.0. A cell origin of 0.5 in x and 0.5 in y would put the data point for each cell in the center of the cell, which is the default representation in FME.

Extents or bounds for a raster are represented by the lower left ground coordinate and the upper right coordinate covered by the raster data. This is sometimes referred to as cell bounded. The minimum x and maximum y values that comprise the upper left corner of the raster extents are equivalent to the raster origin.

Rotation is a measure of the angle in radians of the CCW rotation of the raster from the positive x axis. The rotation point is the top left corner of the top left cell of the image. Note that currently the rotation is not applied during factory or function related processing but is merely being stored at this point. Rotation does not affect the extent values and is considered a separate property.

Ground Control Points, or GCPs, may also be present in the geometry of a raster. If present, these refer to a set of points used to georeference image or elevation data, with each point 'tying' a row and column location in the raster to an x,y location on the earth. A coordinate system will also be present in the properties of a raster containing GCPs, as opposed to being stored on the feature itself. GCPs can either be applied to the raster resulting in the image being georeferenced and tagged with the GCP coordinate system, or the GCPs can be extracted and stored on the resulting data file for those formats supporting unreferenced data and GCP storage.

Band Properties

Rasters contain a set of metadata that defines the properties for each band. These properties include the band name, number of palettes, interpretation, bit depth, 'nodata' value, and properties relating to the preferred method of data access.

The interpretation of a band describes the type of data stored at each cell in the raster and number of bits used for that type. An interpretation also implies an underlying fundamental data type used to store the data. For example, an interpretation 'Gray8' implies that the actual data type is FME_UInt8 and that the size of each cell is 8 bits. Possible interpretation values are Int8, Int16, Int32, Int64, UInt8, UInt16, UInt32, UInt64, Real32, Real64, Gray8, Gray16, Red8, Red16, Green8, Green16, Blue8, Blue16, Alpha8, Alpha16.

Many raster formats store a single data value called 'nodata', transparent or background value that represents unknown or invalid data. Often the value is at one of the extremes of the data type range. Some formats may specify a particular nodata value that is unique to that format, while others are capable of handling any single value designated as nodata.

A second option for nodata specification is a whole band or bitmask of data that acts as a flag for each cell indicating whether the cell is valid data or not. Several formats do not support nodata values at all.

Bands may also have multiple palettes associated with each instance. The number of palettes on a band can be determined as part of the band properties.

Palette Properties

Rasters contain a set of metadata that defines the properties for each palette. These properties include the palette name, key and value interpretation, key and value bitdepth, 'nodata' key and value.

Interpretation on palettes works in much the same way as it does on bands. The interpretation of a palette key must match the interpretation of the related band. Valid palette key interpretations are UInt8, UInt16, and UInt32. The palette value interpretation may be a color model such as RGB or RGBA or string data. Valid palette value interpretations are RGB24, RGBA32, RGB48, RGBA64, Gray8, Gray16, and String.

A palette does not directly store 'nodata' values however since the palette keys are intended to match the band values, which can store 'nodata', a single palette key can be interpreted as 'nodata' if it matched the band 'nodata' value. This nodata key also looks up to a palette value which is then considered the 'nodata' value. A nodata value may not exist without a nodata key.

Raster Concepts

FME Features with raster geometry have some particular features and details not always present in vector features. Some of these features include storage and formatting concerns such as compression, pyramiding, interleaving, interpretation, tiling, mosaicking, band merging and splitting, palette creation or resolution and selection.

Compression

Compression is used to reduce the size of a raster on disk, often traded for lessened performance since the format must often be uncompressed to read and compressed to write. The types of compression available depend on specific format support.

Pyramiding

Pyramids or overviews create lower resolution views of an original dataset. Often several pyramids are created at various lower resolutions to be used in the place of the original raster when only a snapshot or overview of the data is required. An example of when a pyramid is typically employed, is when a raster viewer zooms out leaving a smaller raster with less detail. Often the smaller raster is rendered using a cached pyramid instead of resampling the image to a lower resolution at the time of the zoom out request.

Interleaving

Interleaving refers to the storage of multicomponent interpretations and the order in which the individual cell values are stored together. Bands in FME use Band Sequential (BSQ) interleaving indicating that they are all stored uniquely. Palettes in FME are Band Interleaved by Pixel (BIP) as each palette value is stored together with each key in the palette.

Interpretation and Data Type

Interpretation and data type are two related concepts associated with both bands and palettes on a raster. Data type refers to the fundamental type of the data stored at each cell and is expressed as an enumeration of various floating point or signed or unsigned integer numbers. Interpretation refers to what the data type is representing. For example, a group of three UInt8 data types in each cell may correspond to an interpretation of RGB24 on a palette, such that each of the three UInt8 values corresponds to red, green and blue values respectively. The data type of a band or palette can be determined from interpretation but the converse is not true.

Palette Resolution

Rasters containing bands with palettes can be resolved to band without palettes through a process called palette resolution. During this process each band value is looked up in the palette and the resultant value is placed in the band. Once complete, the palette is removed and the band interpretation and data type are adjusted. The resultant raster has the same appearance and values as the original, except the palette is not present. Alternatively palettes can be directly removed without the palette resolution step.

Tiling and Mosaicking

Tiling and mosaicking raster refers to dividing or combining spatially related rasters. A single raster can be tiled into smaller adjacent rasters. A resultant set of tiled rasters can then be mosaicked into a single raster again. The focus is on the spatial relationship of the tiles which should fit together like pieces in a puzzle.

Band Combining and Separating

Not to be confused with mosaicking, band combining is a raster structural operation that allows for the combination of bands to form multiple rasters into one raster. The values of each band remain unchanged and the spatial relationship required is equivalent resolution and extents between all input rasters. This is useful in situations such as when one wants to combine three individual one band rasters combined into a single three band raster. Conversely, separating a raster with multiple bands and palettes is also supported and can be employed to write multi-band or multi-palette rasters to destination formats that support only single-band or single-palette output.

Band and Palette Selection

Rasters that contain multiple bands and/or palettes need not be split to be operated on individually. FME allows for individual band and palette selection for operational purposes. For example, an RGB raster that has three bands can have only the red band selected such that subsequent processing occurs only on the red band, and does not affect the remaining green and blue bands. Bands and palettes are selected based on their numeric location in the raster. The numerical relationship is zero based such that the first band is at index 0, the second band at index 1 and so on.

Raster Processing

A secondary set of features specific to processing rasters also exists and is expressed through the variety of raster functions and factories inside FME. Please see the various transformers in the Raster category or search for *raster* in the transformer search box.

Raster versus Vector Features

FME Features with raster geometry cannot be processed in all the ways that vector features can. If an operation that is not yet supported for a raster is attempted, a vector FME polygon feature is used instead. This substitute feature represents the original raster bounding box, and contains the original attributes.

Raster-to-vector data translation and vector-to-raster data translation is not an automatic process. There are factories through which vector data can be transformed into raster data, and raster data into vector points. There currently do not exist any means of stroking or transforming raster data into non-trivial vector output through FME.

FME features with raster geometry each typically represent one raster data file. Raster writers typically accept a directory as a destination dataset. When writing multiple raster files for one dataset directory the feature type name is used to determine the filename. If multiple features are written to the same dataset the name will be suffixed to be unique. Some writers use feature type fanout by default on the `fme_basename` attribute on each feature, thus providing a name with respect to the source data and having some degree of uniqueness.

Raster File Naming

Many file-based raster writers use the feature type as the output filename. For example, if you passed a feature to the feature type "image" on the TIFF writer, the output would be "image.tif".

When used in Workbench, most file-based raster format writers fanout on `fme_basename`. When this is the case, the feature type will effectively be the value of the `fme_basename` attribute, which is set by all raster format readers to be the filename without the path or extension. For example, if you read two files, `image1.tif` and `image2.tif`, two features

would be produced, one with an `fme_basename` value of "image1", and one with a value of "image2". Then, if these two features were written to a writer when fanning out on `fme_basename`, two new files would be produced, e.g. `image1.png` and `image2.png`.

Raster format writers that store their data in files require a mechanism to avoid overwriting existing files and differentiate output file from one another when multiple rasters are written to a writer (particularly if the writer outputs one file per raster feature). Raster file based writers implement a simple renaming mechanism to deal with name collisions. Renaming the output files only occurs within a single instance of the writer within a given translation.

The first output file is written using the name requested in the workspace. If additional files are produced from the same feature type, the subsequent files are automatically distinguished by appending sequential numbers to the filenames. For example, if four rasters are written to the same feature type, named "image", the result is a set of output files with the names `image.tif`, `image_1.tif`, `image_2.tif`, and `image_3.tif`.

Multiple translations of the same workspace that incorporates a file based raster writer will overwrite previous file output if name collisions occur. Similarly, using multiple writer instances targeted at the same directory is considered unsafe if the same feature types are used in both translations since data overwriting may occur.

World Files

World files are used to store georeferencing information for rasters. More specifically, they describe the origin, spacing, and rotation of a raster.

Several raster format readers will read world files present alongside a dataset, and many raster writers have the option to generate a world file to accompany the output dataset. Consult individual format documentation for more information on their world file support.

Raster format readers will give world file georeferencing more precedence than georeferencing in the raster dataset. That is, if the world file stores georeferencing information that is different than that from the source dataset, it is the world file georeferencing that will be applied to the raster. If this is not desired, a simple workaround is simply to move or rename the world file so it will not be read by the format reader. Additionally, note that readers that read both world and TAB files will give more precedence to the world file.

Also note that most raster format writers will not write a world file if the output raster contains only default georeferencing information: an origin of (0, 0), spacing of 1.0, and rotation of 0.0.

MapInfo TAB Files

Raster TAB files are used to store control points, a coordinate system, and user attributes.

Most raster format readers will read TAB files present alongside a dataset, and most raster format writers have an option to generate a TAB file to accompany the output dataset. Consult individual format documentation for more information on their TAB file support.

The control points represent georeferencing information for the raster. When reading this information, FME will attempt to determine if these control points represent the extents of the raster (i.e. they occur at the corners of the raster) or if they are Ground Control Points, and apply this information accordingly. Note that georeferencing information will be discarded for datasets that contain multiple subdatasets, as there is no way to ascertain which subdataset the information corresponds to.

Attributes are not natively a part of raster TAB files. However, FME will read and write attributes to raster TAB files in the same manner as is done for vector TAB files. This enables the storage of user attributes for many formats that don't otherwise support attribution.

Information in TAB files will be given more precedence than information in the raster dataset. For example, if the TAB file stores georeferencing information that is different than that from the source dataset, it is the TAB file georeferencing that will be applied to the raster. If this is not desired, a simple workaround is simply to move or rename the TAB file so it will not be read by the format reader. Additionally, note that readers that read both world and TAB files will give more precedence to the world file.

Database Writer Mode

Overview

Most database writers share a WRITER_MODE specification, which indicates the default operations that will be performed by the writer.

Note: Some writers implement only portions of these specifications (for example, the ArcSDE writer and Geo-database writers do not implement the Table Level mode).

Database Writer Mode

Overview

Most database writers share a WRITER_MODE specification, which indicates the default operations that will be performed by the writer.

Note: Some writers implement only portions of these specifications (for example, the ArcSDE writer and Geo-database writers do not implement the Table Level mode).

Generic Database Writer Mode

The writer mode can be specified at three unique levels:

- Writer
- Table
- Feature

Writer Level

At the writer level, the WRITER keyword is `<writer>_MODE`, prefixed by the writer keyword (for example, `SDE30`). Possible values are:

- INSERT (default) – Implies insert only; can be overridden only by table-level modes, not feature-level modes.
- UPDATE – can be overridden by table and feature level modes.
- DELETE – can be overridden by table and feature level modes.

Table Level

At the table level, there is a database-specific feature type DEF parameter called `_mode` prefixed by the writer keyword (for example, `SDE30_MODE` or `postgis_mode`). Possible values for this attribute are:

0. INHERIT_FROM_WRITER - default
1. INSERT
2. UPDATE
3. DELETE

Feature Level

At the feature level, there is an FME generic attribute called `fme_db_operation`. Possible values for this attribute are:

0. <no attribute> - defaults to table level mode
1. INSERT
2. UPDATE
3. DELETE

The FME generic attribute overwrites the value given to the writer keyword `<WRITER>_MODE` for that feature only regardless of the value of the table mode, except for when the table level mode is INSERT.

Used in conjunction with the `fme_where` and the `fme_db_transaction` attributes for updates.

Feature Level Generic Examples

Insert example:

```
WRITER LEVEL: UPDATE
TABLE LEVEL: UPDATE, INSERT or DELETE
feature type roads
num_lanes 5
surface_type gravel
age 106
location canada
condition poor
name Highway 1
road_id 1234
fme_geometry fme_no_geom
fme_db_operation INSERT
```

This will insert a row into a table named "roads". This will append to an existing table. Columns not specified will receive their default values, if there are default values.

Update example:

```
WRITER LEVEL: UPDATE
TABLE LEVEL: UPDATE or DELETE
feature type roads
condition good
fme_db_operation UPDATE
fme_where road_id = 1234
```

This will update the row in the table "roads" where the road_id = 1234. The column "condition" will have its value changed from "poor" to "good".

Delete example:

```
WRITER LEVEL: UPDATE
TABLE LEVEL: UPDATE or DELETE
feature type roads
fme_db_operation DELETE
fme_where road_id = 1234
```

This will delete the row in the table "roads" where the road_id = 1234.

Feature Level Specific Format Examples (PostGIS)

Insert example:

```
WRITER LEVEL: UPDATE
TABLE LEVEL: UPDATE, INSERT or DELETE
feature type roads
num_lanes 5
surface_type gravel
age 106
location canada
condition poor
name Highway 1
road_id 789
fme_geometry fme_point (45, 67)
fme_db_operation INSERT
```

This will insert a row into a table named "roads". This will append to an existing table. Columns not specified will receive their default values, if there are default values.

Update example:

```
WRITER LEVEL: UPDATE
TABLE LEVEL: UPDATE or DELETE
feature type roads
fme_geometry fme_point (100234, 2349)
fme_db_operation UPDATE
fme_where road_id = 789
```

This will update the row in the table "roads" where the road_id = 789. The geometry column will have its value changed from (45, 67) to (100234, 2349).

Notes:

- Table level mode specification overrides writer level mode specification.
- Feature level mode specification overrides table level mode specification when the table level mode is NOT "INSERT".
- Table level mode specification defaults to INHERIT_FROM_WRITER.
- Updates performed on rows that DO NOT EXIST are NOT turned into inserts. The user is warned and the feature is skipped.
- Inserts performed on rows that EXIST are NOT turned into updates. FME will still attempt to perform the insert: if it is not prevented by a unique index, it will insert a duplicate row; if it is prevented by a unique index, the translation will halt with an error.
- Update features are formed in one of two common ways:
 - Using a stripped down source feature to update one row per update feature
 - Using a new feature and adding the **fme_db_operation** and **fme_where** and any attributes that you want to update.
- Updates are not limited to one row per feature, but can update the entire table if desired. It is mandatory to use the **fme_where** attribute to specify which rows to update (or delete).
- Geometry can also be updated using update mode. The geometry on the update feature will replace the geometry in all of the matched rows. However, an update feature with no geometry will not change the geometry column in any way.
- Some formats that allow UPDATE and DELETE mode using the **fme_where** attribute require that the attribute to be specified to identify which rows to operate on, and will fail if the **fme_where** attribute is not present (for example, PostGIS, MySQL).

Feature Selection

Once you have determined how to specify whether an insert, update, or delete should be performed, you will need to select the features in the database to be updated or deleted.

Note: This section is irrelevant for features being inserted.

Terminology

- *edit*: an insert or update
- *query feature*: the FME feature passed to the writer, which will be used to determine which features from the database are edited

Methods

Currently, there are two ways in which to determine which features to edit:

1. **fme_where** – This is found on the query feature, and can be any valid WHERE clause (it does not contain the word WHERE). The value for this keyword is not meant to be parsed and split up by the writer, but passed

directly to the database. The features returned from using this attribute are the same features that would be returned from executing the SQL query:

```
SELECT * FROM <feature type of query feature> WHERE <fme_where>;
```

2. **<format_name>_update_key_columns** – This specifies which columns to use in creating a WHERE clause. It is found on the DEF line to which it applies. The values for those columns are then taken from the query features. For example, the DEF for the table roads would look like:

```
GEODATABASE_MDB_DEF roads  
GEODATABASE_UPDATE_KEY_COLUMNS name,location,age
```

The query feature might look like:

```
feature type roads  
num_lanes 5  
surface_type gravel  
age 106  
location canada  
condition poor  
name Highway 1  
road_id 1234
```

The WHERE clause that the writer assembles would then be:

```
name = 'Highway 1' AND location = 'Canada' AND age = 106
```

(notice the use of single quotes around the text columns).

Some writers (for example, ArcSDE and Geodatabase) use the object ID column as the default key column if the user did not specify any of their own. (However, since we don't know what the object ID column is when the workspace/mapping file is generated, a blank value is assigned to the parameter and the writer interprets this as "use the object ID column".)

It should not be mandatory that the columns/attributes specified for this parameter exist on the DEF line, because if the table exists, attributes are optional on the DEF line.

Limitations

Although a single query feature may cause more than one feature to be edited, the update/delete feature is restricted to changing only those features of the same feature type as it. This means that we cannot have an update feature with a feature type of "provinces" updating features of feature type "cities". In general a feature type in a database format corresponds to a table; however, it may correspond to a view.

Date and Text Fields

It will probably be necessary to place single quotes (') around date and text fields when creating your own WHERE clause using the columns in

```
<format_name>_update_key_columns
```

Or, if you don't need to put single quotes around date fields, you may have to convert the date from the FME format into the database-specific format. Either way, it will be necessary to know the column data types.

Attribute Selection

In update mode, only the attributes on the feature will be used to update the existing columns. If an attribute is missing on the feature, the associated column is not updated in the database. Similarly, if the feature has no geometry, any geometry associated with the row in the database will remain unchanged.

1Spatial Internal Feature Format (IFF) Reader/Writer

Format Notes: This format is not available in FME Base Edition.

The 1Spatial (previously known as Laser-Scan) Internal Feature Format (IFF) Reader/Writer module enables FME to read and write IFF files. The IFF is an internal ASCII format originally created by Laser-Scan Ltd. This chapter assumes that you are familiar with the Internal Feature Format.

Overview

IFF files store both geometry and attribution for features. An IFF file has the following file name extension:

File Name Extension	Contents
.iff	Vector geometric data

The extension is added to the basename of the IFF file.

The Internal Feature reader supports the types of symbol, line, polygon, and text geometric data in **.iff** files. The IFF format also stores features with no geometry. Features that have no geometry are referred to as having a geometry of *none*. IFF files support both two- and three-dimensional geometry.

IFF Quick Facts

Format Type Identifier	IFF
Reader/Writer	Both
Licensing Level	Professional
Dependencies	None
Dataset Type	Directory or File
Feature Type	File base name
Typical File Extensions	.iff
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	No
Transaction Support	No
Geometry Type Attribute	iff_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	no
circular arc	no		raster	no
donut polygon	no		solid	no
elliptical arc	no		surface	no
ellipses	no		text	yes
line	yes		z values	no
none	yes			

Reader Overview

The IFF reader produces FME features for all feature data held in IFF files that are in the specified directory.

The reader first scans the given directory specified by the **DATASET** keyword. For each IFF file found, it checks to see if that file is requested in the translation by comparison with the list specified by the ID's keyword. Then the reader extracts features from an IFF file one at a time, and passes them on to the rest of the FME for further processing. When the file is exhausted, the IFF reader moves onto the next file in the directory.

Optionally a single IFF file can be specified. In this case, only that IFF file is read.

Reader Directives

The directives processed by the IFF reader are listed below. The suffixes shown are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the IFF reader is **IFF**.

DATASET

Required/Optional: *Required*

The value for this directive contains the directory where the IFF files will be read, or the file path to the single IFF file. A typical mapping file fragment specifying an input IFF dataset looks like:

```
IFF_DATASET /usr/data/iff/iffFile.iff
```

Workbench Parameter: *Source 1 Spatial Internal Feature Format (IFF) File(s)*

DEF

Required/Optional: *Required*

Each IFF file may optionally be defined before it is read. The definition specifies the base name of the file, and the names and the types of all attributes. The syntax of an IFF **DEF** line is:

```
<ReaderKeyword>_DEF <baseName> \
[<attrName> <attrType>]+
```

The file names of the physical IFF files are constructed by using the directory specified by the **DATASET** keyword, the basename specified on the IFF **DEF** lines, and the **.iff** extension.

The following table shows the attribute types supported.

Field Type	Description
char(<width>)	Character fields store fixed-length strings. The width

Field Type	Description
	parameter controls the maximum number of characters that can be stored by the field. No padding is required for strings shorter than this width.
date	Date fields store dates as character strings with the format YYYYMMDD.
number(<width>, <decimals>)	Number fields store single and double precision floating point values. The width parameter is the total number of characters allocated to the field, including the decimal point. The decimals parameter controls the precision of the data and is the number of digits to the right of the decimal.
smallint	Small integer fields store 16-bit signed integers and therefore have a range of -32767 to +32767.
integer	Integer fields store 32-bit signed integers.
logical	Logical fields store TRUE/FALSE data. Data read or written from and to such fields must always have a value of either true or false.
float	Float fields store 4-byte floating point values. There is no ability to specify the precision and width of the field.
double	Double fields store 8-byte floating point values.

The following mapping file fragment defines a IFF file. Notice that the definition specifies the geometric type of the entities it will contain since IFF files may contain any of the valid geometry types.

```
IFF_DEF landcover \
  area number(12,3) \
  landcoverType char(11) \
  perimeter float
```

IDs

Required/Optional: *Optional*

Contains a list of IFF files to process. If more IFF files were in the directory, they are ignored. If this is not specified, then all defined IFF files in the directory are read.

This specification is used to limit the available and defined IFF files read. If no IDs are specified, then all defined and available IFF files are read. The syntax of the **IDS** keyword is:

```
<ReaderKeyword>_IDS      <baseName1> \
<baseName2> ... \
<baseNameN>
```

The file **IDS** must match those used in DEF lines.

The example below selects only the **iff_data** IFF file for input during a translation:

```
IFF_IDS iff_data
```

APPLY_ORIGIN_OFFSET

Required/Optional: *Optional*

This directive specifies whether or not to apply the origin offset found in Type 2 Map Descriptor record, to all the features.

Values: *yes* | *no*

Default: *no*

Workbench Parameter: *Apply Origin Offset*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

*** Workbench Parameter**

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>

*** Workbench Parameter**

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The IFF Writer creates and writes feature data to IFF files in the directory specified by the directive **DATASET**. Any existing IFF files are overwritten with a new IFF file. As features are routed from the FME to the IFF writer, the writer determines which file the features are to be written to and outputs them accordingly. More than one IFF file can be written in one translation.

Writer Directives

The IFF writer processes the **DATASET** and **DEF** directives as described in the *IFF Reader Overview's Reader Directives* subsection. It does not use the **IDS** directive.

DATASET

Required/Optional: *Required*

Workbench Parameter: *Destination 1 Spatial Internal Feature Format (IFF) Directory*

DEF

Required/Optional: *Required*

Feature Representation

IFF features consist of geometry and attributes of feature. The attribute names are defined in the **DEF** line and there is a value for each attribute in each IFF feature. In addition, each IFF feature contains several special attributes to hold the type of the geometric entity and its display parameters. All IFF features contain the **iff_type** attribute, which identifies the geometric type.

Depending on the geometric type, the feature contains additional attributes (in addition to the generic FME feature attributes that FME Workbench adds to all features [see *About Feature Attributes*]) specific to the geometric type. These are described in subsequent sections.

The general attributes of all IFF geometric features are described in the table below.

Attribute Name	Contents
iff_type	<p>The IFF geometric type of this entity.</p> <p>Range: iff_symbol iff_line iff_polygon iff_text iff_none</p> <p>Default: No default</p>
iff_history	<p>The history of the IFF file.</p> <p>Range: Maximum of 256 characters Default: Blank</p>
iff_map_area	<p>The map area.</p> <p>Range: Maximum of 256 characters Default: No default</p>
iff_map_grid	<p>The map grid representation</p> <p>Range: Maximum of 256 characters Default: No default</p>
iff_map_scale	<p>The map scale.</p> <p>Range: Maximum of 256 characters Default: No default</p>
iff_map_origin_offset	<p>The map's origin offset or local origin. (Read-only)</p> <p>Range: Maximum of 256 characters Default: No default</p>
iff_map_projection	<p>The map's projection as a number. (Read-only)</p> <p>Range: Integer Default: No default</p>
iff_map_spheroid	<p>The map's spheroid as number. (Read-only)</p> <p>Range: Integer Default: No default</p>
iff_map_units	<p>The map's unit. (Read-only)</p> <p>Range: Integer Default: No default</p>

Attribute Name	Contents
iff_map_proj_stat	The map projection status. Range: Maximum of 256 characters Default: No default
iff_map_aux_grid	The map auxiliary grid. Range: Maximum of 256 characters Default: No default
iff_cubic_coef{#}	The matrix for coordinate transformation. Each list attribute of this type contains 2 numbers that are part of the transformation matrix for the feature. Range: Maximum of 256 characters. Default: No default
iff_ctrl_pt_nw	The Northwest control points. Range: Maximum of 256 characters Default: No default
iff_ctrl_pt_sw	The Southwest control points. Range: Maximum of 256 characters Default: No default
iff_ctrl_pt_se	The Southwest control points. Range: Maximum of 256 characters Default: No default
iff_ctrl_pt_ne	The Northeast control points. Range: Maximum of 256 characters Default: No default
iff_sec_descr	The section description. Range: Maximum of 256 characters Default: Blank
iff_layer_num	The layer number of the feature. (0 is a reserved layer number that is often ignored) Range: 0...32767 Default: 1
iff_layer_stat	The layer status flag (currently not used). Range: Integer Default: 0
iff_layer_ptr	The layer's pointer to location of matching end of layer marker. Range: Maximum of 256 characters Default: Blank
iff_serial_num	The feature serial number. This number is often the

Attribute Name	Contents
	<p>same as the iff_seq_num. Range: 0...65535 Default: No default</p>
iff_seq_num	<p>The feature internal sequence number. This number is unique and corresponds with creation order. Range: 0...65535 Default: 1 and increment for each new feature</p>
iff_feat_code	<p>The feature code number. Range: 0...32767 Default: 0</p>
iff_feat_stat	<p>The feature status. Range: 0...32767 Default: 0</p>
iff_proc_code	<p>The feature type or process code. The last two bits specify the feature type (0 = line, circle, area or symbol string feature, 1 = symbol feature, 2 = text feature, and 3 = value reserved). Range: 0...32767 Default: 0</p>
iff_user_word	<p>The user defined word, this is a reserved field for use by users. Range: Smallint Default: No default</p>
iff_anc_code{#}.type	<p>The ancillary code type. Range: 0...32767 Default: No default</p>
iff_anc_code{#}.value	<p>The ancillary code value. Range: Maximum of 256 characters Default: No default</p>
iff_anc_code{#}.text	<p>The ancillary code text. Range: Maximum of 255 characters Default: No default</p>
iff_anc_code	<p>The comma-separated ancillary code list. Range: Maximum of 256 characters Default: No default</p>
iff_pen_stat	<p>The pen status. Range: 0 - pen up, 1 - pen down Default: 0</p>

Attribute Name	Contents
iff_junc_blk{#}.sec_num	The junction block section number. Range: integer Default: No default
iff_junc_blk{#}.next_jb	The junction block pointer to the next junction block. Range: Maximum of 256 characters Default: No default
iff_junc_blk{#}.offset	The junction block offset number. Range: Maximum of 256 characters Default: No default
iff_junc_blk{#}.arms_num	The junction block number of arms. Range: Maximum of 256 characters Default: No default
iff_junc_blk{#}.x_coord	The junction block x-coordinate. Range: Double Default: No default
iff_junc_blk{#}.y_coord	The junction block y-coordinate. Range: Double Default: No default
iff_junc_blk{#}.pnt_no	The junction block string vertex number. Range: Integer Default: No default
iff_junc_blk{#}.addr	The junction block address location of the arm coordinates. Range: Maximum of 256 characters Default: No default
iff_junc_blk	The comma separated junction block list. (Nested junction block lines are separated by '\s'). Range: Maximum of 256 characters Default: No default
iff_junc_ptr{#}.offset	The junction pointer junction block offset. Range: Maximum of 256 characters Default: No default
iff_junc_ptr{#}.addr	The junction pointer junction block address. Range: Maximum of 256 characters Default: No default
iff_junc_ptr	The comma separated junction pointer list. Range: Maximum of 256 characters Default: No default

Attribute Name	Contents
iff_void_size	The size of the void. Range: Integer Default: No default

Symbol

iff_type: iff_symbol

IFF symbol features are point features that specify a single x and y coordinate in addition to any associated user-defined attributes.

There are no special FME attribute names used to control the IFF symbol settings.

Lines

iff_type: iff_line

IFF line features specify linear features defined by a sequence of x and y coordinates.

The following table lists the special FME attribute names used to control the IFF line settings.

Attribute Name	Contents
iff_size	The line thickness. Range: Integer Default: No default

Text

iff_type: iff_text

IFF text features are used to specify annotation information. Although IFF files can have features that have more than one set of annotation information, only simple text features can be written. The IFF reader is able to read features with more than one set of annotation information by splitting them into separate features. But the IFF writer will not merge these separated features when writing, only features with only a set of annotation information are written.

The following table lists the special FME attribute names used to control the IFF text settings.

Attribute Name	Contents
iff_rot	The text label's rotation. Range: 0.00...360.00 Default: 0
iff_text_string	The text label. Range: Maximum of 256 characters Default: Blank
iff_size	The text size. Range: Integer Default: No default
iff_text_code	The text code value. Range: Integer Default: No default

Attribute Name	Contents
iff_text_cmpnt	The text component. Range: Integer Default: No default
iff_text_res1	Reserved space for future use. Range: Maximum of 256 characters Default: No default
iff_text_res2	Reserved space for future use. Range: Maximum of 256 characters Default: No default

Adobe 3D PDF Writer

Format Notes:

This format is not available in FME Base Edition.

The 3D PDF Writer enables FME to write Adobe® Portable Document Format (PDF) files embedded with interactive 3D annotations.

Overview

The Adobe Reader software version 7.0 and above has included support for interactive 3D annotations in PDF files. These annotations allow users to visualize 3D models. For example, users can view the models from different angles and select sub-elements of the model by picking them with the mouse.

The writer represents the 3D models in the ECMA-363 Universal 3D File Format, which become embedded in a PDF document. The 3D model can be viewed by PDF viewer applications that support PDF's interactive 3D annotations.

PDF Quick Facts

Format Type Identifier	PDF
Reader/Writer	Writer
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	U3D Node
Typical File Extensions	PDF
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	Never
Enhanced Geometry	Yes
Geometry Type Attribute	pdf_type

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	yes	point	yes
circles	no	polygon	yes
circular arc	no	raster	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
donut polygon	yes		solid	yes
elliptical arc	no		surface	yes
ellipses	no		text	no
line	yes		z values	yes
none	yes			

Writer Overview

The writer outputs PDF version 1.7 files. The document will have one page which contains the 3D annotation.

The 3D model has a hierarchal structure of Nodes, which are elements of the model. Feature types become Nodes with no geometry. Features become Nodes that may have geometries and attributes. Feature Nodes are children of their corresponding Feature Type Node.

Writer Directives

The directives that are processed by the PDF writer are listed below. The suffixes shown are prefixed by the current `<WriterKeyword>_` in a mapping file. By default, the `<WriterKeyword>` for the PDF writer is `PDF`.

DATASET

Required/Optional: *Required*

The value for this directive is the path to the output file. If the output file does not exist, then the writer will create a new file. If the output file exists, then the writer will overwrite it. If other applications have the output file opened, then the writer will be unable to continue and the translation will fail.

Workbench Parameter: *Destination PDF File*

DEF

Required/Optional: *Required*

The PDF writer uses `PDF_DEF` lines to define feature types. A typical mapping file fragment specifying a feature type looks like:

```
PDF_DEF <featureName> \  
    [<attributeName> <attributeType>]*
```

The configuration parameters present on the definition line are described in the following table:

Parameter	Contents
featureName	This declares the name of the feature type.
attributeName	This declares the name of an attribute. The maximum length of attribute names is 200 characters.
attributeType	This declares the type of the attribute. The only valid attribute type is string.

CENTER_COORDINATES

Required/Optional: *Optional*

This directive specifies whether the coordinates of all features should be normalized to the numerical range [-0.5, 0.5]. The PDF format stores coordinates in single precision format. Translating datasets that stores coordinates in double precision to PDF without normalizing the coordinates may result in severe visual artifacts.

Values: *YES* | *NO*

Default Value When Keyword Not Specified: *YES*

Default for New Workspaces/Mapping Files: *YES*

Workbench Parameter: *Center coordinates at the origin*

BACKGROUND_COLOR

Required/Optional: *Optional*

This directive specifies the background color of the 3D annotation when the output file is viewed with Adobe Acrobat. The format of the value is a comma delimited list of red, green, and blue components of the desired background color. Each rgb (red green blue) value should be a real number between 0.0 and 1.0, inclusive. The default value for this directive is *0.2,0.2,0.2*, which is a dark grey color.

Values: *<0.0 ... 1.0>*, *<0.0 ... 1.0>*, *<0.0 ... 1.0>*

Default Value When Keyword Not Specified: *0.2,0.2,0.2*

Default for New Workspaces/Mapping Files: *0.2,0.2,0.2*

Workbench Parameter: *Background color*

PAGE_SIZE

Required/Optional: *Optional*

This directive specifies the size of the output page of the PDF document. The 3D annotation will fill the entire page leaving a slight margin on all sides. The default value for this directive is *600 600*, which specifies a page size of 600 by 600 pixels.

Values: *<0.0 ... >* *<0.0 ... >*

Default Value When Keyword Not Specified: *600 600*

Default for New Workspaces/Mapping Files: *600 600*

Workbench Parameter: *Page size*

NODE_CREATION_LIST

Required/Optional: *Optional*

This directive can be used to quickly and conveniently create empty group nodes in the scene graph. The nodes created by this directive can be used as the targets of the `pdf_parent_uid` format-specific feature attributes. The format of the value is a comma delimited lists of node specifiers. Node specifiers are a period delimited list of a node's ancestry, starting with the root UID and ending with the leaf UID. For example, the directive value `'Node1.Node2.Node3,Node1.Node4'` creates 4 nodes altogether with the following node hierarchy:

```
Node1 <--- Node2 <--- Node3
      ^--- Node4
```

Values: *<name>[.<name>]*[,<name>[.<name>]*]**

Default Value When Keyword Not Specified: *<empty string>*

Default for New Workspaces/Mapping Files: *<empty string>*

Workbench Parameter: *Node creation list*

DISPLAY_NAV_UI

Required/Optional: *Optional*

This directive controls whether the the Adobe Acrobat software will display the left-hand side Node navigation UI by default when opening the ouput PDF file.

Values: *YES* | *NO*

Default Value When Keyword Not Specified: *NO*

Default for New Workspaces/Mapping Files: *NO*

Workbench Parameter: *Display Navigation UI*

2D_FEATURE_HANDLING

Required/Optional: *Optional*

This directive controls whether features with no Z coordinates will have their normals adjusted such that the feature is visible immediately after opening the PDF file. If the value is YES, all 2D features will be visible from the default camera position after opening the PDF file in Adobe Acrobat software. If the value is NO, the geometries will not be adjusted and will be written as-is.

Values: *REORIENT* | *AS_IS*

Default Value When Keyword Not Specified: *AS_IS*

Default for New Workspaces/Mapping Files: *REORIENT*

Workbench Parameter: *Reorient 2D features for visibility*

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

PDF features consist of geometry and attributes. The attribute names are defined in the DEF line and there is a value for each attribute in each PDF feature. In addition, each PDF feature contains several special attributes to hold the type of the geometric entity and its display parameters. All PDF features contain an **pdf_type** attribute, which identifies the geometric type. Depending on the geometric type, the feature contains additional attributes specific to the geometric type. These are described in subsequent sections.

Geometries with no Z coordinates (2D geometries) will be assigned zero as their z values.

The following format specific attributes are applicable to all geometry types, and these attributes do not appear as user attributes in the output data:

Attribute Name	Contents
pdf_uid	This is an optional format attribute that can be used to specify the unique Node name of the feature. If this attribute is not set, then an unique name is autoatically generated and assigned. If this attribute is set and each feature is not given an unique value, then undefined behaviour will result.
pdf_parent_uid	This is an optional format attribute that can be used to specify the parent feature Node for this feature Node. The value for this attribute should correspond to a value given to the pdf_uid attribute in a different feature.
pdf_child_uid	This is an optional format attribute that can be used to specify the child feature Node for this feature Node. The value for this attribute should correspond to a value given

Attribute Name	Contents
	to the pdf_uid attribute in a different feature.
pdf_merge_tolerance	This is an optional format attribute that can be used to set the numerical tolerance used to merge spatially close vertices. If the value is 0.0 or if the attribute is unset, then no merging will be done. Otherwise, if a positive real number less than 1.0 is specified, then vertices that lie within the maximum extent of the scene divided by the number specified will be merged together. For example, if this attribute is set to 0.01, and the maximum extent of the scene is [0,2500] then vertices that are within a distance of 25 units of each other will be merged together.
pdf_ambient_color	This is an optional format attribute that can be used to specify the ambient component of the feature's material according to the Phong lighting model. The format of this attribute is R,G,B where each color component is a real value in the range [0,1]. For example, a value 0,1,0 specifies a green ambient color for the feature.
pdf_diffuse_color	This is an optional format attribute that can be used to specify the diffuse component of the feature's material according to the Phong lighting model. The format of this attribute is R,G,B where each color component is a real value in the range [0,1]. For example, a value 0,1,0 specifies a green diffuse color for the feature.
pdf_specular_color	This is an optional format attribute that can be used to specify the specular component of the feature's material according to the Phong lighting model. The format of this attribute is R,G,B where each color component is a real value in the range [0,1]. For example, a value 0,1,0 specifies a green specular color for the feature.

Points

pdf_type: pdf_point

PDF point features specify Nodes that is a collection of points. The points are rasterized according to the rendering mode of the viewer application.

Lines

pdf_type: pdf_line

PDF line features specify Nodes that is a collection of linear line segments. The line segments may be disjoint.

Circular and elliptical arc segments will be stroked into linear line segments. Lines have no area or volume, and will appear as rasterized according to the rendering mode of the viewer application.

Mesh

pdf_type: pdf_mesh

PDF mesh features specify Nodes with 3D meshes. Meshes are composed of triangular faces. If the input mesh contains faces with more than three distinct vertices, then the face will be converted into multiple triangular faces. The triangular faces of a mesh need not be connected.

Faces are one-sided: they are only visible from one view direction. A face is visible when its normal points toward the observer. If the vertices of the outer boundary of the face are observed to be in anti-clockwise order, then the normal of the face points toward the observer, implying that the face is visible.

Polygons and donuts are treated as meshes. They will be converted into triangular faces that represents the inner area of the polygon or donut.

Textures are supported. The texture coordinates can be specified through the `fme_texture_coordinate_u`, `fme_texture_coordinate_v`, `fme_texture_coordinate_w`, and `fme_texture_coordinate_q` point measures.

Collection

pdf_type: pdf_collection

PDF collection features specify a parent Node with no geometry but with the feature's attribute values, and child Nodes for each element of the collection. The child Nodes do not have the feature's attribute values. Child nodes can be of any geometry type.

Adobe Geospatial PDF Writer

Format Notes:

This format is not available in FME Base Edition.

The PDF2D Writer enables FME to write Adobe® Portable Document Format (PDF) with vector drawings and geospatial information.

Overview

PDF is a document exchange format created by Adobe Systems.

The PDF2D writer will write features with 2D geometry as vector drawings on a page of a PDF document. The output PDF file can be viewed with Adobe Acrobat Reader or any other PDF viewer application.

Features will belong to a layer according to its feature type. Feature attribute can be queried using the analysis tools of the Adobe Acrobat Reader software. If features have a coordinate system defined, then geospatial coordinates of the cursor location can also be displayed.

PDF Quick Facts

Format Type Identifier	PDF2D
Reader/Writer	Writer
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	Layer
Typical File Extensions	PDF
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	Yes
Spatial Index	Never
Schema Required	Yes
Transaction Support	Never
Enhanced Geometry	Yes
Geometry Type Attribute	pdf_type
Encoding Support	Yes

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	yes	point	yes

Geometry Support				
Geometry	Supported?		Geometry	Supported?
circles	yes		polygon	yes
circular arc	yes		raster	no
donut polygon	yes		solid	no
elliptical arc	yes		surface	no
ellipses	yes		text	yes
line	yes		z values	no
none	yes			

Writer Overview

The writer outputs PDF version 1.7 files. The document will have one page and features will be drawn in a rectangular region of the page called the viewport. Measurements on the page use the unit of a typographical point. Also known as a PostScript point, it is defined as 1/72 of an inch on the output page.

If attribution is written, then each feature and feature type will be represented by a logical structure element. In Adobe Acrobat Reader, features can be visually picked using the Object Data tool.

Features with unsupported geometry types will not be drawn, but their attribution data will still be written.

Features will be grouped into layers according to their feature types. In Adobe Acrobat Reader, the visibility of layers can be toggled.

PDF files can be opened through a command or an URL that specifies what and how the contents are displayed.

For more details about this feature, see this external documentation: http://www.adobe.com/devnet/acrobat/pdfs/pdf_open_parameters.pdf

Writer Directives

The directives that are processed by the PDF2D writer are listed below. The suffixes shown are prefixed by the current `<writerKeyword>`_ in a mapping file. By default, the `<writerKeyword>` for the PDF2D writer is `PDF2D`.

DATASET

Required/Optional: *Required*

The value for this directive is the path to the output file. If the output file does not exist, then the writer will create a new file. If the output file exists, then the writer will overwrite it. If other applications have the output file opened, then the writer will be unable to continue and the translation will fail.

Workbench Parameter: *Destination PDF File*

DEF

Required/Optional: *Required*

The PDF2D writer uses `PDF2D_DEF` lines to define feature types. A typical mapping file fragment specifying a feature type looks like:

```
PDF2D_DEF <featureName> \
  [pdf_layer_order <layerOrder>]? \
  [pdf_in_page_coordinates <pageCoordinates>]? \
  [pdf_default_opacity <opacity>]? \
  [pdf_layer_visibility <visibility>]? \
  [<attributeName> <attributeType>]*
```

The configuration parameters present on the definition line are described in the following table:

Parameter	Contents
featureName	This declares the name of the feature type.
attributeName	This declares the name of an attribute. The maximum length of attribute names is 200 characters.
attributeType	This declares the type of the attribute. The only valid attribute type is string.
layerOrder	This declares the layer order of the feature type. Valid values are all integers. Feature types with lower layer orders will be drawn first. Therefore, features in feature types with higher layer orders will appear on top of features in feature types with lower layer orders. If a value is not specified, then the feature type will have an effective layer order value of '0'. If two features have identical layer order values, then the two will be ordered arbitrarily.
pageCoordinates	The value specifies whether the coordinates of geometries will be interpreted in page coordinates. If this attribute is set to YES, then the coordinates of the geometry are treated as page coordinate values, and the feature can be drawn anywhere on the page. The default value is NO.
opacity	This determines the opacity level of features of this feature type when their pdf_opacity feature attribute is unset. If this parameter is set, the value overrides the writer parameter DEFAULT_OPACITY. A value of 1.0 is fully opaque, and 0.0 is completely transparent.
visibility	If the value is VISIBLE, then the layer will be visible by default after opening the output file in Adobe Acrobat Reader. If set to HIDDEN, then the layer will not be initially visible. The visibility of layers can be toggled in Adobe Acrobat Reader after opening the file.

PAGE_SIZE

This directive specifies the size of the output page of the PDF document. The default page size is Letter.

Preset page sizes for common paper sizes can be selected, or the page size can be specified in typographical points in the format <width> <height>.

Required/Optional

Optional

Values

A3 | A4 | A5 | B5 | Ledger | Legal | Legal-half | Letter (Default) | Letter-half | <0 ...> <0 ...>

* Workbench Parameter

Page size

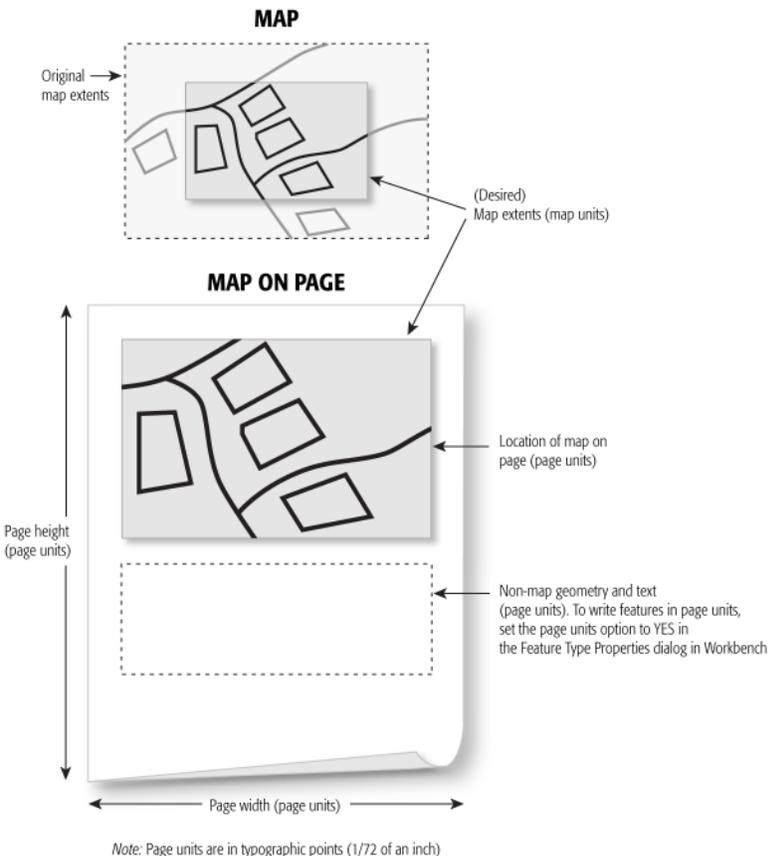
PAGE_VIEWPORT (Location of Map on Page)

This directive determines where to place the map on the page, and how large the map should be on the page.

The format for this directive is four integers separated by spaces describing the lower left corner and the upper right corner of the viewport/rectangle, specified in typographical points. The lower left corner of the page contains coordinate (0,0) and the top right corner contains coordinate (<width>,<height>), where these two values are the page size specified by **PAGE_SIZE**.

If the aspect ratios of the page viewport and the world viewport (**WORLD_VIEWPORT**) differ, then the lesser scaling factor will be chosen: data inside the world viewport will not be clipped and data outside the world viewport might become visible.

If a value for the directive is not specified, then the page viewport rectangle will be a centered rectangle with a width and length that is 90% of the page width and length. The page viewport coordinates must be between (0,0) and (page width,page height).



Required/Optional

Optional

Values

<minimum x> <minimum y> <maximum x> <maximum y>

The values can also be specified as a percentage of the page width and page height. The values must be an integer ending with a percentage sign. The values can also be negative values, and they are interpreted as being relative to the top and right edges instead of the left and bottom edges. For example, for a page size of 1000 by 1000 points, the rectangle "50 50 -50 -50" is identical to the rectangle "50 50 950 950" for this page size.

*** Workbench Parameter**

Page viewport dimensions

WORLD_VIEWPORT (Map Extents)

This directive specifies the extents of the map to write within the page viewport, by defining the lower left and upper right corners of the page viewport in map units.

Geometry outside these extents will be clipped when drawn on the page. The format for the directive is four floating point numbers separated by spaces describing the lower left corner and the upper right corner of the rectangle.

If a value for the directive is not specified, then the world viewport rectangle will be the bounding box of the entire dataset.

Required/Optional

Optional

Values

<minimum x> <minimum y> <maximum x> <maximum y>

*** Workbench Parameter**

World viewport dimensions

DEFAULT_OPACITY

This directive specifies the opacity value of the fill color of area geometries. The boundaries of area geometries are not affected by this setting.

Required/Optional

Optional

Values

<0.0...1.0>

A value of 0 corresponds to complete transparency and a value of 1 is complete opaqueness.

Default Value: 0.4

*** Workbench Parameter**

Default fill opacity value

DEFAULT_POINT_SIZE

This directive specifies the default radius in typographical points for point geometry.

Required/Optional

Optional

Values

<0.0...>

Default Value: 1.0

*** Workbench Parameter**

Default point size value

DEFAULT_LINE_WIDTH

This directive specifies the default width in typographical points for line geometry and boundaries of area geometry.

Required/Optional

Optional

Values

<0.0...>

Default Value: 1.0

*** Workbench Parameter**

Default line width value

PANEL_VISIBILITY

This directive determines the panel that is visible immediately after opening the output PDF file in Adobe Acrobat software.

Required/Optional

Optional

Values

None (default): No panel will be initially displayed

Layers: Layer panel will be visible after opening the file

Pages: Page Thumbnails panel will be visible

*** Workbench Parameter**

Navigation Panel to Display

RANDOMIZE_FEATURE_TYPE_COLOR

This directive specifies whether features without the fme_color attribute set will be assigned a random color based on its feature type.

Required/Optional

Optional

Values

YES (default)

NO (features without the fme_color attribute set will be assigned the color black)

* **Workbench Parameter**

Randomize Feature Type Color

RICH_TEXT

Required/Optional: *Optional*

This directive specifies whether the text string of text features is in the rich text format. If this directive is set to NO, then the text string is written as-is to the page. If this directive is set to YES, then the text string will be processed for style directives. For more details, see the "Text" section under Feature Representation.

Values: *YES|NO*

Default Value: *NO*

Workbench Parameter: *Text in rich text format*

FONT_DIRECTORIES

Required/Optional: *Optional*

This directive specifies the directories that the writer will search in to find the TrueType fonts used in the workspace. The workspace directory of the translation is always searched.

Values: *<multiple directories>*

Default Value:

Workbench Parameter: *TrueType font directories*

WRITE_ATTRIBUTES

Required/Optional: *Optional*

This directive specifies whether attribution data will be written. Not writing attribution data will decrease the file size of the output file and may improve viewing performance.

Values: *YES|NO*

Default Value: *YES*

Workbench Parameter: *Write attributes*

COMPRESS_STREAMS

Required/Optional: *Optional*

This directive specifies whether streams in PDF files will be compressed.

Values: *YES|NO*

Default Value: *YES*

Workbench Parameter: *Compress streams*

PDF14_COMPATIBLE

Required/Optional: *Optional*

This directive specifies whether the output file will be PDF1.4 compatible. If the directive is set to NO, then the output file can only be opened by applications that are compatible with PDF1.5 and above.

Values: *YES|NO*

Default Value: *NO*

Workbench Parameter: *PDF 1.4 compatible*

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Attribute Name	Contents
pdf_type	<p>The value specifies PDF geometric type of this entity.</p> <p>Range: pdf_area pdf_collection pdf_line pdf_point pdf_text</p> <p>Default: No default</p>
pdf_name	<p>If attribution data is written, then the value of this attribute determines the name of the structure element associated with the feature. If this attribute is not set, then the structure element will be numbered sequentially. The names need not be unique.</p> <p>Range: <string></p> <p>Default: <auto-generated integer></p>
pdf_line_width	<p>The value specifies the line width in typographical points of line geometries and boundaries of area geometries. Points within half the line width perpendicular distance from the line path will be painted.</p> <p>Range: <float></p> <p>Default: 1.0</p>
pdf_line_cap_style	<p>The value specifies the cap style for the ends of lines.</p> <p>Range: <0,1,2></p> <p>0 - Butt cap: Lines are squared off at the end and do not project past the end of the line.</p> <p>1 - Round cap: Semicircles with diameter equal to the line width cap the ends of lines.</p> <p>2 - Projecting square cap: Lines project past the end by a</p>

Attribute Name	Contents
	<p>distance equal to half the line width and are squared off.</p> <p>Default: 0</p>
pdf_line_join_style	<p>The value specifies the shape of corners between segments of paths.</p> <p>Range: <0,1,2></p> <p>0 - Miter join: Outer edges of segments are extended until they meet.</p> <p>1 - Round join: Arcs with diameter equal to the line width are drawn around corners.</p> <p>2 - Bevel join: Two adjacent segments are finished with butt caps, and the notch beyond the ends is filled with a triangle.</p> <p>Default: 0</p>
pdf_line_miter_limit	<p>For miter joins, the miter limit imposes a maximum on the ratio of the miter length to the line width. For example, a miter limit of 1.414 will bevel the ends of two segments meeting at an angle less than 90 degrees (the far corner will be at a distance $\sqrt{1^2+1^2}=\sqrt{2}$ from the line).</p> <p>Range: <0.0...></p> <p>Default: 0.0</p>
pdf_line_dash_pattern{}	<p>The values in this list attribute specify the dash pattern for line geometries and the boundaries of area geometries. This attribute works together with <code>pdf_line_dash_pattern_phase</code> to establish a simple dashed line style. Elements of the list specify the alternating lengths of dashes and gaps. The pattern starts with a dash.</p> <p>Range: For each element in the list: <1,2,...></p> <p>Default: Empty list</p>
pdf_line_dash_pattern_phase	<p>The value specifies the starting phase of the dash pattern. This attribute works together with <code>pdf_line_dash_pattern</code> to establish a simple dashed line style. The following is an example dash pattern specification:</p>

Attribute Name	Contents
	<p>pdf_line_dash_pattern{0} = 2 pdf_line_dash_pattern{1} = 3 pdf_line_dash_pattern_phase = 1</p> <p>A dash of length 1 will be drawn, then gaps of length 3 and dashes of length 2 will cyclically follow thereafter. Range: <0,1,2,...></p> <p>Default: 0</p>
pdf_url	<p>If this attribute is set, then the feature will become an interactive annotation. When a user clicks on the feature in a PDF viewer application that supports URI actions, the value will be treated as a URI and it will be resolved. In the common case that the value is a URL, Adobe Acrobat Reader will open a web browser to resolve the address specified.</p> <p>Note: See the “Annotations” section under Feature Representation for behavioral notes.</p>
pdf_tooltip	<p>If this attribute is set, then the feature will become an interactive annotation. The value specifies the tooltip string that will be displayed when an user hovers over the feature with the mouse cursor in the PDF viewer application.</p> <p>Note: See the “Annotations” section under Feature Representation for behavioral notes.</p>
pdf_fill_opacity	<p>The value specifies the opacity of the fill color of the feature. A value of 1.0 is fully opaque, and 0.0 is completely transparent. If this value is not set, then the opacity of the feature is determined by the pdf_default_opacity feature type parameter. If the feature type parameter is not set either, then the writer directive DEFAULT_OPACITY determines the opacity.</p>
pdf_pen_opacity	<p>The value specifies the opacity of the stroking color of the feature. A value of 1.0 is fully opaque, and 0.0 is completely transparent. If this value is not set, then the stroking opacity is set to fully opaque.</p>

Annotations

pdf_type: any

Features with the pdf_url or the pdf_tooltip attribute set become annotation objects. There are several behavioral differences between annotation objects and non-annotation objects:

- Annotation objects will always appear above non-annotation objects, regardless of layer ordering.
- The interactive area of an annotation object is the rectangular bound of the feature instead of its precise outline.
- Annotation objects are no longer selectable through the Object Data tool or the Model Tree interface.
- Even when the annotation object's layer is hidden, the annotation will still provide tooltips and be interactive. The annotation object's parent layer does not affect the visibility of the annotation; only the object's layer itself will affect its visibility.

Points

pdf_type: pdf_point

A PDF point feature is drawn as a point with a radius of 1 typographical point.

The following attribute is applicable to point features:

pdf_point_width	The value specifies the point width in typographical points of point geometries. Range: <float> Default: 1.0
-----------------	-----------------------------------------------------------------------------------------------------------------

Lines

pdf_type: pdf_line

A PDF line feature is drawn as a stroked line.

Area

pdf_type: pdf_area

A PDF area feature is written as a filled area with a stroked boundary. The fill opacity is controlled by the **DEFAULT_OPACITY** directive.

Collection

pdf_type: pdf_collection

Each component of a PDF collection feature is drawn according to their geometry type.

Text

pdf_type: pdf_text

A PDF text feature is drawn as a text annotation according to its fme_text_string, fme_text_size, and fme_rotation attributes.

The encoding of the text string is determined as follows: if the font is one of the PDF Core 14 fonts, then the string is decoded using Windows ANSI code page 1252. If the font is a TrueType font, then the string is decoded using the Macintosh Roman code page. If the TrueType font has a Microsoft Symbol character map table, then the font is treated as a symbolic font, and the text string can specify characters in the FF00-FFFF range of the character map by encoding only the low-byte of the code point. Desired characters in symbolic fonts can either be specified using XML numeric character references (NCR) in rich text format (see below) or if the code point coincides with ASCII characters, the ASCII characters themselves.

The following attributes are applicable to text features:

Attribute Name	Contents
----------------	----------

pdf_text_font	The value specifies the default font family of the text representation. If left blank, Helvetica will be used. Default: Helvetica
pdf_text_underline	If the value is 'Y', the text will be underlined. Default: N
pdf_text_strikethrough	If the value is 'Y', the text will have a strikethrough. Default: N
pdf_text_bold	If the value is 'Y', the text will have a bold style. Default: N
pdf_text_italic	If the value is 'Y', the text will have an italic or oblique style. Default: N

The text string can be specified in a rich text format. The format is a subset of XHTML. For more information on XHTML, visit <http://www.w3.org/TR/xhtml1/>. The following are the supported XML elements:

- `<body>...</body>`, `...`, `<p>...</p>` - Can be used to specify a style for its enclosed text through its "style" attribute.
- `...` - Bolds the enclosed text.
- `<i>...</i>` - Italicizes the enclosed text.
- `<u>...</u>` - Underlines the enclosed text.
- `...` - Adds a strikethrough to the enclosed text.
- `
` - Adds a line break.

The "style" XML attribute has the following format:

`"property:value;...;property:value"`

The following properties are supported:

- font-family - Specifies the font family of the text.
- font-size - Specifies the point size of the font.
- color - Specifies the color of the text. The color can be specified through the format "#RRGGBB" where each color component is specified as a hexadecimal value, or through the 16 HTML color names (<http://www.w3.org/TR/REC-html40/types.html#h-6.5>).
- text-decoration - Valid values are "underline" and "line-through".

The following is an example rich text fme_text_string value:

`<body>Hello
World!</body>`

In the PDF document, the text "Hello" will use the styling specified through the format attributes. The text "World!" appears on the next line and will have a font size of 30 but will inherit all other style attributes.

Adobe Illustrator (IEPS) Writer

The Adobe Illustrator Encapsulated PostScript® (IEPS) Writer module enables FME to write Encapsulated PostScript export files specifically formatted to work with Adobe Illustrator. Illustrator IEPS is a different flavour of EPS and makes use of some of the functionality of Adobe Illustrator. The most significant additions are the use of layers and object attributes. In this format, many of the PostScript keywords have been shortened into special Adobe Illustrator single letter functions. The implication is that EPS files produced by this writer cannot be used outside of Adobe Illustrator. The standard EPS writer should be used if the EPS is to be used in other applications.

IEPS is most often used for high-quality plots in desktop publishing software.

Note: This writer may write files that are quite large since it does create an output coordinate for every source coordinate. If you find your .eps files getting too large, it is recommended that you first generalize your source data to make it less dense using the FME's @Generalize function (or the FME Workbench Generalizer transformer).

IEPS Quick Facts

Format Type Identifier	IEPS
Reader/Writer	Writer
Licensing Level	Base
Dependencies	None
Dataset Type	File
Feature Type	Layer name
Typical File Extensions	.eps
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	Yes
Spatial Index	Not applicable
Schema Required	Yes
Transaction Support	No
Geometry Type Attribute	ieps_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
ellipses	no		text	no
line	yes		z values	no
none	no			no

Overview

IEPS is a two-dimensional (2D) format with the ability to store user-defined attributes for the geometric data.

All IEPS information is contained in a single page beginning with a version header as well as a bounding box definition. IEPS is based upon the PostScript format which provides methods for graphical drawing, simple programming control structures and the ability to create user-defined variables and functions.

All IEPS data is contained in a single file with an **.ieps** extension.

File Name Extension	Contents
.ieps	All vector geometric data.

The IEPS writer supports export of points, lines, polygons, and text geometric data.

Some geometric entities may have display properties such as pen width, line type, and color. Color may be specified in red/green/blue (RGB) as well as cyan/magenta/yellow/black (CMYK).

Writer Overview

The IEPS writer creates and writes feature data to an IEPS file specified by the **DATASET** keyword. The writer searches the mapping file for the **<writerKeyword>_DATASET** keyword in the mapping file. This keyword is required to be in the mapping file. An old IEPS file in the directory with the same file name is overwritten with the new feature data. A typical mapping file fragment specifying the output IEPS file looks like:

```
IEPS_DATASET /usr/data/ieps/myfile.ieps
```

Writer Directives

The directives processed by the IEPS writer are listed below. The suffixes shown are prefixed by the current **<writerKeyword>** in a mapping file. By default, the **<writerKeyword>** for the IEPS writer is **IEPS**.

DATASET

Required/Optional: *Required*

The IEPS writer processes the **DATASET** keyword as described in *Writer Overview*. Additional keywords can be used to set default parameters that are applied to all applicable features in the file. However, the values set by the keywords can be overwritten if the feature itself has a value defined for that parameter. For example, although the **LINE_WIDTH** keyword may be used to specify a default width of 5 for all lines in the file, if an **ieps_polyline** feature has its **ieps_line_width** set to a value of 2, then the line width of 2 will be used over the default value of 5.

Workbench Parameter: *Destination Adobe Illustrator EPS File*

DEF

Required/Optional: *Required*

This is a required keyword that defines the layers within the file. DEF lines also list the attributes that will be saved as object tags on features of that layer, and may also include the attribute **IEPS_LAYER_COLOR**. This should be fol-

lowed by an RGB combination ranging in intensities from 0 to 255, separated by commas. This defines the layer color seen in Adobe Illustrator.

Attribute	Contents	Required/Optional
IEPS_LAYER_COLOR	<p>This is an attribute that can be used on a DEF line. It defines the layer color seen in Illustrator.</p> <p>Range: 0..255, 0..255, 0..255</p> <p>Default: No Default</p>	Optional

RESOLUTION _X and RESOLUTION _Y

Required/Optional: *Optional*

These directives define the bounding box of the IEPS output file. The bounding box extends from the lower left corner of the page (defined as 0,0) and extends out to the values entered. By default, the X value is set to 612 and the Y value is set to 792. These values map onto an 8.5 x 11-inch piece of paper.

Range: *Integer > 0*

Default:

RESOLUTION_X: 612

RESOLUTION_Y: 792

Workbench Parameter: *Width (points), Height (points)*

MAINTAIN_ASPECT

Required/Optional: *Optional*

This directive is followed by a value of **YES** or **NO**. By default, the value is set to **YES**. A **YES** indicates that the original map aspect will be maintained to fit within the destination-defined bounding box. This means that the entire destination bounding box defined may not be used. Alternatively, the value **NO** causes the original map to be stretched onto the defined destination bounding box.

Range: *YES | NO*

Default: *YES*

Workbench Parameter: *Maintain Map Aspect Ratio*

LINE_WIDTH

Required/Optional: *Optional*

This directive is followed by the value in pixels of the line width you wish to use by default. The default value is set to 0, which is the thinnest printable line width.

Range: *float >= 0*

Default: *0.0 (1 pixel wide: the thinnest line that can be rendered at device resolution)*

Workbench Parameter: *Line Width (pixels)*

TEXT_WIDTH

Required/Optional: *Optional*

This directive has an attribute just like **LINE_WIDTH** except that this width is applied to text features. The default value is set to 0, which is the thinnest printable line width.

Range: *float >= 0*

Default: *0.0 (1 pixel wide: the thinnest line that can be rendered at device resolution)*

Workbench Parameter: *Text Width (pixels)*

TEXT_FONT

Required/Optional: *Optional*

This directive specifies the default font applied to all text features. The font must be a PostScript name. The fonts supported depend on the destination of the IEPS file. Some typical fonts are NewBaskerville, Times, Helvetica and Courier. The default is NewBaskerville since it is the most commonly installed with Adobe Illustrator.

Range: *String*

Default: *NewBaskerVille*

Workbench Parameter: *Text Font*

TEXT_STYLE

Required/Optional: *Optional*

This directive specifies the default style to be applied to the text font all text features. This attribute must be matched to the current font since it is the combination of text font and text style that is recognized by Adobe Illustrator. Some typical font and style combinations are NewBaskerville-(None, Bold), Times-(None, Roman, Italic, Bold, BoldItalic), Helvetica-(None, Oblique, Bold, BoldOblique), and Courier-(None, Oblique, Bold, BoldItalic). Note that the keyword **NONE** can be used to specify that no style should be applied to the font.

Range: *String*

Default: *Bold*

Workbench Parameter: *Text Style*

LINE_JOIN_TYPE

Required/Optional: *Optional*

This directive is followed by the values **0**, **1**, or **2**. These values specify the default shape to be put at corners of paths painted: **0** specifies a sharp corner, **1** specifies a rounded corner, and **2** specifies a butt-end corner.

Range: *0, 1, 2*

Default: *0*

Workbench Parameter: *Line Join Type*

LINE_CAP_TYPE

Required/Optional: *Optional*

This directive is followed by the values **0**, **1**, or **2**. These values specify the default cap that will be used on line segments. **0** specifies butt-end caps, **1** specifies rounded-end caps and **2** specifies square-end caps.

Range: *0, 1, 2*

Default: *0*

Workbench Parameter: *Line Cap Type*

FORCE_CMYK

Required/Optional: *Optional*

By setting the value following this keyword to **YES**, then all color usage output to the IEPS file is in CMYK. By default, this value is **NO**, meaning that a mix of RGB and CMYK color schemes may be in the output IEPS file. However, despite forcing CMYK color output, some IEPS viewers may not support the **setcmykcolor** call in their library. In these cases, the actual output of colors is done using a function we define in PostScript which interfaces exactly like the **setcmykcolor** call, but uses **setrgbcolor** underneath. This will depend on the IEPS viewer you are using.

Range: *YES | NO*

Default: NO

Workbench Parameter: *Force CMYK*

LOCK_FEATURES

Required/Optional: *Optional*

If set to YES, by default all features will be locked and cannot be selected or edited in Adobe Illustrator. Note: Even if **LOCK_FEATURES** is set to YES, individual features can be unlocked if its **eps_lock_feature** is set to 0 (meaning NOT locked). Hence, an individual **eps_lock_feature** value overrides this **LOCK_FEATURES** default value.

Range: YES | NO

Default: NO

Workbench Parameter: *Lock Features*

RENDER_TYPE

Required/Optional: *Optional*

This directive determines how the text is output. This value will be used as the default render type for all text in the file but it will be overridden if the text feature has its own user-defined render type value.

This directive is followed by the values 0, 1, or 2. These values specify the default rendering that will be applied to text features: 0 = fill, 1 = stroke, 2 = stroke and fill. The default value is 2.

Range: 0, 1, 2

Default: 2

Workbench Parameter: *Render Type*

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

IEPS features consist of geometry but no user-defined attributes, although there are special attributes to hold the type of the geometric entity and its display parameters. The feature type of a feature written to IEPS is used to specify its layer in the output Adobe Illustrator file.

All IEPS features contain a **ieps_type** attribute, which identifies the geometric type. Each element type also has a color associated with it. Depending on the geometric type, the feature contains additional attributes specific to the geometric type. These are described in subsequent sections.

Attribute Name	Contents
ieps_type	The IEPS geometric type of this entity. Range: ieps_polyline ieps_area ieps_text ieps_point Default: No default
ieps_cmyk_color	This is a string that represents the color intensities of the element. It is formatted as cyan (C), magenta (M), yellow (Y) and black (K), This color attribute has highest priority. If present, it will be used in preference over ieps_color and fme_

Attribute Name	Contents
	<p><code>color</code> attributes. Range: String. (0..1, 0..1, 0..1, 0..1) Default: String (0,0,0, 1)</p>
ieps_cmyk_fill_color	<p>This is a string that represents the fill color intensities of the element. It is formatted as cyan (C), magenta (M), yellow (Y) and black (K), This color attribute has highest priority. If present, it will be used in preference over <code>ieps_fill_color</code> and <code>fme_fill_color</code> attributes. Range: String. (0..1, 0..1, 0..1, 0..1) Default: String (0,0,0,1)</p>
ieps_color	<p>This is a string that represents the color intensities of the element. It is formatted as red, green, blue intensities which range between 0..1 Note that if this attribute is not found, then <code>fme_color</code> will be used. Range: String. (0..1, 0..1, 0..1) Default: String (0,0,0)</p>
ieps_fill_color	<p>This is a string that represents the color intensities of the element. It is formatted as red, green, blue intensities which range between 0..1. If this attribute is not found, then the writer will refer to <code>fme_fill_color</code>. Range: String. (0..1, 0..1, 0..1) Default: None</p>
ieps_url	<p>Allows you to attach a URL to a feature. The URL should be formatted as <i>http://www.safe.com</i>. Range: String Default: No Default</p>
ieps_dash_on	<p>The number of pixels to be used as the <code>on</code> part of the dashed line used to draw the feature. If <code>ieps_pen_linewidth</code> is specified, then this value is multiplied by the size of the pen to determine the number of pixels. If both <code>ieps_dash_on</code> and <code>ieps_dash_off</code> are 0, then a solid line is used. Range: Integer > 0 Default: 0</p>
ieps_dash_off	<p>The number of pixels to be used as the <code>off</code> part of the dashed line used to draw the feature. If <code>ieps_pen_linewidth</code> is specified, then this value is multiplied by the size of the pen to determine the</p>

Attribute Name	Contents
	number of pixels. If both <code>ieps_dash_on</code> and <code>ieps_dash_off</code> are 0, then a solid line is used. Range: Integer > 0 Default: 0
<code>ieps_line_join_type</code>	Specify the type of corner that should be drawn onto this path. 0 = sharp corners, 1 = rounded corners, 2 = butt-end corners Range: 0, 1, 2 Default: 0 Optional: Yes
<code>ieps_line_cap_type</code>	Specify the type of caps on line ends. 0 = butt end caps, 1 = rounded end caps, 2 = square end caps Range: 0, 1, 2 Default: 0 Optional: Yes
<code>ieps_locked_flag</code>	This determines whether or not the feature can be selected for editing when the document is opened in Adobe Illustrator. If set to 0, the feature can be selected for editing. If set to 1, the feature is locked and cannot be selected. Range: 0, 1 Default: 0 Optional: Yes

Areas

ieps_type: `ieps_area`

IEPS polygon features specify area (polygonal) features. The areas that make up a single feature may or may not be disjoint, and may contain polygons that have holes. Each area has a pen style associated with it to control the color, line weight, line type, and brush pattern used when it's drawn. If the area contains holes then when the fill pattern is applied, the holes enclosed by the area will **not** be filled. If no pen style is defined for a polygon entity, the previous style is used.

The following table lists the special FME attribute names used to control the IEPS polygon settings.

Attribute Name	Contents
<code>ieps_line_width</code>	Defines the line width used to draw the polyline. By default, the line is drawn one pixel wide. Range: Float >= 0 Default: 0.0 (the thinnest line that can be rendered at device resolution, i.e. 1 pixel wide)

Polylines

ieps_type: `ieps_polyline`

IEPS polyline features specify linear features defined by a sequence of x and y coordinates. Polyline encapsulate the concept of a line since a line is just a sequence of two points. Each polyline has a pen style associated with it that specifies the color, line weight, and line type used when the line is drawn. If no pen type is defined for a polyline entity, if line attributes aren't found, then default parameters are used.

The table below lists the special FME attribute names used to control the IEPS polyline settings.

Attribute Name	Contents
ieps_line_width	<p>Defines the line width used to draw the polyline. By default, the line is drawn one pixel wide.</p> <p>Range: Float ≥ 0</p> <p>Default: 0.0 (the thinnest line that can be rendered at device resolution, i.e. 1 pixel wide)</p>

Text

ieps_type: ieps_text

IEPS text is used for text annotation in IEPS. The coordinates specify the lower left coordinates of the text when it is placed. In addition, the size and angle in which the text is output can be specified.

The table below lists the special FME attribute names used to control the IEPS text:

Attribute Name	Contents
ieps_size	<p>The size of the text specified in ground units</p> <p>Range: float > 0</p> <p>Default: 0</p>
ieps_illustrator_size	<p>The size of the point text specified in points. If this is set, it will override the ieps_size value.</p> <p>Range: float > 0</p> <p>Default: 12pt</p>
ieps_rotation	<p>The text rotation is given in degrees and measured counterclockwise up from the horizontal.</p> <p>Range: -360..360</p> <p>Default: 0</p>
ieps_font	<p>The PostScript name of the font. The fonts supported depend on the destination of the IEPS file. Some typical fonts are Times, Helvetica and Courier.</p> <p>Range: String</p> <p>Default: NewBaskerville</p>
ieps_style	<p>The style of the font. This attribute must be matched with the current font since it's the combination of font and style that IEPS recognizes. Some typical fonts and styles are Times-(None, Roman, Italic, Bold, BoldItalic), Helvetica-(None, Oblique, Bold, BoldOblique) and Courier-(None, Oblique, Bold, BoldOblique). Note the keyword</p>

Attribute Name	Contents
	'NONE' can be specified to indicate no style on the font. Range: String Default: Bold
ieps_text_string	The text to be displayed. Range: String Default: No default
ieps_text_width	Defines the line width used to stroke the text. By default, the stroked line is drawn one pixel wide. Range: Float ≥ 0 Default: 0.0 (the thinnest line that can be rendered at device resolution, i.e. 1 pixel wide)
ieps_render_type	This determines how the text is output. 0 = filled, 1 = stroked, 2 = stroked and filled Range: 0,1,2 Default: 2

Point

ieps_type: ieps_point

IEPS point is used for point annotation in IEPS. Points will be represented as text. By default, a symbol will be represented by a period.

Attribute Name	Contents
ieps_size	The size of the point text specified in ground units Range: float > 0 Default: 0
ieps_illustrator_size	The size of the point text specified in points. If this is set, it will override the <i>ieps_size</i> value. Range: float > 0 Default: 12pt
ieps_rotation	The text rotation is given in degrees and measured counterclockwise up from the horizontal. Range: -360..360 Default: 0
ieps_font	The PostScript name of the font. The fonts supported depend on the destination of the IEPS file. Some typical fonts are Times, Helvetica and Courier. Range: String Default: NewBaskerville

Attribute Name	Contents
ieps_style	<p>The style of the font. This attribute must be matched with the current font since it's the combination of font and style that IEPS recognizes. Some typical fonts and styles are Times-(None, Roman, Italic, Bold, BoldItalic), Helvetica-(None,Oblique, Bold, BoldOblique) and Courier-(None,Oblique, Bold, BoldOblique). Note the keyword 'NONE' can be specified to indicate no style on the font.</p> <p>Range: String Default: Bold</p>
ieps_symbol_string	<p>The text to be displayed.</p> <p>Range: String Default: "."</p>
ieps_symbol_width	<p>Defines the line width used to stroke the text. By default, the stroked line is drawn one pixel wide.</p> <p>Range: Float ≥ 0 Default: 0.0 (the thinnest line that can be rendered at device resolution, i.e. 1 pixel wide)</p>
ieps_render_type	<p>This determines how the text is output.</p> <p>0 = filled, 1 = stroked, 2 = stroked and filled</p> <p>Range: 0,1,2 Default: 2</p>

Aeronautical Information Exchange Model (AIXM) Reader/Writer

The AIXM Reader/Writer enables FME to read Aeronautical Information Exchange Model format files.

This chapter assumes familiarity with the AIXM format.

Overview

The Aeronautical Information Exchange Model (AIXM) format was developed by EUROCONTROL, the European Organisation for the Safety of Air Navigation, to allow aeronautical data standardization and exchange. The role of AIXM is to enable systems to exchange aeronautical information in the form of XML-encoded data.

AIXM Quick Facts

Format Type Identifier	AIXM
Reader/Writer	Both
Dataset Type	File
Licensing Level	Professional
Dependencies	None
Feature Type	AIXM entity name
Typical File Extensions	.xml
Automated Translation Support	No
User-Defined Attributes	No
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Never
Schema Required	No
Transaction Support	No
Geometry Type Attribute	xml_type

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	yes	point	yes
circles	yes	polygon	yes
circular arc	yes	raster	no
donut polygon	no	solid	no
elliptical arc	no	surface	no
ellipses	no	text	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
line	yes		z values	no
none	yes			

Reader Overview

The AIXM reader presents features by normalizing the XML data into the entities of the AIXM Entity-Relational model. Thus, the feature representation is not equivalent to the AIXM XML format representation of the AIXM E-R model entity.

Reader Directives

The suffixes shown below are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the AIXM reader is `AIXM`.

DATASET

Required/Optional: *Required*

The value for this directive is the path of the AIXM file to be read. A typical mapping file fragment specifying an input AIXM dataset looks like:

```
AIXM_DATASET /usr/data/aixm.xml
```

Workbench Parameter: *Source Aeronautical Information Exchange Model (AIXM) File(s)*

INTERPOLATE

Required/Optional: *Optional*

The value for this directive determines whether non-linear interpolation will be performed between two vertices of an area or line geometry. This keyword will also determine the representation of geometry data. Further information on this topic can be found under the Feature Representation heading. An example mapping file fragment specifying that interpolation should be performed looks like:

```
INTERPOLATE Yes
```

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

Writer Overview

The AIXM writer has a fixed output schema that closely resembles the AIXM Entity-Relational model.

The reader can be connected directly to the writer and the output file will be nearly identical to the original source file.

Writer Directives

The following table lists the directives processed by the AIXM writer. The suffixes shown will be prefixed by the current <writerKeyword> in a mapping file. By default, the <writerKeyword> for the AIXM writer is **AIXM**.

DATASET

Required/Optional: *Required*

The value for this keyword is the path of the output AIXM file. A typical mapping file fragment specifying an output AIXM file looks like:

```
AIXM_DATASET /usr/data/aixm.xml
```

Workbench Parameter: *Destination Aeronautical Information Exchange Model (AIXM) File*

WRITE_MODE

Required/Optional: *Optional*

The value for this keyword determines the type of AIXM file, either an AIXM Snapshot or AIXM Update, produced by the writer. Valid values are UPDATE and SNAPSHOT. The default value is UPDATE:

AIXM_WRITE_MODE UPDATE

Workbench Parameter: *AIXM writer mode*

ORIGIN

Required/Optional: *Optional*

The value for this keyword is a string that determines the originator of the AIXM message:

AIXM_ORIGIN ABC

Workbench Parameter: *Origin*

CREATED

Required/Optional: *Optional*

The value for this keyword determines the date and time that the AIXM message was created. The string should be a valid XML dateTime string:

AIXM_CREATED 2002-10-10

Workbench Parameter: *Created*

EFFECTIVE

Required/Optional: *Optional*

The value for this keyword determines the date and time that the AIXM message becomes effective. The string should be a valid XML dateTime string:

AIXM_CREATED 2002-10-10

Workbench Parameter: *Effective*

USE_CHG

Required/Optional: *Optional*

The value for this keyword determines whether the 'chg' XML attributes will be added to each XML element written by the writer. Valid values are YES and NO. If the value is YES, then XML elements whose names appear in the aixm_update_changed format specific attribute will have an XML attribute named 'chg' with a value of '1' inserted.

The default value for this keyword is YES:

AIXM_USE_CHG YES

Workbench Parameter: *Add 'chg' attributes*

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), special FME feature attributes are used by the AIXM reader to store the characteristics of the features it reads.

The AIXM Reader module utilizes the XML Reader module in processing the AIXM XML file. Thus, the feature representation is similar to the feature representation of the XML Reader module. The format attribute, **xml_type**,

which may identify the geometry type of the feature, is identical in intent to the same attribute set by the XML Reader. Details of this attribute can be found in the *XML Reader/Writer* documentation.

Attribute Name	Contents
aixm_update_ID	When an AIXM Update message changes the natural key that identifies an object, then this attribute will hold the old natural key of the object to be updated.
aixm_update_group_no	This attribute determines the order of the AIXM Group elements within the output file. The values of this attribute are integers, and identifies the feature with a specific group. When all features are received by the AIXM writer, features are grouped according to the values of their aixm_update_group_no attribute, and the groups are written in ascending order of through group numbers. If this attribute is not specified, then the feature will be grouped with group number zero.
aixm_update_name	The value of this attribute specifies the 'name' attribute of the AIXM Group element that holds the feature.
aixm_update_subname	The value of this attribute specifies the 'sub-name' attribute of the AIXM Group element that holds the feature.
aixm_update_reason	The value of this attribute specifies the 'reason' attribute of the AIXM Group element that holds the feature.
aixm_update_type	The value of this attribute determines the type of the AIXM Update message for that particular feature: New, Update, or Withdrawn.
aixm_noseq	The AIXM reader normalizes the XML schema. After this transformation, child elements that composed a parent element may become independent features. If the child elements were ordered within the parent element, then this attribute will hold the sequence number that determines the child element's placement within a parent element.
aixm_update_changed	This is a list attribute that holds the names of attributes that are flagged as changed in an AIXM Update message.

ASPRS LIDAR Data Exchange Format (LAS) Reader/Writer

Note: This format is not available in FME Base Edition.

The American Society Photogrammetry and Remote Sensing (ASPRS) LIDAR (LAS) Reader allows FME to read LIDAR (data exchange format standard) LAS specifications.

Overview

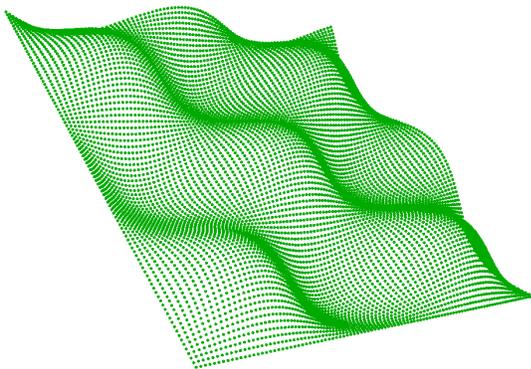
The LAS file is intended to contain LIDAR point records. The data will generally be put into this format from software (provided by LIDAR hardware vendors) which combines GPS, IMU, and laser pulse range data to produce X, Y, and Z point data. The intention of the data format is to provide an open format that allows different LIDAR hardware and software tools to output data in a common format. FME supports LAS versions 1.0, 1.1, and 1.2.

The format contains binary data consisting of a header block, Variable Length Records (VLRs), and point data.

Note: Reading and writing of arbitrary VLRs is not currently supported; only defined georeferencing information VLRs are supported.

About Point Clouds

A point cloud is a type of geometry that is useful for storing large amounts of data, typically gathered from LIDAR applications. The use of LIDAR allows for fast and accurate collection of data, such as for forestry canopy measurements, or landscape modeling. Point cloud geometry allows for quick and efficient processing of a large collection of vertices in 3D space that represent the external surfaces of objects. Together, these vertices form a model which can be transformed, and visualized. Some operations of the point cloud geometry involve thinning, splitting, and combining to produce a more useable set of vertices.



Associated with each vertex are a number of properties called components, which contains a value describing the point. These component values can be used to classify different sections of the collection of points contained in the point cloud geometry. The specific set of components stored by the point cloud is referred to as the interpretation.

Interpretation	Allowed Values	Description
Intensity	1.7E +/- 308 (15 digits)	The magnitude of the intensity of the pulse return.
Color	0 to 65,535	The color of the object at the point, in RGB color.
Classification	0 to 65,535	The classification value categorizes the points into fields, such as ground, building, water, etc.
Returns	1 - 5	The return value is the return number from a pulse.
Number of returns	1 - 5	The total number of detected returns from a single pulse.
Angle	-90 to 90	The angle of the pulse that the point was scanned at.
Flight line	0 to 4,294,967,295	The flight line number the point was detected in.

Scan Direction	0 and 1	The direction in which a scanning mirror was directed when the point was detected.
Point ID	1 to 65,535	This point ID is indicative of the point origin.
POSIX time	1.7E +/- 308 (15 digits)	Used to express the time, as the number of seconds elapsed since UTC January 1 st , 1970.
User data	0 to 65,535	The user data value is for the user to use.
GPS time and GPS week	GPS Week: 1.7E +/- 308 (15 digits) GPS Time: 0 to 65,535	Together, these two values express the time since January 6th, 1980. The GPS Week represents a week number, and the GPS time represents the number of seconds into a week.
Flight line Edge	1 for points on the edge, 0 otherwise.	The flight line edge value is a flag for points that lie on the edge of the scan, along the flight line.

LIDAR Quick Facts

Format Type Identifier	LIDAR
Reader/Writer	Reader
Licensing Level	Professional
Dependencies	None
Dataset Type	File base name
Feature Type	Feature Name
Typical File Extensions	.las
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Not Applicable
Transaction Support	No
Geometry Type Attribute	lidar_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	no
circles	no		point cloud	yes
circular arc	no		polygon	no
donut polygon	no		raster	no
elliptical arc	no		solid	no
ellipses	no		surface	no
line	no		text	no
none	no		z values	no

Point Cloud Component	Data Type	Notes
fmepc_angle	REAL64	Range: -90 to 90
fmepc_classification	UINT8	
fmepc_color_r	UINT16	Only supported in version 1.1+
fmepc_color_g	UINT16	Only supported in version 1.1+
fmepc_color_b	UINT16	Only supported in version 1.1+
fmepc_flight_line_edge	UINT8	Range: 0 to 1
fmepc_flight_line	not supported	While not directly supported, flight line will be written as point source ID if point source ID does not exist on the point cloud.
fmepc_gps_time	REAL64	
fmepc_gps_week	not supported	
fmepc_intensity	UINT16	
fmepc_number_of_returns	UINT8	Range: 1 to 5
fmepc_point_source_id	UINT16	Only supported in version 1.1+
fmepc_posix_time	not supported	
fmepc_return	UINT8	Range: 1 to 5
fmepc_scan_direction	UINT8	Range: 0 to 1
fmepc_user_data	UINT8 (version 1.1+) or UINT16 (version 1.0)	

Reader Overview

FME considers a single LAS file to be a dataset. Each dataset contains a single FME point cloud feature.

Reader Directives

The directives listed below are processed by the LIDAR/LAS reader. The suffixes shown are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the LIDAR reader is **LIDAR**.

DATASET

The value for this directive is the LIDAR/LAS file to be read.

Required/Optional

Required

Mapping File Syntax

LIDAR_DATASET /usr/data/test.las

* Workbench Parameter

Source ASPRS LAS File(s)

GROUP_BY_DATASET

The value for this directive can be either Yes or No.

When the value is set to No, the only feature type this reader will use is the reader type name, which in this case is LIDAR. When the value is set to Yes, the feature type of each dataset is the filename (without the path or the extension) of the dataset. The default value for this directive is No.

Required/Optional

Required

Mapping File Syntax

LIDAR_DATASET /usr/data/test.las

* Workbench Parameter

Source ASPRS LAS File(s)

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

`<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>`

* **Workbench Parameter**

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

`<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]`

* **Workbench Parameter**

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* **Workbench Parameter**

Additional Attributes to Expose

Writer Overview

FME considers a dataset to be a directory name. The feature type of each dataset is the filename. The LIDAR writer distinguishes duplicate output files by appending numbers to the filenames.

Writer Directives

The directives listed below are processed by the LIDAR/LAS writer. The suffixes shown are prefixed by the current `<writerKeyword>` in a mapping file. By default, the `<writerKeyword>` for the LIDAR writer is `LIDAR`.

DATASET

The value for this directive is the path of the output directory where the data will be written..

Required/Optional

Required

Mapping File Syntax

LIDAR_DATASET /usr/data/

* Workbench Parameter

Destination ASPRS LAS Directory

VERSION

The version of the LAS file to be written.

Required/Optional

Optional

Values

1.0 | 1.1 | 1.2 (default)

Mapping File Syntax

LIDAR_DATASET VERSION 1.1

* Workbench Parameter

ASPRS LAS Version

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Attribute Name	Contents
lidar_type	This will always be lidar_point_cloud.
lidar_file_creation_date	The date on which this file was created (LAS 1.1 and 1.2), or the date on which the data was collected (LAS 1.0).
lidar_file_source_id	The file source ID. A value of zero is interpreted to mean that an ID has not been assigned.
lidar_generating_software	Description of the generating software.
lidar_project_id	A complete Globally Unique Identifier to serve as a project ID. By assigning a Project ID and using a File Source ID (defined above) every file within a project and every point within a file can be uniquely identified, globally.
lidar_system_identifier	A string identifying the hardware system or operation that generated the data.
lidar_version	The version of the LAS file.
lidar_vertical_coordsys_code	The GeoTIFF code identifying the vertical coordinate system.
lidar_vertical_datum_code	The GeoTIFF code identifying the vertical datum.
lidar_vertical_units_code	The GeoTIFF code identifying the units of the vertical coordinate system.

Australian Asset Design & As Constructed (ADAC) XML Reader

Format Notes: This format is not supported by FME Base Edition.

The ADAC XML format is developed by the Asset Design & As Constructed (ADAC) consortium. This reader supports the ADAC XML version 3.0.1 and 4.0.0.

Further information on ADAC can be found at <http://www.adac.com.au>.

Overview

An ADAC XML document consists of a root ADAC element containing various data structures from civil engineering assets.

ADAC v3 defines the following asset themes:

- Sewerage
- Roads
- Water
- Stormwater
- Cadastre

For backwards compatibility, the interpretation/mapping of the ADAC v3 assets remains unchanged.

ADAC v4 defines the following asset themes:

- Sewerage
- Transport
- WaterSupply
- StormWater
- OpenSpace
- Cadastre
- Surface
- Enhancements
- Supplementary

ADAC Quick Facts

Format Type Identifier	ADAC
Reader/Writer	Reader
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	The ADAC asset structures
Typical File Extensions	.xml
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Never
Schema Required	No
Transaction Support	No
Geometry Type	xml_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	yes		polygon	yes
circular arc	yes		raster	no
donut polygon	yes		solid	no
elliptical arc	yes		surface	no
ellipses	yes		text	yes
line	yes		z values	yes
none	yes			

Reader Overview

The ADAC reader creates FME features from the various ADAC asset structures.

The reader now supports ADAC v4. The asset structures in v4 are mapped differently from the ADAC v3.

See the Feature Representation section for details.

Coordinate Systems

FME ADAC features are tagged with a coordinate system when the reader finds a mapping between the name specified in the ADAC <HorizontalCoordinateSystem> element and an FME coordinate system name. The ADAC <Hori-

zonalCoordinateSystem> is a child of the <CoordinateSystem> element which is a child of the ADAC <Project> element.

Reader Directives

The suffixes shown are prefixed by the current <ReaderKeyword> in a mapping file. By default, the <ReaderKeyword> for the ADAC reader is **ADAC**.

DATASET

The location of the ADAC file to be read.

Mapping File Syntax

```
ADAC_DATASET c:\data\adac_sample.xml
```

Required/Optional

Required

* Workbench Parameter

Source Australian ADAC XML File(s)

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Feature Representation

ADAC v3

The ADAC XML reader recognizes the following ADAC v 3.0.1 asset structures:

- **Sewerage asset:** Manhole, PipeNonPressure, PipePressure, Valve, Fitting, House_Connection.
- **Roads asset:** Pavement, Parking, RoadEdge, RoadIsland, RoadPathway, PramRamp, RoadSubsoilDrain.
- **Water asset:** Pipe, Valve, Hydrant, Meter, Fittings, Maintenance_Hole.
- **Stormwater asset:** ManholePit, EndStructure, Pipe, SurfaceDrainage.
- **Cadastre asset:** LandParcel.
- **OtherData asset:** Object.

The feature type names for ADAC FME features closely resemble the naming for the ADAC v3 XML document. Feature type names are whitespace separated combinations of theme and asset structure names. For example, "Sewerage Valve", "Water Valve", "Stormwater Pipe", and "Cadastre LandParcel".

Attribute names also closely resemble their XML counterpart. The non-geometrical, non-repeating, and non-nested child elements, i.e., the simple type elements whose maxOccurs is 1, of an asset structure are mapped with their names unchanged. For example, the simple type child element <OutletType> of the <ManholePit> element is mapped as the "OutletType" attribute of the "Stormwater ManholePit" feature.

Non-geometrical nested child elements, i.e., the complex type elements, of an asset are mapped as whitespace separated combinations of child and descendant elements. For example, the <ChamberSize> complex type child element of the <ManholePit> element creates the following FME attributes in a "Stormwater ManholePit" feature: "ChamberSize Blankend", "ChamberSize PS", "ChamberSize Rectangular" and "ChamberSize Circular".

Most non-geometrical child elements in an asset structure are non-repeating, these are mapped as simple, atomic FME attribute values. Repeating non-geometrical child elements, i.e., those with maxOccurs greater than 1 or unbounded, such as the <ComponentInfo> element, are mapped as CSV values.

The geometry for a feature is mapped from the various ADAC geometry elements. Some ADAC asset structures, such as the "RoadSubsoilDrain", may have more than one geometry, these are mapped as aggregates. Two special FME geometry traits are assigned to the geometries to help identify their original ADAC role. The "adac_geometry" trait identifies the original ADAC XML geometry element, while the "adac_geometry_parent" trait identifies the geometry's parent.

Data not explicitly defined in the ADAC schema is supported in the ADAC XML via the "OtherData" asset model. This <OtherData> asset model element can contain one or more <Layer> child elements, and each <Layer> can contain one or more <Object> elements. The FME ADAC reader maps these <Object>s into a single "Object" feature type.

All FME ADAC "Object" features carry a "LayerName" attribute which identifies its original <Layer> container in the XML document. An "Object" feature can be a point, line, area, text geometry, or an aggregate of these geometries.

An <Object> element may have zero or more <ComponentInfo> child elements. The information for the first <ComponentInfo> element, as with the other predefined ADAC structures which can have at most one <ComponentInfo>, is represented in the FME ADAC feature by the following attributes:

"ComponentInfo InfrastructureCode",
"ComponentInfo Status",
"ComponentInfo Notes", and
"ComponentInfo Supporting_Info".

All of the above except for the "ComponentInfo Status" attribute have CSV values. Multiple <ComponentInfo> elements in a single layer are represented in the FME ADAC feature as a structure list attribute, except for the list indexing the representation is similar to the above:

"ComponentInfo{0}.InfrastructureCode",
"ComponentInfo{0}.Status",
"ComponentInfo{0}.Notes", and
"ComponentInfo{0}.Supporting_Info".

"ComponentInfo{1}.InfrastructureCode",
"ComponentInfo{1}.Status",
"ComponentInfo{1}.Notes", and
"ComponentInfo{1}.Supporting_Info".

Note that the ComponentInfo list attribute with index 0 is identical to the non-indexed set of ComponentInfo attributes.

ADAC v4

The ADAC XML reader parses the ADAC v4.0.0 schema to determine the asset structures available to be read.

In ADAC v4 the FME feature type names also closely resemble, as in V3, the ADAC v4 XML element asset names. ADAC FME feature type names are whitespace separated combinations of theme and asset names. For example, the <Pit> asset in the <Stormwater> theme is mapped as the FME "Stormwater Pit" feature type. Note that the <Pits> element,

the element representing the feature class of stormwater pits, i.e., the parent element enclosing the <Pit> elements, is not part of the FME feature type.

Most non-geometrical child elements in an asset structure are non-repeating, these are mapped as simple, atomic FME attribute values. These elements are mapped as FME attributes with their name unchanged. For example, the simple type child element <PitNumber> of the <Pit> element is mapped as the "PitNumber" attribute of the "Stormwater Pit" feature.

Repeating non-geometrical child elements, i.e., those with maxOccurs greater than 1 or unbounded, such as the <SupportingFile> element, is mapped as a list attribute. The <SupportingFile> element is part of the <ComponentInfo> element, which is an element defined as a complex type, hence <SupportingFile> is mapped in FME as a nested list attribute, "ComponentInfo.SupportingFiles.SupportingFile{ }".

Non-geometric nested child elements, i.e., the complex type elements, of an asset structure are mapped differently from v3, in ADAC v4 these complex property elements are mapped as FME nested list attributes. For example, the <ChamberSize> complex type child element of the <Pit> element creates the following FME nested list attributes in a "Stormwater Pit" feature:

```
"ChamberSize.Rectangular.Length_mm"  
"ChamberSize.Rectangular.Width_mm"  
"ChamberSize.Circular.Diameter_mm"  
"ChamberSize.Extended.Radius_mm"  
"ChamberSize.Extended.Extension_mm"
```

The geometry for a feature is mapped from the various ADAC <Geometry> elements. Unlike ADAC v3, the assets data structure for ADAC v4 do not have multiple geometries.

Geometry

Note: ADAC v4 assets have exactly one geometry. The following applies only to ADAC v3.

ADAC v3 asset structures with multiple geometry elements are mapped into FME geometry aggregates. FME geometry traits are used to help identify the original ADAC role. Two special geometry traits are assigned, the "adac_geometry" trait identifies the original ADAC XML geometry element, while the "adac_geometry_parent" trait identifies the geometry's parent. The values for these two traits are their respective ADAC XML element names. The following example logs a "Stormwater ManholePit" feature:

```
+++++  
Feature Type: Stormwater ManholePit'  
Attribute(encoded: utf-16): ChamberConstruction' has value Precast'  
Attribute(encoded: utf-16): ChamberSize Circular Diameter' has value 1050'  
Attribute(encoded: utf-16): Construction Date' has value 2007-04-14'  
Attribute(encoded: utf-16): Drawing Number' has value B02166-C66'  
Attribute(encoded: utf-16): FireRetardant' has value false'  
Attribute(encoded: utf-16): InletStructure Depth_m' has value 1.35'  
Attribute(encoded: utf-16): InletStructure InvertLevel_m' has value 6.030'  
Attribute(encoded: utf-16): InletStructure LidType' has value CIRC CAST IRON'  
Attribute(encoded: utf-16): InletStructure PitNumber' has value 13/8'  
Attribute(encoded: utf-16): InletStructure SurfaceLevel_m' has value 7.380'  
Attribute(encoded: utf-16): OutletType' has value Dry'  
Attribute(encoded: utf-16): Owner' has value Council'  
Attribute(encoded: utf-16): Project Name' has value '  
Attribute(encoded: utf-16): Use' has value Manhole'  
Attribute(string) : fme_geometry' has value fme_aggregate'  
Attribute(string) : fme_type' has value fme_point'  
Attribute(string) : xml_type' has value xml_aggregate'  
Coordinate System: '  
Geometry Type: IFMEAggregate  
Number of Geometries: 2  
-----  
Geometry Number: 0  
Geometry Type: IFMEPoint  
Number of Geometry Traits: 2  
GeometryTrait(encoded: utf-16): adac_geometry' has value Location'  
GeometryTrait(encoded: utf-16): adac_geometry_parent' has value InletStructure'
```

```
Coordinate Dimension: 3
(529958.46299999999,6942011.182,0)
```

```
-----
Geometry Number: 1
Geometry Type: IFMEPoint
Number of Geometry Traits: 2
GeometryTrait(encoded: utf-16): adac_geometry' has value Location'
GeometryTrait(encoded: utf-16): adac_geometry_parent' has value ManholePit'
Coordinate Dimension: 3
(529958.46299999999,6942011.182,0)
=====
```

The ADAC v3 <ManholePit> element has two descendant <Location> elements, one is an immediate child element, while the other grandchild element. In the above example, the "adac_geometry_parent" trait on the point geometries can be used to identify the geometry's original role in the ADAC XML document, the first location refers to the <Inlet-Structure> element, while the second refers to the <ManholePit> element.

Mapped rotated point geometries include an additional "Rotation" trait, illustrated by the log of the following "Water Valve" feature:

```
+++++
Feature Type: Water Valve'
Attribute(encoded: utf-16): Construction Date' has value 2007-04-14'
Attribute(encoded: utf-16): Drawing Number' has value B02166-C66'
Attribute(encoded: utf-16): Owner' has value Council'
Attribute(encoded: utf-16): Project Name' has value '
Attribute(encoded: utf-16): Size_mm' has value 150'
Attribute(encoded: utf-16): Type' has value Gate'
Attribute(encoded: utf-16): Use' has value Control'
Attribute(string) : fme_geometry' has value fme_point'
Attribute(string) : fme_type' has value fme_point'
Attribute(string) : xml_type' has value xml_point'
Coordinate System: '
Geometry Type: IFMEPoint
Number of Geometry Traits: 3
GeometryTrait(encoded: utf-16): Rotation' has value 83.496'
GeometryTrait(encoded: utf-16): adac_geometry' has value Location'
GeometryTrait(encoded: utf-16): adac_geometry_parent' has value Valve'
Coordinate Dimension: 3
(529952.79399999999,6942138.1310000001,0)
=====
```

The geometry for ADAC v3 and v4 features may be identified by the xml_type attribute. The valid values for this attribute are:

xml_type	Description
xml_no_geom	FME Feature with no geometry.
xml_point	Point geometry.
xml_line	Linear geometry.
xml_area	Simple polygon geometry
xml_text	Annotation geometry
xml_aggregate	An aggregate of the above geometries.

No Geometry

xml_type: xml_no_geom

Features with their xml_type attribute set to xml_no_geom do not contain any geometry data.

Points

xml_type: xml_point

Features with their xml_type set to xml_point are single coordinate features or an aggregate of single points.

Lines

xml_type: xml_line

Features with their xml_type set to xml_line are polyline features or an aggregate of polylines.

Areas

xml_type: xml_polygon

Features with their xml_type set to xml_polygon are polygon features which may or may not have interior boundaries, or an aggregate of such polygons.

Annotation

xml_type: xml_text

Features with their xml_type set to xml_text are feature with annotation geometry. The text geometry is mapped from the ADAC "annotation_geometry" complex type. The "annotation_geometry" components: <Text>, <Location>, including its <Rotation>, and <Height_m> are loaded into the FME text geometry's "text string", "point geometry", "text rotation", and "text height", respectively, other "annotation_geometry" child elements, such as "Justification", "FontName" and "Width_m", do not map cleanly into the FME text geometry and are thus mapped as geometry traits.

The following is an ADAC "Object" feature with an annotation geometry, notice that all of the "annotation_geometry" components, such as <Text>, <Rotation>, <FontName, etc,..., are also represented as geometry traits:

```
+++++
Feature Type: Object'
Attribute(encoded: utf-16) : Construction Date' has value '
Attribute(encoded: utf-16) : Drawing Number' has value '
Attribute(encoded: utf-16) : LayerName' has value CADASTRE_CANCELLED_LOTPLAN'
Attribute(encoded: utf-16) : Owner' has value Council'
Attribute(encoded: utf-16) : Project Name' has value Test.dwg'
Attribute(string) : fme_geometry' has value fme_point'
Attribute(indirect: 64 bit real) : fme_rotation' has value 81'
Attribute(indirect: 64 bit real) : fme_text_size' has value 1.5'
Attribute(indirect: encoded: utf-16): fme_text_string' has value My Annotated text.'
Attribute(string) : fme_type' has value fme_text'
Attribute(64 bit real) : xml_rotation' has value 81'
Attribute(64 bit real) : xml_text_size' has value 1.5'
Attribute(encoded: utf-16) : xml_text_string' has value My Annotated text.'
Attribute(string) : xml_type' has value xml_text'
Coordinate System: '
Geometry Type: IFMEText
Number of Geometry Traits: 8
GeometryTrait(encoded: utf-16): FontName' has value Consolas'
GeometryTrait(encoded: utf-16): Height_m' has value 1.5'
GeometryTrait(encoded: utf-16): Justification' has value Left-Bottom'
GeometryTrait(encoded: utf-16): Rotation' has value 81'
GeometryTrait(encoded: utf-16): Text' has value My Annotated text.'
GeometryTrait(encoded: utf-16): Width_m' has value 2.0'
GeometryTrait(encoded: utf-16): adac_geometry' has value Annotation'
GeometryTrait(encoded: utf-16): adac_geometry_parent' has value Object'
Text String: My Annotated text.
Text Size: 1.5
Text Rotation (degrees CCW): 81
Geometry Type: IFMEPoint
Number of Geometry Traits: 2
GeometryTrait(encoded: utf-16): adac_geometry' has value InsertionPoint'
GeometryTrait(encoded: utf-16): adac_geometry_parent' has value Location'
Coordinate Dimension: 3
(511337.43646974798,7033866.8198164497,0)
=====
```

Aggregates

xml_type: xml_aggregate

Features with their `xml_type` set to `xml_aggregate` are aggregate features whose members maybe point, line, area, annotation or aggregate geometries.

Autodesk 3ds Writer

Format Notes: This format is not available in FME Base Edition.

Overview

The 3ds Writer allows FME to read and write Autodesk® 3ds Format (3ds) files.

The 3ds format was originally developed as the native format for Autodesk 3D Studio (Releases 1 to 4). It is now commonly used as an interchange format between different 3D modelling and rendering applications.

3ds Quick Facts

Format Type Identifier	3DS
Reader/Writer	Both
Licensing Level	Professional
Dependencies	None
Dataset Type	Reader: File Writer: Directory
Feature Type	3DS_ELEMENT
Typical File Extensions	.3ds
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	Never
Enhanced Geometry	Yes
Geometry Type Attribute	3ds_type

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	yes	point	no
circles	yes	polygon	yes
circular arc	no	raster	no
donut polygon	yes	solid	yes
elliptical arc	no	surface	yes
ellipses	yes	text	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
line	no		z values	yes
none	yes			

Writer Overview

The 3D model has a hierarchical structure of Nodes, which are elements of the model.

For each node, there is a corresponding mesh, which contains the geometry of the object. Feature types become Nodes. Features become Meshes that may have geometries and attributes.

The 3ds Writer does not support feature type fanout.

Note: The 3ds format is limited to 32-bit precision for its coordinates and, as a result, translations involving a greater level of precision (i.e., using world coordinates instead of local coordinates) may produce 3ds data where different coordinates are collapsed into a single coordinate. You can resolve this issue by offsetting the x,y,z coordinates such that the model's origin is moved to (0,0,0) or another point close to this, which has the effect of moving the model into a local coordinate system.

Writer Directives

This section lists the directives that are processed by the 3ds Writer.

The suffixes shown are prefixed by the current <WriterKeyword>_ in a mapping file. By default, the <WriterKeyword> for the 3ds writer is 3DS.

DATASET

Required/Optional

Required

Values

The value for this directive is the path to the output directory. If the output directory does not exist, then the writer will create a new directory.

The output file will be created within the specified directory and associated texture files, if any, will be written to the same directory.

For example, if the output directory is *C:\3dsFiles\house* then the output file will be *C:\3dsFiles\house\house.3ds*. If the output file already exists, then the writer will overwrite it.

If any other applications have the output file opened, then the writer will be unable to continue and the translation will fail.

Workbench Parameter

Destination Autodesk 3ds Directory

DEF

The 3ds Writer ignores this directive because the format itself does not have a notion of layers/feature types.

Workbench Parameter

Not applicable

Required/Optional

Optional

MOVE_TO_LOCAL_COORDSYS

If the value is PRJ_ONLY, a companion.prj file containing the coordinate system and having the same name as the .3ds file will be written in the same directory as the .3ds file.

If the value is Yes, in addition to writing the.prj file as in the PRJ_ONLY option, a companion.fwt file with the same name as the .3ds file will be written in the same directory as the .3ds file. The coordinates of all the points in the written features will be normalized to the interval [-0.5, 0.5] on the largest side of their XYZ-bounding cube.

The other dimensions will be scaled proportionally. The transformation matrix required to scale the model back to world coordinates is contained in the .fwt file. This can be used to improve precision of the written coordinates.

Required/Optional

Optional

Values

Yes | No (default)

✳ Workbench Parameter

Parameter Name

Move to Local Coordinate System

Parameter Values

Yes | No (default) | PRJ Only

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

3ds features consist of geometry and attributes. The attribute names are defined in the DEF line and there is a value for each attribute in each 3ds feature.

In addition, each 3ds feature contains several special attributes to hold the type of the geometric entity and its display parameters. All 3ds features contain a *3ds_type* attribute, which identifies the geometric type. Depending on the geometric type, the feature contains additional attributes specific to the geometric type. These are described in subsequent sections.

Geometries with no Z coordinates (2D geometries) will be assigned 0 as their z values.

The following format-specific attributes are applicable to all geometry types, and these attributes do not appear as user attributes in the output data:

Attribute Name	Contents
3ds_mesh_name	This is an optional attribute that contains the name of the mesh read from the 3ds file. The name must not contain more than 8 characters.

These attributes may be set on the feature or on the geometry of the feature at any level.

If some of the attributes are not set on a certain geometry, they will take the values set the geometry's container. If the values are not found, they will take the values set on the feature. If the values are not found at the feature level, they will assume default values.

If a certain geometry and its container has different values for the same attribute, the value on the geometry, not its container, will be used.

Mesh

3ds_type: 3ds_mesh

Meshes are composed of triangular faces. If the input mesh contains faces with more than three distinct vertices, then the face will be converted into multiple triangular faces. The triangular faces of a mesh need not be connected.

Polygons and donuts are treated as meshes. They will be converted into triangular faces that represents the inner area of the polygon or donut.

The name of a mesh read by the reader will be stored in (3ds_mesh_name) as a string. The name of a mesh produced by the writer is a unique number.

Material and Appearance

If the feature being read does not contain a valid 3ds material reference, the appearance on the individual face in the mesh will be set to FME's default appearance. Any raster referenced as a texture in the 3ds file will be read by FME, as long as the source format is supported by FME.

If the feature being written does not contain a valid appearance reference, the default material will be assigned to the corresponding faces. If the feature contains a valid appearance reference, it will be written as faces referenced to a corresponding 3ds material.

A two-sided surface with matching appearance references will be written out as two-sided faces sharing one material in 3ds. Due to a limitation within 3ds, a two-sided surface with different appearance references will be written out as two one-sided faces with different materials.

If the incoming feature contains deprecated attributes such as material name (3ds_material), color (3ds_ambient_color, 3ds_diffuse_color, or 3ds_specular_color), or texture image (3ds_texture_image) information, a material with these properties will be created and assigned to the mesh corresponding to the feature.

The material name in 3ds is limited to 8 characters; the writer will truncate appearance names longer than 8 characters.

Autodesk AutoCAD DWF Reader/Writer

The AutoCAD® Reader/Writer enables FME to read and write files used by Autodesk® AutoCAD and compatible systems. AutoCAD drawing files consist of drawing settings and configuration, as well as a series of entities, or graphic elements, organized into layers.

FME provides broad support for many AutoCAD entity types and options and for reading and writing AutoCAD file version up to and including 2007. When AutoCAD data is output, header information may be copied from a supplied template, or prototype, file.

This chapter assumes familiarity with AutoCAD-compatible systems and the entities (features) that are manipulated within these systems.

Note: Throughout this chapter, the AutoCAD file is referred to as a drawing file rather than a DWF file.

Overview

There are three supported formats used by AutoCAD:

- DXF (drawing exchange format) files, which are large ASCII files,
- DWG (drawing) files, which are binary and support the most entity types, and
- DWF (drawing web format) files, which are binary files of reduced size and functionality intended for display on limited-bandwidth mediums such as the Internet.

Logically, both DWG and DXF files are identical and, therefore, FME treats both file types in the same manner. DWG/DXF files are read by the AutoCAD DWG/DXF reader and writer (see [Autodesk AutoCAD DWF/DXF Reader/Writer](#)).

DWF files are handled seamlessly but internally they undergo a different series of translation processes. These are read separately by the AutoCAD DWF reader and writer. Currently the DWF reader and writer can only read and write two-dimensional (2D) DWF files.

This document covers information specific to AutoCAD DWF files. For general AutoCAD DWG/DXF information and AutoCAD feature types supported by FME, please refer to the documentation on the [AutoCAD DWG/DXF Reader/Writer](#).

AutoCAD DWF Quick Facts

Format Type Identifier	DWF
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	File
Feature Type	Layer name
Typical File Extensions	.dwf
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	Yes
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type Attribute	autocad_entity

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	yes		polygon	yes
circular arc	yes		raster	yes
donut polygon	yes		solid	yes
elliptical arc	yes		surface	yes
ellipses	yes		text	yes
line	yes		z values	no
none	no			

Reader Overview

The AutoCAD DWF reader extracts entities, one at a time, from the entity section of the drawing file and passes them on to the rest of the FME for processing. Complex entities such as polylines and inserts are exploded and broken into several individual FME features. If the entity has attribution stored as extended entity data, then this is also read and placed in the feature.

When the AutoCAD reader encounters an entity type it does not know how to process, it simply sets the entity type of the feature and returns it. This feature is then logged by the FME correlation subsystem and the reader moves on to the next entity.

Reader Directives

This section describes the directives that are recognized by the AutoCAD DWF reader. Each directive is prefixed by the current <ReaderKeyword>_ when placed in a mapping file.

DATASET

Required/Optional: *Required*

The dataset into which feature data is to be read.

Workbench Parameter: *Source Autodesk AutoCAD DWF File(s)*

PASSWORD

Required/Optional: *Optional*

This statement specifies the password to open the DWF file for reading if it is password protected. The statement is of the following form:

```
<writerKeyword>_PASSWORD <autocad dwf password>
```

The statement below instructs the AutoCAD reader try to open the given dataset with the password "mypass":

```
DWF_PASSWORD mypass
```

Value: *<valid password>*

Default value: *no password*

Workbench Parameter: *Password*

PAPER_WIDTH, PAPER_HEIGHT

Required/Optional: *Optional*

These statements specify the maxima of the width and height in millimeters for the sheets read from the input DWF file.. The statements are of the following form:

```
<writerKeyword>_PAPER_WIDTH <width in mm>  
<writerKeyword>_PAPER_HEIGHT <height in mm>
```

The statements below instruct the AutoCAD reader limit the extents of the sheets read from the input DWF file to 297mm by 210 mm:

```
DWF_PAPER_WIDTH 297  
DWF_PAPER_HEIGHT 210
```

Paper_Width Value: *<Valid positive numeric>*

Paper_Width Default Value: *297*

Paper_Height Value: *<Valid positive numeric>*

Paper_Height Default Value: *210*

Workbench Parameter: *Width (mm) and Height (mm)*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The AutoCAD writer provides the following capabilities when writing AutoCAD files.

- **Password security:** Passwords can be created for DWF files written by FME.
- **Multiple file formats:** DWF files can be written as either compressed or uncompressed binary files, or as ASCII files.
- **Resolution:** DWF file resolution can be determined at translation time in the form of the X Size and Y Size writer keywords. These are specified in pixels.
- **Color Map Optimization:** Colors in the color map that are unused can be eliminated from the output file to reduce space.
- **Export Invisible Layers:** Invisible Layers can be optionally exported if desired. (Not supported for Write 3D)
- **Force View Extents:** The initial viewport of the output file can be overwritten to include the entire extents of the drawing instead of the default active viewport.
- **Use Inked Area:** The inked area of the DWF file can be calculated from the entities in the file to produce a tight bounding rectangle around drawable graphic entities.
- **Skip Layer Info:** Additional layer information can be omitted from the output DWF file to reduce space.
- **Skip Named Views:** Named viewports can be omitted from the output DWF file in order to save space.
- **Multi-version Support:** The AutoCAD DWF writer supports files that are compatible with any current AutoCAD release.

When creating AutoCAD DWF files, the AutoCAD writer first defines the linetypes and layers defined within the FME mapping file. The writer then reads in a template file, if specified, and copies the linetypes, layer definitions, shape file header information, and block information from the template file to the output dataset.

The AutoCAD writer then outputs each feature it is given to the output file in the appropriate entity type.

When writing an AutoCAD DWF file, the format of file output is determined as follows:

- If the file name contains .dwf or .DWF, then the output dataset is written in DWF format.
- Otherwise, if an error exists in the mapping file, the translation is halted.

Writer Directives

This section describes the directives processed by the AutoCAD DWF writer module. Each of the directives is prefixed by the current <WriterKeyword>_ when they are placed in a mapping file. By default, the <WriterKeyword> for the AutoCAD DWF writer is DWF.

DATASET

Required/Optional: *Required*

The dataset into which feature data is to be written.

Workbench Parameter: *Destination Autodesk AutoCAD DWF File*

DWF_VERSION

Required/Optional: *Optional*

The version of the AutoCAD DWF file to be produced.

The value corresponds with the release number of the AutoCAD DWF file that is produced. This statement specifies the version of AutoCAD file to be output. The statement is of the following form:

```
<writerKeyword>_DWF_VERSION <autocad dwf version>
```

The example statement below instructs the AutoCAD writer to produce a version 4.2 DWF file:

```
DWF_DWF_VERSION 4.2
```

Values: 4.2 | 5.5 | 6.0

Default value: 5.5

Workbench Parameter: *Version*

PASSWORD

Required/Optional: *Optional*

This statement specifies the password to open the DWF file for reading if it is password protected. The statement is of the following form:

```
<writerKeyword>_PASSWORD <autocad dwf password>
```

The statement below instructs the AutoCAD writer try to open the given dataset with the password "mypass":

```
DWF_PASSWORD mypass
```

Value: *<valid password>*

Default value: *no password*

Workbench Parameter: *Password*

TEMPLATEFILE

Required/Optional: *Optional*

The name of an existing AutoCAD DWF file that contains the block definitions and linetype definitions to be used when creating the output dataset.

Value: *<valid password>*

Default value: *no password*

Workbench Parameter: *Template File*

TEMPLATEFILE_PASSWORD

Required/Optional: *Optional*

DWF files support an optional password for additional security. If specified, the given password is provided when the file is opened.

Value: *<valid password>*

Default value: *no password*

Workbench Parameter: *Template File Password*

FORMAT

Required/Optional: *Optional*

The format of the AutoCAD DWF file to be produced.

Values: *COMPRESSED_BINARY | UNCOMPRESSED_BINARY | ASCII*

Default value: *COMPRESSED_BINARY*

Workbench Parameter: *Format*

X_SIZE

Required/Optional: *Optional*

Specifies the horizontal width of the output DWF file in pixels.

Values: *<valid positive numeric>*

Default value: *36000*

Workbench Parameter: *X Size*

Y_SIZE

Required/Optional: *Optional*

Specifies the vertical height of the output DWF file in pixels.

Values: *<valid positive numeric>*

Default value: *24000*

Workbench Parameter: *Y Size*

OPTIMIZE_COLOR_MAP

Required/Optional: *Optional*

This directive, if set, prevents unused colors in the color map from being stored in the DWF file.

Value: *YES | NO*

Default value: *NO*

Workbench Parameter: *Optimize Colormap*

Example:

```
DWF_OPTIMIZE_COLOR_MAP YES
```

EXPORT_INVISIBLE_LAYERS

Required/Optional: *Optional*

This statement exports invisible layers in an AutoCAD file to be output to the DWF file.

Value: *YES | NO*

Default value: *NO*

Workbench Parameter: *Export Invisible Layers*

Example:

```
DWF_EXPORT_INVISIBLE_LAYERS YES
```

FORCE_VIEW_TO_EXTENTS

Required/Optional: *Optional*

This directive, if set, sets the initial viewport of the DWF file to the entire extents instead of the last actively seen viewport.

Value: *YES | NO*

Default value: *NO*

Workbench Parameter: *Force Initial View to Extents*

Example:

```
DWF_FORCE_VIEW_TO_EXTENTS YES
```

USE_INKED_AREA

Required/Optional: *Optional*

This directive, if set, calculates a tight bounding area around the graphic elements of a drawing.

Value: *YES | NO*

Default value: *NO*

Workbench Parameter: *Use Inked Area*

Example:

```
DWF_USE_INKED_AREA YES
```

SKIP_LAYER_INFO

Required/Optional: *Optional*

This directive, if set, prevents additional layer information from being stored in the DWF file.

Value: *YES | NO*

Default value: *NO*

Workbench Parameter: *Skip Layer Info*

Example:

```
DWF_SKIP_LAYER_INFO YES
```

SKIP_NAMED_VIEWS

Required/Optional: *Optional*

This directive, if set, prevents named views from being stored in the DWF file.

Value: *YES | NO*

Default value: *NO*

Workbench Parameter: *Skip Named Views*

Example:

```
DWF_SKIP_NAMED_VIEWS YES
```

DEF

Required/Optional: *Optional*

The AutoCAD DWF writer requires that every feature written to the AutoCAD file be stored within a predefined AutoCAD layer. In AutoCAD, the layers are used to store collections of logically related attributes. Within the FME, the AutoCAD layer and the type of the feature are treated synonymously as there is a one-to-one correspondence between FME feature type and AutoCAD layer.¹The order of properties in the layer statement is required as shown, though additional attribute name and type pairs may be in any order. The layer statement is of the following form:

```
<WriterKeyword>_DEF <layer name> \  
    autocad_color <default color> \  
    autocad_linetype <default linetype>\  
    [autocad_layer_type frozen] \  
    [<attribute name> <attribute type>]
```

where:

- <layer name> is the name of the layer being defined. This is the name that is used throughout the remainder of the FME mapping files.
- <default color> is the color number used for all features stored within the layer unless explicitly overridden on the correlation lines below. Valid values are between 1 and 255.
- <default linetype> is the name of the linetype to use for the layer if no linetype is specified on the correlation line. The linetype specified must either be:
 - defined in the mapping file,
 - copied from a specified template file, or
 - the predefined linetype named CONTINUOUS.
- <autocad_layer_type> is the type of layer to create. Currently, only the value frozen is supported. If specified, then the created layer is frozen; otherwise, the layer is not frozen.
- <attribute name> <attribute type> is the definition of an attribute to be stored within the extended entity data of features for the layer. If no attributes are defined, then all feature attributes (except those that start with autocad_) are stored. The storing of attributes can be turned off by specifying a value of external_attributes for the autocad_attributes feature attribute on the correlation line. The values for <attribute type> are the same as those for ESRI Shapefiles.

The example below defines a layer called boundary in which entities are drawn using color 13 (unless otherwise specified) and a linetype called dash-dot (unless otherwise specified). The feature also has several attributes specified that will be written to the extended entity data of each feature within the layer.

```
DWF_DEF boundary \  
    autocad_color 13 \  
    autocad_linetype dash-dot \  
    FEATCODE char(12) \  
    PPID char(10) \  
    DATECHNG date \  
    SURVEYDIST number(8,2)
```

¹Layers can also be defined through the use of a TEMPLATEFILE.

Feature Representation

Special FME feature attributes are used to hold AutoCAD entity attributes. The AutoCAD writer uses these attribute values as it fills in an entity structure during output. The AutoCAD reader sets these attributes in the FME feature it creates for each entity it reads.

For more information on general AutoCAD entities and their representations inside FME, please see the documentation on the AutoCAD DWG/DXF reader and writer.

Autodesk AutoCAD DWG/DXF Reader/Writer

Format Notes: This format contains Autodesk® RealDWG by Autodesk, Inc.¹

This chapter contains information related to the AutoCAD DWG/DXF reader/writer and AutoCAD feature types supported by FME. For 2D AutoCAD DWF information, please refer to the *Autodesk AutoCAD DWF Reader/Writer*. For 3D AutoCAD DWF information, please refer to the *Autodesk AutoCAD 3D DWF Reader/Writer*.

The AutoCAD® Reader/Writer allows FME to read and write files used by Autodesk® AutoCAD and compatible systems. AutoCAD drawing files consist of drawing settings and configuration, as well as a series of entities, or graphic elements, organized into layers.

Overview

FME provides broad support for many AutoCAD entity types and options and for reading and writing AutoCAD file versions up to and including 2010.

When AutoCAD data is output, header information may be copied from a supplied template, or prototype, file.

RealDWG



Support for the AutoCAD files up to version 2010 has also been done with a new format type identifier labelled REALDWG. This change means that there are some parts of this document that apply to REALDWG specifically, and may be in duplication of pre-existing information about the handling of AutoCAD files. One significance of the REALDWG reading and writing of AutoCAD files is that it complies with Autodesk AutoCAD TrustedDWG™ reading and writing.

This chapter assumes familiarity with AutoCAD-compatible systems and the entities (features) that are manipulated within these systems.

Note: Throughout this chapter, the AutoCAD file is referred to as a drawing file rather than a DWF file.

What is TrustedDWG?

Users can specify whether they would like Autodesk AutoCAD 2007-2010 to notify them when the DWG file they are opening was saved using an application that was not created by an Autodesk product or RealDWG licensee.

¹Copyright © 1998-2006 Autodesk, Inc. All rights reserved.

AutoCAD DWG Quick Facts

Format Type Identifier	ACAD
Reader/Writer	Both
Dataset Type	File
Licensing Level	Base
Dependencies	None
Feature Type	Layer name
Typical File Extensions	.dwg, .dxf
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	Yes
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type Attribute	autocad_entity
Enhanced Geometry	Yes
Encoding Support	Yes

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	yes		polygon	yes
circular arc	yes		raster	no
donut polygon	yes		solid	yes
elliptical arc	yes		surface	yes
ellipses	yes		text	yes
line	yes		z values	yes
none	no			

AutoCAD RealDWG Quick Facts

Format Type Identifier	REALDWG
Reader/Writer	Both
Dataset Type	File
Licensing Level	Base
Dependencies	None
Feature Type	Layer name
Typical File Extensions	.dwg, .dxf
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	Yes
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type Attribute	autocad_entity
Enhanced Geometry	Yes
Encoding Support	Yes

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	yes		polygon	yes
circular arc	yes		raster	no
donut polygon	yes		solid	yes
elliptical arc	yes		surface	yes
ellipses	yes		text	yes
line	yes		z values	yes
none	no			

Background

There are two formats used by AutoCAD: DXF (drawing exchange format) files, which are large; and ASCII representations of the binary DWG (drawing) files. Logically, both files are identical and, therefore, the FME treats both file types in the same manner. For AutoCAD DWF reading and writing support please see the chapter on AutoCAD DWF.

AutoCAD DWG and DXF have been upgraded to take advantage of the FME enhanced geometry model. Advantages include the storage of linear and bulge arc segments of polylines and hatch loops, the storage of more arcs and

ellipses in more cases without being stroked to lines or polygons, and the reading and writing of 3D geometries. Altogether, the addition of the enhanced geometry model support increases accuracy of geometric representation in AutoCAD-to-AutoCAD translations, as well as the creation and interpretation of more accurate features when translating to or from other FME formats.

AutoCAD files consist of sections, as follows:

1. **HEADER:** This contains settings of variables associated with the drawing.
2. **CLASSES:** This contains class definitions associated with the drawing.
3. **TABLES:** This contains a variety of tables, including:
 - **Layers:** Each layer entry contains layer definition information such as layer color, layer name, and layer linetype. The AutoCAD reader validates the layer names and may modify them to remove invalid characters.
 - **Linetypes:** Each linetype entry contains the linetype definition information such as name and alignment. The AutoCAD writer enables linetype definitions to be copied from an existing AutoCAD file, then referenced by name during the data translation.
 - **Shape Files:** Each shape file entry identifies a shape file referenced by the drawing. Shape files are used by AutoCAD as a different method for defining symbols or fonts. Note: These are similar to the TextStyles in AutoCAD.

Note: AutoCAD shape files are *not* the same thing as ESRI Shapefiles. AutoCAD shape files store symbol and font definitions.

- **Applications:** Each application entry contains the name of an application referenced within the AutoCAD file.
4. **BLOCKS:** These are used to define symbols and other drawing file objects used repeatedly throughout a drawing. The AutoCAD writer enables copying of block definitions from an existing AutoCAD file, which is then referenced by name during a data translation operation.
 5. **ENTITIES:** This is the main section of a drawing file and contains the actual feature entities. Each entity contains standard information, such as its color, layer, thickness, linestyle, and geometry, as well as a number of attributes specific to its entity type. For example, a text entity has fields for font, size, and the text string in addition to the standard display attributes.

Note: FME supports both 2D and 3D AutoCAD entities. However, many applications only support 2D DWG and DXF files. The @Force2D function can be used to ensure that only 2D data is written to an output DWG or DXF file.

6. **OBJECTS:** This section stores dictionaries and other helper non-entity objects.

Each entity may also have associated attribution stored within an extended entity data section. Extended entity data is fully supported by the FME.

All coordinates within a drawing file are stored as 64-bit floating point values in world coordinates. As such, there is no need to scale or otherwise alter coordinates as they are being read from or written to a drawing file.

The AutoCAD reader and writer use symbolic names for the different entity types stored within a drawing file. This simplifies feature type specification. The following table gives a brief description of each of the different AutoCAD entity types currently supported by the reader and/or writer. The entities are described in detail in subsequent sections.

AutoCAD Entity Types and Descriptions

FME autocad_entity	Description
autocad_line	Linear features stored within drawing file as a line or unclosed polyline.
autocad_point	Point features.
autocad_xline	Linear features of type xline.
autocad_ellipse	Features with an elliptical or circular representation.
autocad_shape	Features whose representation is stored in an AutoCAD shape file.
autocad_polygon	Features whose geometry is represented by a closed polyline.
autocad_face	Features represented by a 3D face object. The face object may have 3 or 4 coordinates.
autocad_arc	Features whose geometry represents a portion of a circular arc.
autocad_trace	Features with a 4 coordinate trace geometry.
autocad_solid	Features with a 3 or 4 coordinate solid geometry.
autocad_ray	Features with a linear geometry which represents a ray.
autocad_text	Text features.
autocad_spline	Spline features.
autocad_multi_text	Text features that store multiple lines of text. R14 and later only.
autocad_multi_line	A linear feature that is represented by more two or more parallel lines. Note: R14 and later only. This is supported only by the Reader.
autocad_insert	Point features that represent the location of a block reference entity.
autocad_leader	AutoCAD Leader entity representing leader lines in drawings.
autocad_hatch	Features with 2D boundary loops which form polygons and donuts, and which may be filled with line patterns or color gradients.

FME autocad_entity	Description
autocad_mpolygon	Features with 2D polyline loops which form polygons and donuts and which may be filled with line patterns or color gradients.
autocad_surface	Features with connected and unconnected, planar and non-planar 3D areas, that may represent meshes or the boundary representations of 3D solids.
autocad_solid3d	Features with connected and unconnected, 3D geometries that may be 3D solids or their closed boundary representations.
autocad_attr_def	Features without geometry that contain information about AutoCAD attribute definition. Main information on features would be the Tag, Prompt and Default value for that attribute definition.

Reader Overview

The AutoCAD reader first reads the header and table information from the drawing file being processed, and caches information on blocks, shape files, layers, linetypes, and applications. These cached values are referenced by entities throughout the file and are needed when processing the entities.

The reader then extracts entities, one at a time, from the entity section of the drawing file and passes them on to the rest of the FME for processing. Complex entities such as polylines and inserts are extracted as single FME features. If the entity has attribution stored as extended entity data, then this is also read and placed in the feature.

When the AutoCAD reader encounters an entity type it does not know how to process, it simply sets the entity type of the feature and returns it. This feature is then logged by the FME correlation subsystem and the reader moves on to the next entity.

ESRI Product Coordinate System Information

To specify the FME coordinate system, the FME AutoCAD reader can recognize a coordinate system associated with AutoCAD data by ESRI products.

The AutoCAD reader will first look for the following files in the source directory:

- <filename>.prj
- esri_cad.prj

If neither of these files is present, the AutoCAD reader will try to find an ESRI_PRJ entry embedded in the file.

(This is not applicable to the RealDWG reader.)

Reader Directives

This section describes the directives that are recognized by the AutoCAD reader. Each directive is prefixed by the current **<ReaderKeyword>**_ when placed in a mapping file.

DATASET

Required/Optional: *Required*

The dataset from which feature data is to be read.

Workbench Parameter: *Source Autodesk AutoCAD DWG/DXF File(s)*

STORE_BULGE_INFO (only applicable with classic geometry)

Required/Optional: *Optional*

When specified, the AutoCAD Reader doesn't vectorize the Polyline and LWP Line Bulges but rather just stores the coefficients in the attribute **autocad_bulge**. In addition, when specified, this directive allows the creation of **autocad_start_width** and **autocad_end_width** attributes to represent the width properties of Polyline and LWP entities. This is generally set to **Yes** only when performing AutoCAD-to-AutoCAD translations.

Values: *YES | NO*

Default value: *NO*

Workbench Parameter: *<WorkbenchParameter>*

SPLIT_BULGE_ARCS (only applicable with classic geometry)

Required/Optional: *Optional*

When specified, the AutoCAD Reader doesn't vectorize the polylines but rather returns one feature for each arc that has a bulge in it as an **autocad_arc** feature.

When features are read using enhanced geometry this directive will be ignored. To split enhanced geometry paths use the PathSplitter transformer.

Values: *YES | NO*

Default value: *NO*

Workbench Parameter: *<WorkbenchParameter>*

STORE_SPLINE_DEFS

Required/Optional: *Optional*

When specified, the AutoCAD Reader, in addition to vectorizing the splines, stores the spline coefficients as attributes. See the description of Spline below for the attribute names used to store the spline definition. This is generally set to **Yes** when performing AutoCAD-to-AutoCAD translations.

Values: *YES | NO*

Default value: *YES*

Workbench Parameter: *Store Spline Definitions*

RESOLVE_BLOCKS

Required/Optional: *Optional*

Specifies whether the reader will resolve (or explode) the block entities when processing inserts, or if it should just treat inserts as a point feature. This is generally set to **No** when performing AutoCAD-to-AutoCAD translations.

When the reader resolves blocks, it outputs a feature for each of the AutoCAD entities that are part of the block definition. The original insert is not output. This results in the full graphical representation of the block transferred through FME. The exact insertion point of the block is lost unless the **STORE_INSERT_POINT** directive is also used.

Each block member feature is given the attribute **autocad_block_number** which is set to the same value for each block so that the features comprising each block may be combined in subsequent processing. Arbitrarily deep block nesting is permitted, however, the **autocad_block_number** attribute is only updated for each block at the outermost level. The layer of the block members is determined by the **USE_BLOCK_HEADER_LAYER** directive.

If the block contains "Attribute" then each instance of "Attribute" in the block entity will be returned as "Text" entity along with a non-spatial feature containing information about that "Attribute" definition and its value for that block.

If the exact insertion point of the block is desired, then block resolution should be turned off and the insert entities for each block should be translated into point features in the output system. Alternatively, the **STORE_INSERT_POINT** directive may be specified to keep the insert point on the attributes of block member features.

Values: *YES* | *NO*

Default value: *YES*

Workbench Parameter: *Expand Blocks into Entities*

DO_NOT_RESOLVE_BLOCKS

Required/Optional: *Optional*

This directive is an exception list of the blocks that are not to be resolved, and is processed only when `RESOLVE_BLOCKS` is specified. This is a space delimited list of the block names.

Values: *space-delimited list of block names*

Default value: *empty list*

Workbench Parameter: *Not applicable*

STORE_INSERT_POINT

Required/Optional: *Optional*

Specifies whether the reader should add the insert point location as attributes to the block component entities when resolving (or exploding) inserts entities. This is generally set to No when performing AutoCAD-to-AutoCAD translations.

When the reader resolves blocks, it outputs a feature for each of the AutoCAD entities that are part of the block definition. The original insert is not output, but this directive allows the insert location to still be represented.

This results in each block member feature having the following attributes: `autocad_block_insert_[xyz]`.

Values: *YES* | *NO*

Default value: *NO*

Workbench Parameter: *Expand Blocks into Entities*

USE_BLOCK_HEADER_LAYER

Required/Optional: *Optional*

Specifies how the reader should set the layer of the block component entities when resolving (or exploding) inserts entities.

This directive applies only if `RESOLVE_BLOCKS` is set to Yes. It is generally set to No when performing AutoCAD-to-AutoCAD translations.

When the reader resolves blocks, it outputs a feature for each of the AutoCAD entities that are part of the block definition. When set to Yes, this directive indicates that all block members will be on the same layer as that of the original block. Otherwise, the block members will appear on their respective layers.

Values: *YES* | *NO*

Default value: *YES*

Workbench Parameter: *Use Block Header Layer for Components*

RESOLVE_DIMENSIONS

Required/Optional: *Optional*

Specifies whether or not to resolve (explode) dimensions into their individual pieces. If the value is **yes**, then each piece of the dimension will be output as a separate feature. If the value is **no**, then an aggregate, containing all the pieces of the original dimension, will be output. This is generally set to **No** when performing AutoCAD-to-AutoCAD translations.

Values: *YES* | *NO*

Default value: *YES*

Workbench Parameter: *Resolve Dimensions*

PRESERVE_INSERTS

Required/Optional: *Optional*

If the value for **RESOLVE_BLOCKS** is **yes**, and this directive is also **yes**, then block insert points are output as point features.

Values: *YES* | *NO*

Default value: *NO*

Workbench Parameter: *<WorkbenchParameter>*

CONVERT_ZERO_LENGTH_ARCS_TO_POINTS

Required/Optional: *Optional*

Specifies whether a zero length arc should be converted into a point feature (i.e., **autocad_point**). If the feature becomes a point, it will still retain all the attributes it had while it was an arc. This is generally set to **no** only when performing AutoCAD-to-AutoCAD translations.

Values: *YES* | *NO*

Default value: *YES*

Workbench Parameter: *Convert Zero Length Arcs to Points*

OUTPUT_BLOCKS_AT_START

Required/Optional: *Optional*

Specifies that the reader will output all the block definitions at the beginning of the translation before any other features are output. After the blocks are output, the rest of the translation is run without the blocks being resolved. When specified, this value overrides the value specified by **RESOLVE_BLOCKS**.

When set, all features that are part of a block definition have the attribute **autocad_block_definition**, with the value of the attribute being the name of the block which they are a component.

Values: *YES* | *NO*

Default value: *NO*

Workbench Parameter: *Output Blocks at Start*

IGNORE_FROZEN_LAYERS

Required/Optional: *Optional*

Specifies whether the reader will ignore all features on the frozen layers. If set to **yes**, then features located on the frozen layers are not read from the input data set. If set to **no**, then the features are read from the frozen layer.

Values: *YES* | *NO*

Default value: *NO*

Workbench Parameter: *<WorkbenchParameter>*

IGNORE_LOCKED_LAYERS

Required/Optional: *Optional*

Specifies whether the reader will ignore all features on the locked layers. If set to **yes**, then features located on the locked layers are not read from the input data set. If set to **no**, then the features are read from the locked layer.

Values: *YES* | *NO*

Default value: *NO*

Workbench Parameter: *<WorkbenchParameter>*

IGNORE_HIDDEN_LAYERS

Required/Optional: *Optional*

Specifies whether the reader will ignore all features on the hidden layers. If set to yes, then features located on the hidden layers are not read from the input dataset. If set to no, then the features are read from the hidden layer. If this option is specified at the time of workspace or mapping file generation, and the schema mode is by layer, then no schema information from hidden layers will be used to generate the workspace or mapping file. This is generally set to No when performing AutoCAD-to-AutoCAD translations.

Values: *YES* | *NO*

Default value: *YES*

Workbench Parameter: *<WorkbenchParameter>*

VISIBLE_ATTRIBUTES_AS_TEXT

Required/Optional: *Optional*

Specifies whether the reader should return visible attributes as separate text features or whether they should be returned as attributes of an insert feature. When this is **yes**, then each visible attribute is returned as a single text feature. This is generally set to **No** when performing AutoCAD-to-AutoCAD translations.

Values: *YES* | *NO*

Default value: *YES*

Workbench Parameter: *Read Visible Attributes as Text Entities*

EXTENDED_ENTITY_FORMAT

Required/Optional: *Optional*

Instructs the FME to use the specified manner when decoding the extended entity data. This directive disables the automatic parsing.

Values: *ALTERNATE_NAME_VALUE* | *CSV(<SEPARATOR>)*

where **<SEPARATOR>** specifies the character used to delimit the attribute name from the attribute value.

When **ALTERNATE_NAME_VALUE** is specified, then it is assumed that the values stored with each feature in the extended entity portion of the feature alternate between specifying the attribute name and attribute value.

When **CSV(<SEPARATOR>)** is specified, then an attribute name value pair is specified in each extended entity value. The values are separated by **<SEPARATOR>**.

Workbench Parameter: *<WorkbenchParameter>*

READ_PAPER_SPACE

Required/Optional: *Optional*

Instructs the FME to also read the entities from paper space. By default, the FME only reads the entities from model space.

Values: *YES* | *NO*

Default value: *NO*

Workbench Parameter: *Read Paper Space*

READ_GROUPS

Required/Optional: *Optional*

Determines whether or not AutoCAD groups will be read. By default, FME will not read groups.

Values: *YES* | *NO*

Default value: *NO*

Workbench Parameter: *Read Groups*

IGNORE_UCS

Required/Optional: *Optional*

Instructs the FME to ignore the user defined coordinate system of the file being read. By default, the FME applies the UCS when reading the coordinate data. This is generally set to **Yes** only when performing AutoCAD-to-AutoCAD translations.

Values: *YES | NO*

Default value: *NO*

Workbench Parameter: *Ignore UCS*

USE_DXF_HEADER

Required/Optional: *Optional*

When reading DXF files this instructs the FME Reader to use the specified dxf header file as the header for the file being read. This option is used to handle the case where organizations produce *headerless* dxf files to save storage space.

Values: *<DXF_HEADER_FILE>*

where *<DXF_HEADER_FILE>* specifies the full pathname to the dxf header file. If this is specified when reading a DWG file, then the translation is terminated with an error.

Workbench Parameter: *Use DXF Header*

SKIP_TO_SECTION

Required/Optional: *Optional*

When **USE_DXF_HEADERS** is specified above, this specifies how much of headerless file is to be skipped. In some cases, the headerless file has a placeholder which has to be removed before concatenating the above header file to the dataset. If not specified, then no lines are skipped.

Values: *one of the AutoCAD **SECTIONS** as specified in AutoCAD files*

PRESERVE_COMPLEX_HATCHES

Specifies whether or not to read hatches and mpolygons in a way that preserves their complex properties.

Required/Optional

Optional

Values:

YES | NO (default)

If the value is NO, then the loops of each hatch or mpolygon entity will be converted to areas and aggregated together. If the value is YES, then the loops will be aggregated together as polygons, ordered such that any enclosing loop will be aggregates before any enclosed loop.

This is generally set to YES when performing AutoCAD-to-AutoCAD translations.

Workbench Parameter

Preserve Complex Hatches and MPolygons

READ_AS_2_5D

Required/Optional: *Optional*

Determines whether polylines should have their elevation attribute treated as a Z coordinate; when this occurs the **autocad_elevation** attribute will not be present. Applies to light-weight polylines and 2D polylines.

Note that this option should not be set when doing AutoCAD-to-AutoCAD translations as the elevations converted to Z coordinates when read in will not be converted back to elevation attributes when written out.

Values: *YES* | *NO*

Default value: *NO*

Workbench Parameter: *Read Polylines as 2.5D*

RESOLVE_ENTITY_COLOR

Required/Optional: *Optional*

Specifies whether or not to resolve the color of the entity to the color of the layer for that entity or block of that entity. This resolution only affects entity's who already have a color that is set to **COLOR_BYLAYER** which is indicated by a color of **ByLayer (index 256)**, or **COLOR_BYBLOCK** which is indicated by a color of **ByBlock (index 0)**.

If the value of this directive is **yes**, and the entity has a color of **COLOR_BYLAYER**, then the **autocad_original_color** attribute is set to **ByLayer**, and the **autocad_color** attribute is set to the color index for the layer that the entity is on.

Similarly, If the value of this directive is **yes**, and the entity has a color of **COLOR_BYBLOCK**, then the **autocad_original_color** attribute is set to **ByBlock**, and the **autocad_color** attribute is set to the color index for the block that the entity is in.

If the value of this directive is **no**, then **both the autocad_color** attribute and the **autocad_original_color** attribute will remain unresolved as a value of **ByLayer** or **ByBlock**. This is generally set to **No** when performing AutoCAD-to-AutoCAD translations.

Values: *YES* | *NO*

Default value: *YES*

Workbench Parameter: *Resolve Entity Color*

APPLY_WORLD_FILE

Required/Optional: *Optional*

Use this directive when you have an ESRI World file (*.wld) that you want FME to use when determining the coordinates for features in your dataset. When this directive has a value of **YES** FME will search the directory of the dataset for a file with the same name as your dataset but with a .wld extension. If it cannot find a file with that name it will then look for the file "esri_cad.wld" within the dataset directory. If either of those files exist then FME will use the information in the files to translate the coordinates of the features in the dataset to their new geospatial coordinates. If the files cannot be found then the translation will continue, using the coordinate information found in the dataset, without performing any additional transformation.

Values: *YES* | *NO*

Default Value: *NO*

Default Workbench Value: *YES*

Workbench Parameter: *Apply World File*

EXPLODE_MTEXT

Required/Optional: *Optional*

Specifies whether the reader will explode the mtext entities into separate text entities. When exploding, the resulting text features represent fragments of text with the same mtext properties such as style and location. When not exploding, the mtext entity will be read as a single text feature. This is generally set to **No** when performing AutoCAD-to-AutoCAD translations.

Values: *YES* | *NO*

Default value: *NO*

Workbench Parameter: *Explode MText*

STORE_LAYER_INFO (This is not applicable to the RealDWG reader)

Required/Optional: *Optional*

When specified, the AutoCAD Reader will add additional attributes describing the layer properties for the layer of each feature. These include `autocad_layer_linetype`, `autocad_layer_color`, and `autocad_layer_lineweight`.

Values: *YES | NO*

Default value: *NO*

Workbench Parameter: *Store Layer Properties on Features*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

`<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>`

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

*** Workbench Parameter**

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The `COORDINATE_SYSTEM` directive, which specifies the coordinate system associated with the data to be read, must always be set if the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` to the reader `COORDINATE_SYSTEM` prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

`<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>`

*** Workbench Parameter**

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

✳ Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

✳ Workbench Parameter

Additional Attributes to Expose

Tips for AutoCAD Reading

Tip: The AutoCAD reader automatically determines whether the file is DWG or DXF and processes it accordingly. Therefore, the same mapping file can be used to read either DXF or DWG.

Note that the AutoCAD reader directives list defaults that will produce the best generic result to any non-AutoCAD destination format, resulting in the best presentation at the cost of some of the original AutoCAD types being lost. For example, blocks and dimensions will be resolved into their component parts. If an AutoCAD-to-AutoCAD translation is desired, the following set of options should be used for best preservation of the original drawing.

```
STORE_BULGE_INFO = YES  
STORE_SPLINE_DEFS = YES  
RESOLVE_BLOCKS = NO  
RESOLVE_DIMENSIONS = NO  
VISIBLE_ATTRIBUTES_AS_TEXT = NO  
IGNORE_UCS = YES  
PRESERVE_COMPLEX_HATCHES = YES  
IGNORE_HIDDEN_LAYERS = NO  
RESOLVE_ENTIY_COLOR = NO
```

These are also noted on each individual directive description.

Writer Overview

The AutoCAD writer provides the following capabilities when writing AutoCAD files.

- **User-defined Linetypes:** New linetypes can be defined on FME mapping file lines. These linetypes can then be referenced by features being written to the AutoCAD file.
- **User-defined Layers:** Users must define the layers into which features are stored. The layers can also define the attributes to be stored within the feature.
- **Copy Block Definitions:** Often users have existing AutoCAD drawing files that contain block definitions they want the translated data to carry. Specifying the **TEMPLATEFILE** keyword in the mapping file results in block definitions being copied from the existing file to the output DWG/DXF file. These blocks can then be referred to by insert entities.
- **Copy Linetypes:** Predefined linetypes within existing DWG/DXF files are copied making them available for use by features being written to the destination file. Specifying the **TEMPLATEFILE** keyword in the mapping file results in the predefined linetypes being copied from the template file to the output drawing file. Feature entities can then refer to these linetype definitions.
- **Copy Layer Definitions:** Layer definitions within an existing DWG/DXF file identified by **TEMPLATEFILE** enable layer definitions to be copied to the destination data set and then referenced.
- **Copy Shape Header Definitions:** Shape header definitions are also copied from the file specified by the **TEMPLATEFILE** directive.
- **Automatic Block Creation:** When a feature is passed to the writer that cannot be written as a single AutoCAD entity, such as a donut polygon, the writer automatically defines an AutoCAD block and inserts entities necessary to represent the feature. If a block is already defined with that name, either through previous block creation or through existence in the template file, then the existing block definition will be used and the multi-part feature will be added at an insert point calculated from the feature geometry. If the `autocad_block_insert_[xyz]` attributes are specified, they will be used to specify an insert point for the new block reference.
- **Flexible Attribute Support:** Attribute information is, by default, written to extended entity data for each feature written to the data set. This can be overridden, however, through the use of the `autocad_attributes` attribute being set as shown in the following table.
- **Multi-version Support:** Currently the AutoCAD DWG/DXF writer supports files that are compatible with any current AutoCAD release.

autocad_attributes value	Description
<code>extended_entity_data</code>	This results in the attribution being written to the extended entity for the feature.
<code>insert_attributes</code>	This results in the writer creating an insert entity for each feature and storing all attributes with the insert entity. The insert entity refers to a block that contains the geometry of the output feature.
<code>external_attributes</code>	This is the default value. There are no attributes written to the AutoCAD file. This is useful if the attributes are being stored in an external database or if attribute information is not wanted.

When creating AutoCAD files, the AutoCAD writer first defines the linetypes and layers defined within the FME mapping file. The writer then reads in a template file, if specified, and copies the linetypes, layer definitions, shape file header information, and block information from the template file to the output dataset.

The AutoCAD writer then outputs each feature it is given to the output file in the appropriate entity type.

When writing an AutoCAD file, the format of file output is determined as follows:

- If the file name contains **.dwg** or **.DWG**, then the output data set is written in the ACAD format.
- Otherwise, if the file name contains **.dxf** or **.DXF**, then the output data set is written in DXF format.
- Otherwise, if an error exists in the mapping file, the translation is halted.

The AutoCAD writer uses the above rules to enable the same FME mapping file to be used to create both DXF and DWG output files. Users are able to specify their choice simply by changing the suffix of the output file being produced.

Writer Directives

This section describes the directives the AutoCAD writer module recognizes. Each of the directives is prefixed by the current **<writerKeyword>**_ when they are placed in a mapping file. By default, the **<writerKeyword>** for the AutoCAD writer is the same as the **<writerType>**. The following directives are used by all AutoCAD.

DATASET

Required/Optional: *Required*

The dataset into which feature data is to be written.

Workbench Parameter: *Destination Autodesk AutoCAD DWG/DXF File*

VERSION

Required/Optional: *Required*

The version of AutoCAD file to be produced. The value corresponds with the release number of the AutoCAD file that is produced.

Values:

- *same_as_template* (not applicable to RealDWG) This option takes the version from the template file.
- *Release9* and *Release10* (Support for these versions has been deprecated)
- *Release12* (not applicable to RealDWG)
- *Release13* (not applicable to RealDWG)
- *Release14*
- *Release2000*
- *Release2004*
- *Release2007*
- *Release2010*

Default value: *Release2007*

Workbench Parameter: *AutoCAD Version*

Example:

The example statement below instructs the AutoCAD writer to produce a release 12 AutoCAD file:

```
ACAD_VERSION Release12
```

TEMPLATEFILE

Required/Optional: *Optional*

This statement specifies the name of the existing AutoCAD DXF or DWG file that contains linetype, layer, shape header, block definitions and a codepage to be copied to the destination AutoCAD file. Some AutoCAD users also refer to this as a *prototype file*. This is an optional parameter. If the parameter is not defined, then the output file uses the linetype defined in the mapping file along with the predefined type of **CONTINUOUS** which is always present in an AutoCAD drawing.

Tip:

- *LINETYPE definitions found in the mapping file override any linetype definitions found in the template file.*
- *The template file can also be used to set the codepage of the resulting AutoCAD file.*

The example below specifies that the file called **c:\tmp\test.dwg** contains the block, layer, shape header definitions, and linetype definitions for the output data set.

```
ACAD_TEMPLATEFILE c:/tmp/test.dwg
```

Tip: Many AutoCAD users refer to the template files as prototype files.

Workbench Parameter: *Template File*

AUTO_CREATE_LAYERS

Required/Optional: *Optional*

This statement tells the writer to create layers as needed. Normally, all layers must either be defined by **_DEF** lines or by the template file before they can be used. If **AUTO_CREATE_LAYERS** is specified as **YES**, then when a feature is sent to the writer with a feature type that has not previously been defined as a layer, a new layer will be created with the properties of the last **_DEF** line found in the mapping file.

This example sets the writer into a mode where it creates layers as needed. Each created layer has a color of **10** and a linetype of **CONTINUOUS**.

```
ACAD_AUTO_CREATE_LAYERS yes
ACAD_DEF DEFAULT
autocad_color 10
autocad_linetype CONTINUOUS
```

Values: *YES | NO*

Default value: *NO*

Workbench Parameter: *<WorkbenchParameter>*

OUTPUT_DEFINED_ATTRS_ONLY

Required/Optional: *Optional*

When this directive is set to **yes**, then only those attributes defined as part of the layer definition will be stored (see the DEF directive for more details).

Values: *YES | NO*

Default value: *YES*

Workbench Parameter: *Output Defined Attributes Only*

USE_ATTRDEFS_FOR_INSERTS

Specifies whether the writer should use the attribute definitions that are found within blocks when placing inserts. If **no**, then all the attributes on a feature that is passed to the writer are written as insert attributes. If **yes**, then only the attributes defined within the block being placed are stored as insert attributes.

Values: *YES | NO*

Default value: *YES*

Workbench Parameter: <WorkbenchParameter>

STRIP_HEADER_TO_SECTION

Required/Optional: *Optional*

This directive is only valid when going out to DXF and tells the FME to remove the header up to the start of the specified **SECTION**. The name of the **SECTION** can be any valid autocad section. If not specified then the file is output as before.

NEW_HEADER_CONTENT_FILE

Required/Optional: *Optional*

DEPRECATED. Please use the TEMPLATEFILE directive instead.

This directive is only valid with the **STRIP_HEADER_TO_SECTION** directive above and is the name of the file that contains the new header information. The contents of this file are placed at the start of the output file replacing the contents removed by **STRIP_HEADER_TO_SECTION**.

DEFAULT_ATTR_STORAGE

Required/Optional: *Optional*

This directive specifies the default manner in which attribute data will be stored. If not specified, then the default value is **extended_entity_data**. This directive changes the default value for the **autocad_attributes** feature based directive. If all attributes are to be stored in a single manner, then this directive is the easiest manner in which to do this.

Values:

- *extended_entity_data* instructs the writer to store all attribution in extended entity data as the default
- *insert_attributes* instructs the writer to store all attribution using inserts
- *external_attributes* instructs the writer to not store any attribution in the AutoCAD file

Workbench Parameter: *Attribute Output*

SUPPRESS_FONT_WARNINGS

Required/Optional: *Optional*

Specifies whether to suppress warnings about unknown font metrics being encountered.

Values: *YES* | *NO*

Default value: *NO*

Workbench Parameter: <WorkbenchParameter>

FONT_DIRECTORY

Required/Optional: *Optional*

Specifies the directory in which all specified fonts are located. When specified, FME can calculate the font metrics, enabling it to place fonts more accurately. If not specified, then FME assumes that the full path is specified on the correlation lines via the attribute **autocad_shape_filename**. If this attribute is not set or it is not the full path of the font file, then FME will use the default font file called **default.shx**.

Values:

Workbench Parameter: <WorkbenchParameter>

SHAPE_DIRECTORY

Required/Optional: *Optional*

Specifies the directory in which to look for shapes files (*.shp) from which information about SHAPE entities will be extracted for writing.

Values: *<shape file directory path>*

Workbench Parameter: *AutoCAD Shape File Directory*

DEFAULT_APPLICATION

Required/Optional: *Optional*

The application name that is used when writing extended entity data.

Default values: *ACAD*

Workbench Parameter: *Default Application Name*

LINETYPE

Required/Optional: *Optional*

The AutoCAD writer enables linetypes to be defined within the FME mapping file. This enables the user to control how output lines are to look in the destination data set. The linetype definition is of the following form:

```
<writerkeyword>_LINETYPE <linetype name>      \  
autocad_textpict <picture>                    \  
[autocad_patternLength <pattern Length>      \  
  <segment values>+                            \  
]
```

where:

- **<linetype name>** is the name used throughout the mapping file to refer to the linetype being defined by this statement.
- **<picture>** is the text or name displayed in AutoCAD when linetypes are displayed.
- **<pattern Length>** is the length of a single instance of the line.
- **<segment values>** are the length of each of the segments within the linetype segment. The segment values obey the following rules:
 - **negative** value – pen up length (used to create spaces of varying lengths)
 - **positive** value – pen down length (used to make dashes of varying lengths)
 - **zero** – used to create a dot

The following example creates a linetype called dash-dot that appears as “_ _ . _ _ . _ _ . ” and so on when displayed on the screen.

```
ACAD_LINETYPE dash-dot      \  
  autocad_textpict DASHDOT  \  
  autocad_patternLength 1.0 \  
  0.5 -0.25 0 -0.25
```

CREATED_BLOCK_NAME_PREFIX

Required/Optional: *Optional*

The names of blocks created by the writer during automatic block creation will use the value of this keyword as a prefix if specified. If not specified the syntax will be:

```
<filename>_<writerkeyword>_FME_BLOCK_<blocknumber>
```

Workbench Parameter: *Created Block Name Prefix*

DEF

Required/Optional: *Optional*

The AutoCAD writer requires that every feature written to the AutoCAD file be stored within a predefined AutoCAD layer. In AutoCAD, the layers are used to store collections of logically related attributes. Within the FME, the AutoCAD layer and the type of the feature are treated synonymously as there is a one-to-one correspondence between FME feature type and AutoCAD layer.¹The order of properties in the layer statement is required as shown, though additional attribute name and type pairs may be in any order. The layer statement is of the following form:

```
<writerkeyword>_DEF <layer name>      \
      autocad_color <default color>    \
      autocad_linetype <default linetype> \
      [autocad_layer_type frozen]      \
      [<attribute name> <attribute type>]
```

where:

- **<layer name>** is the name of the layer being defined. This is the name which is used throughout the remainder of the FME mapping files. Layer name cannot be empty. If no layer name is specified, then FME will provide a fixed name “_FME_NO_LAYER_NAME_” for such layers.
- **<default color>** is the color number used for all features stored within the layer unless explicitly overridden on the correlation lines below. Valid values are between 1 and 255.
- **<default linetype>** is the name of the linetype to use for the layer if no linetype is specified on the correlation line. The linetype specified must either be:
 - defined in the mapping file,
 - copied from a specified template file, or
 - the predefined linetype named **CONTINUOUS**.
- **<autocad_layer_type>** is the type of layer to create. Currently, only the value **frozen** is supported. If specified, then the created layer is frozen; otherwise, the layer is not frozen.
- **<attribute name> <attribute type>** is the definition of an attribute to be stored within the extended entity data of features for the layer. If no attributes are defined, then all feature attributes (except those that start with **autocad_**) are stored. The storing of attributes can be turned off by specifying a value of **external_attributes** for the **autocad_attributes** feature attribute on the correlation line. The values for **<attribute type>** are the same as those for ESRI Shapefiles.

The example below defines a layer called **boundary** in which entities are drawn using color 13 (unless otherwise specified) and a linetype called dash-dot (unless otherwise specified). The feature also has several attributes specified that will be written to the extended entity data of each feature within the layer.

```
ACAD_DEF boundary      \
      autocad_color 13  \
      autocad_linetype dash-dot \
      FEATCODE char(12) \
      PPID char(10) \
      DATECHNG date \
      SURVEYDIST number(8,2)
```

AUDIT_AND_FIX (not supported in RealDWG writer)

Required/Optional: *Optional*

This directive can be used to turn on/off internal auditing before the final drawing file is written out. By default it is set to **YES**, so auditing will be performed and any errors found will be fixed. It is recommended that you leave the auditing set to **YES**. If you set it to **NO**, it is possible that the output file may not be as per the AutoCAD file specification. As an example of how auditing fixes errors is that if there is a layer name with a space or any other invalid characters, then that layer name will be changed to something like **\$DDT_AUDIT_GENERATED_(3B)**. If auditing

¹Layers can also be defined through the use of a **TEMPLATEFILE**.

is turned off, then the layer name will not be changed and when it is audited in AutoCAD, it will return errors such as invalid layer names.

Values: *YES* | *NO*

Default value: *YES*

Workbench Parameter: *Audit and Fix Errors*

COORDINATE_SYSTEM_STORAGE (not applicable to RealDWG writer)

Required/Optional: *Optional*

This directive controls whether the writer will optionally store the coordinate system of its features. The coordinate system can be stored inside the output AutoCAD file as an ESRI Well Known Text (in an ESRI_PRJ entry in an internal dictionary in the file), according to ESRI specifications. It can also be stored externally in a companion ESRI .prj file that shares the output AutoCAD file's base name, but has a .prj extension.

Values: *NONE* | *EXTERNAL_PRJ* | *EXTERNAL_AND_INTERNAL* | *INTERNAL_WKT*

Default: *NONE*, which means projection information is not stored anywhere.

AutoCAD data files written this way with projection information will be recognized by FME and the free ArcGIS for AutoCAD application, which installs on top of the AutoCAD application.

Workbench Parameter: *Coordinate System Storage*

Example:

```
ACAD_COORDINATE_SYSTEM_STORAGE EXTERNAL_PRJ
```

APPEND_TO_TEMPLATEFILE

Required/Optional: *Optional*

This directive can be used to allow the file specified by the DATASET directive to be written as the concatenation of the full contents of the file specified by the TEMPLATEFILE directive with all written data. By default it is set to **NO**, so only header information but no data is used from the template file if one is specified. If this directive is set to **YES** then the full header and data information is used from the template file.

Values: *YES* | *NO*

Default value: *NO*

Workbench Parameter: *Append Data to Template File*

USE_BLOCK_NAME_FOR_CREATION

Required/Optional: *Optional*

The names of blocks created by the writer during automatic block creation will use the value of the autocad_block_name attribute if the attribute is present and this directive is specified.

If a template file is used during writing, the block name will be used to try to match an existing block definition in the template file. This is similar to the behavior for insert entity writing. This directive does not affect the use of the block name for insert entity writing.

If the intention is that only insert entities should be used to try to match block names with block definitions, set this directive to *NO*.

If not specified, the syntax may be the following, but may also be modified by the usage of the CREATED_BLOCK_NAME_PREFIX directive:

```
<filename>_<writerkeyword>_FME_BLOCK_<blocknumber>
```

Values: *YES* | *NO*

Default Value: *NO*

Workbench Parameter: Use the block name to create blocks

Feature Representation

Special FME feature attributes are used to hold AutoCAD entity attributes. The AutoCAD writer uses these attribute values as it fills in an entity structure during output. The AutoCAD reader sets these attributes in the FME feature it creates for each entity it reads.

The FME considers the AutoCAD layer¹ to be the *FME feature type* of an AutoCAD feature. Each AutoCAD entity, regardless of its entity type, shares a number of other attributes, as described in the following table. Subsequent sections describe attributes specific to each of the supported entity types.

Attribute Name	Content
autocad_layer	The name of the feature's layer. This is the same value as the feature's type and is stored when reading for reasons of convenience. This value is ignored when entities are being written to a drawing file. Value: char(33) Default: No default
autocad_layer_color (not used in RealDWG)	This is the color value for the layer of the entity. See autocad_color for more information. This is only set when the STORE_LAYER_INFO reader directive is set to yes. Range: 0...256 Default: 256
autocad_layer_linetype (not used in RealDWG)	This is the linetype value for the layer of the entity. See autocad_linetype for more information. This is only set when the STORE_LAYER_INFO reader directive is set to yes. Range: char[33] Default: BYLAYER
autocad_layer_lineweight (not used in RealDWG)	This is the lineweight value for the layer of the entity. See autocad_lineweight for more information. This is only set when the STORE_LAYER_INFO reader directive is set to yes. Range: 0, 5, 9, 13, 15, 18, 20, 25, 30, 35, 40, 50, 53, 60, 70, 80, 90, 100, 106, 120, 140, 158, 200, 211, -1 (by layer), -2 (by block), -3 (default) Default: -3 (Default)
autocad_layer_type (not used in RealDWG)	This is used by the Reader only and indicates whether or not the feature comes from a frozen

¹The feature layer name corresponds to be the feature type and autocad_layer when reading. This enables the layer name to be extracted without the need to use the @FeatureType function.

Attribute Name	Content
	<p>layer.</p> <p>Range: frozen not_frozen</p>
<p>autocad_layer_frozen (only in RealDWG)</p>	<p>This is used by the Reader only and indicates whether or not the feature comes from a frozen layer.</p> <p>Range: yes no</p> <p>Default: no</p>
<p>autocad_layer_locked (only in RealDWG)</p>	<p>This is used by the Reader only and indicates whether or not the feature comes from a locked layer.</p> <p>Range: yes no</p> <p>Default: no</p>
<p>autocad_layer_hidden (only in RealDWG)</p>	<p>This is used by the Reader only and indicates whether or not the feature comes from a hidden layer.</p> <p>Range: yes no</p> <p>Default: yes</p>
<p>autocad_color</p>	<p>The color number of the entity. If the value is 0, then the color of the entity is that of the enclosing block; if the value is 256, then the color of the entity is that specified by the entity's layer; otherwise, the number specified determines the color of the entity. If autocad_color is not specified, then the value will be set from fme_color. If fme_color is also not specified, then it will be set to COLOR_BYLAYER.</p> <p>Range: 0...256</p> <p>Default: 256</p>
<p>autocad_original_color</p>	<p>The color of the entity before it may be resolved to a specific color index. If the value is COLOR_BYBLOCK (index 0) or COLOR_BYLAYER (index 256), then the value of this attribute is ByBlock or ByLayer respectively. Otherwise, the number specified determines the color of the entity, like the value of autocad_color. See autocad_color for more information.</p> <p>Range: 0...256</p> <p>Default: 256</p>
<p>autocad_true_color</p>	<p>The true color Red Green Blue (RGB) values of the entity. This attribute is conditionally set on read in addition to the autocad_color attribute. This attribute is used in preference to the autocad_color attribute on write to set the color of an entity. If not present, see the autocad_color attribute.</p>

Attribute Name	Content
	Range: 0...255,0...255,0...255 Default: No default
autocad_entity_handle	The hexadecimal unique identifier for the entity. This value is unique within each AutoCAD file. Range: Hexadecimal identifier. Default: No default
autocad_entity_visibility	This is used by the Reader only and indicates whether or not the feature is visible. Range: visible invisible.
autocad_linetype	The name of the feature's linetype. This can be a specific linetype value or it may be set to BYLAYER, indicating that the linetype will be set to the linetype value of the layer. See autocad_resolved_linetype for more information. Range: char[33] Default: BYLAYER
autocad_linetype_scale	The amount to scale the feature's linetype by for viewing in AutoCAD. Failure to set appropriate values for linetype may result in viewing errors such as dashed lines appearing solid. Range: 64 bit Real Default: 1.0
autocad_lineweight	The lineweight of the AutoCAD entity in 100ths of a millimeter. To set a lineweight of 0.05 mm in AutoCAD, set the attribute value to 5. Range: 0, 5, 9, 13, 15, 18, 20, 25, 30, 35, 40, 50, 53, 60, 70, 80, 90, 100, 106, 120, 140, 158, 200, 211, -1 (by layer), -2 (by block), -3 (default) Default: -3 (Default)
autocad_resolved_linetype	This is used to store actual linetype value used for a feature. It will be the specific linetype value of the feature, or if the autocad_linetype has the value of BYLAYER, then this value will be the linetype of the layer. Range: char[33]
autocad_thickness	The thickness of the entity's lines. Range: 64 bit Real Default: 0
autocad_entity	The FME name for the type of entity this feature represents.

Attribute Name	Content
	<p>Range: See AutoCAD Entity Types and Descriptions Default: No default</p>
autocad_original_entity	<p>This attribute indicates that the entity is part of a block reference entity that has been resolved into its components. In general, if this attribute exists, its value will be <i>insert</i>.</p> <p>Range: See AutoCAD Entity Types and Descriptions Default: insert</p>
autocad_original_entity_type	<p>The FME name for the original type of entity this feature represents. For example, if the autocad_entity attribute is autocad_line, this attribute will indicate what type of line, i.e. line, 2dpolyline, lwpolyline or 3dpolyline.</p> <p>Range: See AutoCAD Entity Types and Descriptions Default: The value of the autocad_entity attribute</p>
autocad_original_position_x	<p>This indicates the original location in the x dimension for the first point of this entity. This attribute is set when the entity location is changed, such as for a component of a block reference when block references are resolved, and a block offset is applied.</p> <p>Range: 64 bit Real Default: No default</p>
autocad_original_position_y	<p>This indicates the original location in the y dimension for the first point of this entity. This attribute is set when the entity location is changed, such as for a component of a block reference when block references are resolved, and a block offset is applied.</p> <p>Range: 64 bit Real Default: No default</p>
autocad_original_position_z	<p>This indicates the original location in the z dimension for the first point of this entity. This attribute is set when the entity location is changed, such as for a component of a block reference when block references are resolved, and a block offset is applied.</p> <p>Range: 64 bit Real Default: No default</p>
autocad_space	<p>This is used by the Reader only and indicates if the entity being read came from paper space or model space.</p> <p>Range: model_space paper_space Default: No default</p>
autocad_attributes	<p>Used by the writer module only. This directs the</p>

Attribute Name	Content
	<p>writer on how the attributes for the feature are to be stored. If this attribute is not specified or is specified as <code>extended_entity_data</code> then the attribution associated with the feature is written to the extended entity portion. If the value is <code>insert_attributes</code>, insert entities are created for the attributes. If the value is <code>external_attributes</code> then the attribution is not written to extended entity data.</p> <p>Range: <code>extended_entity_data</code> <code>insert_attributes</code> <code>external_attributes</code></p> <p>Default: <code>external_attributes</code></p>

Extended Entity Data

Each entity in an AutoCAD file may have associated extended entity data. This data is typically used by applications to store attribute information. The AutoCAD reader attempts to make extended entity data as simple to use as possible by storing it in three different formats within the FME feature object. The first two formats merely store the data as found in the drawing file in the feature, while the third format attempts to present the attribute information in a more useful manner. It is important to remember that when extended entity data is read from an AutoCAD file, all three formats are stored within a single FME feature. The format that is actually used (if any) is dependent on the configuration of the remainder of the FME mapping file.

The AutoCAD writer understands both the *list format*, and the *interpreted format*, creating extended entity data from attribute data in the *list format* form, if they are present. In the absence data in the *list format* form, the writer will create extended entity data from attribute data in the *interpreted format*, when `autocad_attributes` is set to `extended_entity_data`. The size of extended entity data that can be stored on an single entity is limited to 16K bytes. The AutoCAD writer is limited to creating 8K bytes per entity. (Note that this is not applicable to ReadDWG.)

The *interpreted format* setting is described in [Interpreted Format](#), for extended entity data. When writing extended entity data, the FME features being output must structure their attributes in this way. That is, the attribute data is stored with each attribute being a single extended entity string in the form `<attribute name> = <attribute value>`. Storing the data in this manner enables the data to be easily viewed by AutoCAD and read by the FME reader module.

List Format

In this format, the data is simply stored in a list as found in the AutoCAD file. The data is stored in a single list named `extended_data_list`. Each value in the list is of the form `<attribute tag>: <attribute value>`. The `<attribute tag>`s supported by the FME are restricted to those given in the following table. The `<attribute tag>`s define the domain for the associated `<attribute value>`. Note that the AutoCAD codes associated with each kind of extended entity data are not stored in the FME feature.

Attribute Name	Content
application_name	<p>The name of the application which the following entity data is associated. This <code>application_name</code> remains in effect until another <code>application_name</code> entry is specified.</p> <p>AutoCAD Code: 1001</p> <p>Example: <code>application_name:ACAD</code></p>

Attribute Name	Content
autocad_layer	The name of the layer the extended data is associated. AutoCAD Code: 1003 Example: autocad_layer:Water
string	A character string value from 0 to 255 characters in length. AutoCAD Code: 1000 Example: string:Thompson
Binary data	A hexadecimal string from 0 to 254 characters in length. AutoCAD Code: 1004 Example: binary:E3B4
three_reals	Three 64-bit real numbers separated by commas. AutoCAD Code: 1010,1020,1030 Example: three_reals:2.3,4.5,3.4
world_position	Three real numbers which represent a world position. Each of the numbers is separated by a comma. AutoCAD Code: 1011, 1021, 1031 Example: world_position:23.4, -123.5, 0
world_displacement	Three real values which represent a world displacement value. Each of the values is separated by a comma. AutoCAD Code: 1012, 1022, 1032 Example: world_displacement:1.5, 2.3, 0
world_direction	Three real values which represent a world direction vector. Each of the values is separated by a comma. AutoCAD Code: 1013,1023,1033 Example: world_direction: 30.0, -12.4, 10
real	A 64-bit real number. AutoCAD Code: 1040 Example: real:3.1415926
distance	A 64-bit real number which represents a distance. AutoCAD Code: 1041 Example: distance:4.56
scale	A 64-bit real number which represents a scaling factor. AutoCAD Code: 1042 Example: scale:34.5
16Bit_integer	A 16-bit integer value. AutoCAD Code: 1070 Example: 16Bit_integer:245
32Bit_integer	A 32-bit integer value. AutoCAD Code: 1071

Attribute Name	Content
	Example: 32Bit_integer:12983

For example, if the following data was stored in extended entity data:

```

1001 C_NODE
    1000 CONNOBJ_1=43F4
    1000 COUNT=3
    1000 CONNOBJ_2=43F3
    1000 CONNOBJ_3=43F2
    1005 163
    1010 45.4
    1020 -123.5
    1030 0
    1001 DPRINT
    1000 postscript

```

then the FME AutoCAD reader would store this information as a list within the FME feature:

Attribute Name	Attribute Value
extended_data_list{0}	application_name:C_NODE
extended_data_list{1}	string:CONNOBJ_1=43F4
extended_data_list{2}	string:COUNT=3
extended_data_list{3}	string:CONNOBJ_2=43F3
extended_data_list{4}	string:CONNOBJ_3=43F2
extended_data_list{5}	handle:163
extended_data_list{6}	three_reals:45.4,-123.5,0
extended_data_list{7}	application_name:DPRINT
extended_data_list{8}	string:postscript

Notice how the AutoCAD codes are converted to attribute tags when stored in the FME features.

Structure Format

In this representation of extended entity data, the fields are stored with the tags forming part of the attribute names for each of the extended entity entries. The data is stored in a single structure in the FME feature named **extended_data**. As the extended entity data within AutoCAD is grouped into sections, with each section beginning with an application group code, the **extended_data** structure itself is also divided into different sections with each section beginning with **extended_data{#}**. The remainder of the attribute name consists of one of the parameters:

Extended Entity Parameter	Contents
application_name	The name of the application which the entity data is associated. AutoCAD Code: 1001

Extended Entity Parameter	Contents
autocad_layer{#}	The name of the layer the extended data is associated. AutoCAD Code: 1003
string{#}	A character string value from 0 to 255 characters in length. AutoCAD Code: 1000
three_reals{#}.real1 three_reals{#}.real2 three_reals{#}.real3	Three real numbers. AutoCAD Code: 1010,1020,1030
world_position{#}.x world_position{#}.y world_position{#}.z	Three values represent the x, y, and z components of a world_position value. AutoCAD Code: 1011, 1021, 1031
world_displacement{#}.x world_displacement{#}.y world_displacement{#}.z	Three values which represent a world displacement value. AutoCAD Code: 1012, 1022, 1032
world_direction{#}.x world_direction{#}.y world_direction{#}.z	Three real values which represent a world direction vector. AutoCAD Code: 1013,1023,1033
real{#}	A 64 bit real number. AutoCAD Code: 1040
handle{#}	AutoCAD handle value. AutoCAD Code: 1005
distance{#}	A 64 bit real number which represents a distance. AutoCAD Code: 1041
scale{#}	A 64 bit real number which represents a scaling factor. AutoCAD Code: 1042
16Bit_integer{#}	A 16 bit integer value. AutoCAD Code: 1070
32Bit_integer{#}	A 32 bit integer value. AutoCAD Code: 1071

For example, given the following extended entity data:

```

1001 C_NODE
    1000 CONNOBJ_1=43F4
    1000 COUNT=3
    1000 CONNOBJ_2=43F3
    1000 CONNOBJ_3=43F2
    1005 163
    1010 45.4

```

```

1020 -123.5
      1030 0
      1001 DPRINT
      1000 postscript

```

The information will be stored in the FME feature using structure notation as follows:

Attribute Name	Attribute Value
extended_data{0}.application_name	C_NODE
extended_data{0}.string{0}	CONNOBJ_1=43F4
extended_data{0}.string{1}	COUNT=3
extended_data{0}.string{2}	CONNOBJ_2=43F3
extended_data{0}.string{3}	CONNOBJ_3=43F2
extended_data{0}.three_reals{0}.real1	45.4
extended_data{0}.three_reals{0}.real2	-123.5
extended_data{0}.three_reals{0}.real3	0
extended_data{0}.handle{0}	163
extended_data{1}.application_name	DPRINT
extended_data{1}.string{0}	postscript

Notice how, in this case, the AutoCAD codes are used to form extensions for the attribute names. Also notice how the **extended_data** items are grouped in the FME feature as they are within the drawing file.

Interpreted Format

Finally, the FME AutoCAD reader module also attempts to interpret any *string* held in the extended entity data. If it is successful in interpreting any data, then it stores it as attributes within the feature. As it is reading each extended entity string entry, it attempts to determine if the value is composed of an attribute name or value pair and, if it does, it stores the information as such. For example, if the extended entity data from the previous example were read, the following interpreted values would be stored within the FME feature.

Attribute Name	Attribute Value
CONNOBJ_1	43F4
COUNT	3
CONNOBJ_2	43F3
CONNOBJ_3	43F2

The reader is able to do this by recognizing the = divider within each of the string attributes as the separator between an encoded attribute name and attribute value. The reader also recognizes a space character as a separator.

The remaining sections discuss the representation of each supported AutoCAD entity type.

Proxy Data

Some proxy data is also supported within the AutoCAD reader and writer. Proxy data is yet another manner in which data is stored within AutoCAD files. This data is normally associated with ARX extensions. To the rest of the FME, the

proxy objects are made to look as close as possible to regular AutoCAD data. For example, linear entity types are called **autocad_line**. Proxy features have a number of associated attributes that are not present in other entities.

Note: Only MPolygon proxy data is supported in AutoCAD R12 and older.

Proxy Data Attribute	Contents
autocad_proxy_number	A unique number that is assigned to all the components of a single object. Since a single proxy object can have a number of geometric primitives associated with it, all proxy objects are output with the same number so that, if necessary, they can be identified as belonging together by the rest of the FME processing.
autocad_class_number	The number given to the class of which this proxy object is an instance.
autocad_class_dxfname	The dxf class name of the proxy class.
autocad_class_cppname	The C++ class name of the proxy class.
autocad_class_appname	The application class name of the proxy class.
autocad_class_version	The class version of the proxy class.

Lines

autocad_entity: autocad_line

Features with **autocad_entity** set to **autocad_line** are stored in and read from drawing files in one of two ways, depending on the number of coordinates they have, and whether they store bulge arcs. Bulge arcs are limited to circular, non-closed arcs within the segments of the line. Any attempts to store elliptical or closed arcs will result in the arc being stroked into a line segment.

Number of Coordinates	AutoCAD Entity Type	Description
2	line	If the feature contained exactly two points, then an AutoCAD line entity is used to store the data.
Greater than 2	polyline	If the number of coordinates is greater than 2, then the AutoCAD polyline entity is used to store the coordinates. The polyline closed flag is set to indicate that the polyline entity is not closed.

Attribute Name	Content
autocad_bulge	Comma-separated value list of the vertex bulges. This is only useful when performing AutoCAD-to-AutoCAD translations, and is a measurement of the curvature at each vertex.
Applicable only with classic geometry.	
autocad_elevation	The elevation value stored with the line entity. This is often

Attribute Name	Content
	used to set the elevation for contour lines, as the single elevation value is applied to all the vertices.
autocad_linetype_generation	Whether the generation of the autocad linetype will restart at every vertex, or be generated continuously around the entire polyline. Not applicable for 3d polyline features. The possible values are: 0 = Restart generation at each vertex. 1 = Generate continuously around entire polyline.
autocad_width	The width of the line.
autocad_polyflag	A bit-coded flag. This attribute is only present or used on autocad_line features that contain more than two vertices. Values can be combined by using addition. The values are: 1 = The line is closed (or the line is a polygon mesh closed in the M direction). Please note that if this bit is set when reading, then the feature will be interpreted as an autocad_polygon . 128 = The linetype pattern is generated continuously around the vertices of the line.

XLines

autocad_entity: autocad_xline

Features with **autocad_entity** set to **autocad_xline** are stored in and read from drawing files as an FME feature with two coordinates representing a line. The reader and writer modules automatically convert the xline to and from its unit vector representation into a line.

There are no attributes specific to this type of entity.

Points

autocad_entity: autocad_point

Features with **autocad_entity** set to **autocad_point** are stored in and read from drawing files as a single coordinate feature.

Attribute Name	Content
autocad_ucs_xangle	The rotation angle around the z axis.

Ellipses

autocad_entity: autocad_ellipse

Ellipse features are point features used to represent both AutoCAD circle and AutoCAD ellipse entities. The point serves as the centre of the ellipse. Ellipse entities with an **autocad_primary_axis** equal to the **autocad_secondary_axis** are stored within the drawing file as a circle entity. Additional attributes specify the rotation, major axis, and minor axis of the ellipse.

Tip: The function @Arc() can be used to convert an ellipse to a polygon. This is useful for representing ellipses in systems that do not support them directly.

Attribute Name	Content
autocad_primary_axis	The length of the semi-major axis in ground units. Range: Any real number > 0 Default: No default
autocad_secondary_axis	The length of the semi-minor axis in ground units. Range: Any real number > 0 Default: No default
autocad_rotation	The rotation of the major axis. The rotation is measured in degrees counterclockwise up from horizontal. Range: -360.0..360.0 Default: 0

Polygons

autocad_entity: autocad_polygon

Features with **autocad_entity** set to **autocad_polygon** are stored in and read from drawing files as closed polyline entities.

Attribute Name	Content
autocad_width	The width of the line.

Splines

autocad_entity: autocad_splines

Spline features are linear or area features – depending on whether or not they are closed – and are used to represent features that have smooth curves. Each spline has a number of attributes that completely make up the spline. When **STORE_SPLINE_DEFS** is set to **yes**, the reader sets the coordinates to be either the fit points or the control points (depending on what is used to define the spline). Splines are always 3D – there is no way in AutoCAD to indicate if the feature was intended to be only 2D. If **STORE_SPLINE_DEFS** is not specified or set to **no**, then the coordinates of the spline returned by the reader are interpolated values based on the spline definition.

Tip: When you are performing an AutoCAD-to-AutoCAD translation, then you should always set STORE_SPLINE_DEFS to yes to get the best results.

AutoCAD splines have several attributes, which are returned when reading and must be specified when writing.

Attribute Name	Content
autocad_degree	The degree of the polynomial used to form the spline.
autocad_knot_tolerance	The tolerance of the spline knots.
autocad_degree	The degree of the spline.
autocad_cntl_pt_tolerance	The tolerance of the control points.
autocad_fit_tolerance	The tolerance of fit points.
autocad_knot_tolerance	The tolerance of knots.

Attribute Name	Content
autocad_num_cntl_pts	The number of control points.
autocad_num_fit_pts	The number of fit points.
autocad_knots	The number of knots.
autocad_flag	The flag that indicates the type of spline. It is a bit vector normally only used when going from AutoCAD to AutoCAD. 1. CLOSED 2. PERIODIC 4. RATIONAL 8. PLANAR 16. LINEAR
autocad_start_tangent_x autocad_start_tangent_y autocad_start_tangent_z	The start tangent for the spline.
autocad_end_tangent_x autocad_end_tangent_y autocad_end_tangent_z	The end tangent for the spline.
autocad_control_x autocad_control_y autocad_control_z	A comma separated list. The control point coordinates in comma separated values. If STORE_SPLINE_DEFS is specified, then the control points are also stored as the coordinates.
autocad_control_weights	The control point weights. A comma-separated list of the weight values for each control vertex.

When writing to splines, the spline must be specified exactly as it is returned from the reader with **STORE_SPLINE_DEFS** set to yes:

1. If the spline is defined by fit points then `autocad_num_cntl_pts` must be zero and `autocad_num_fit_pts` must be the same as the number of coordinates in the feature. The coordinates of the feature are taken to be the fit points.
2. If the spline is defined by control points then the `autocad_num_fit_pts` must be zero and `autocad_num_cntl_pts` must be the same as the number of coordinates in the feature. The coordinates of the feature are taken to be the control points.

Shapes

autocad_entity: autocad_shape

Features with **autocad_entity** set to **autocad_shape** are point features that identify where to place an AutoCAD shape object. The reader and writer modules process all attributes needed to fully specify the shape object reference. Depending on the output file (**.dxf** or **.dwg**), different information will be required to write shape entity. If a template file is specified using the **TEMPLATEFILE** keyword, then information about shape entity is extracted from the template file, which requires access to the shape file.

When writing to a DXF file, a shape name and a shape file name is all that is required. The presence of a shape file during translation is not required since there is no information lookup.

When writing to a DWG file and a shape name is given, then a lookup is performed to determine the shape index or shape number from the shape file, which is what DWG stores. For the lookup to be successful, the writer needs access to the shape file, and the specified shape name should be in the shape file. Similarly when writing to DWG, a shape number and shape file is all that is required. If there is no shape index or shape number, then the writer has to perform a lookup from the shape name to the shape index, and for this, access to the shape file is required.

When specifying a shape file, either the full path or just the filename can be specified. If only the shape filename is given, the writer will first look for that shape file in the directory specified by **SHAPE_DIRECTORY** keyword, and if not found then it will look in the directory where the output dataset is being written.

Tip: When an AutoCAD file is output, any shape files it references must be shipped together with the file.

Attribute Name	Contents
autocad_scale	The scale of the shape object for this point. Range: Any real number. Default: 1
autocad_shape_index	This identifies the index of the particular shape within the shape file. A single shape file may contain many different shapes. Range: Any real number > 0 Default: No default
autocad_rotation	The rotation of the shape for this entity. Range: -360.0..360.0 Default: 0
autocad_width_factor	The width factor for the shape. Range: Any real number > 0 Default: 0
autocad_oblique	The oblique angle of the shape. Range: -85.0 ..85.0 Default: 0
autocad_big_fontname	The name of the file which contains fonts for large character sets. Range: char[65] Default: NULL
autocad_shape_name	The name of the shape which is being read or written. Range: char[33] Default: No default
autocad_shape_filename	The name of the file in which the shape is defined. Range: char[65]; Default: No default
autocad_shape_rotation	The rotation of the shape definition relative to the shape file specification. Range: Any real number

Attribute Name	Contents
	Default: 0
autocad_shape_height	The height of the shape. Range: Any real number Default: 0
autocad_shape_width	The width of the shape. Range: Any real number Default: 1

Leaders

autocad_entity: autocad_leader

Features with **autocad_entity** set to **autocad_leader** are linear features that identify where to place an AutoCAD leader entity. The reader module returns the following leader-specific attributes. This is currently not supported by the writer.

Attribute Name	Contents
autocad_path_type	The type of path the leader follows. The path for a leader is one of autocad_straight_leader in which case the leader is a straight line, or autocad_spline_leader in which case the leader is a spline.
autocad_arrow_head_on	This specifies if the leader line has an arrowhead on it. Range: True False Default: True
autocad_hook_line_on_xdir	This is True if the hook line is in the same direction as the x direction and False if it is not. Range: True False Default: True
autocad_has_hook_line	This is True if the leader has a hook line, and False if it does not. Range: True False Default: True
autocad_anno_type	The type of annotation of the leader. Range: autocad_anno_text – annotation is mtext entity, autocad_anno_tolerance – annotation is a tolerance entity, autocad_anno_block – annotation is a block entity, and autocad_anno_none – no annotation with leader. Default: autocad_anno_none

Attribute Name	Contents
autocad_anno_height	The height of the associated mtext entity. Range: Real64
autocad_anno_width	The width of the associated mtext entity. Range: Real64
autocad_txt_offset_x	The offset of the last leader vertex from the annotation placement point. Range: Real64
autocad_txt_offset_y	The offset of the last leader vertex from the annotation placement point. Range: Real64
autocad_txt_offset_z	The offset of the last leader vertex from the annotation placement point. Range: Real64
autocad_x_dir_x	The x component of a vector indicating the horizontal direction of the text. Range: Any real number
autocad_x_dir_y	The y component of a vector indicating the horizontal direction of the text. Range: Any real number
autocad_x_dir_z	The z component of a vector indicating the horizontal direction of the text. Range: Any real number
autocad_offset_blkinspt_x	The x component of the offset of the last leader vertex from the block reference insertion point. Range: Any real number
autocad_offset_blkinspt_y	The y component of the offset of the last leader vertex from the block reference insertion point. Range: Any real number
autocad_offset_blkinspt_z	The z component of the offset of the last leader vertex from the block reference insertion point. Range: Any real number

Faces

autocad_entity: autocad_face

Features with **autocad_entity** set to **autocad_face** are stored as AutoCAD face entities. Additional attributes are used to define the visibility of the edges of the Face entity. Within the FME, if the reader is not using enhanced geometry, faces are stored as four-sided (five vertex) polygons. If the reader is using enhanced geometry, faces will be represented as surfaces with **autocad_entity** set to **autocad_surface**. See surfaces below for details.

Face surfaces are one-sided: they are only visible from one view direction. A face is visible when its normal points toward the observer. If the vertices of the outer boundary of the face are observed to be in anti-clockwise order, then the normal of the face points toward the observer, implying that the face is visible.

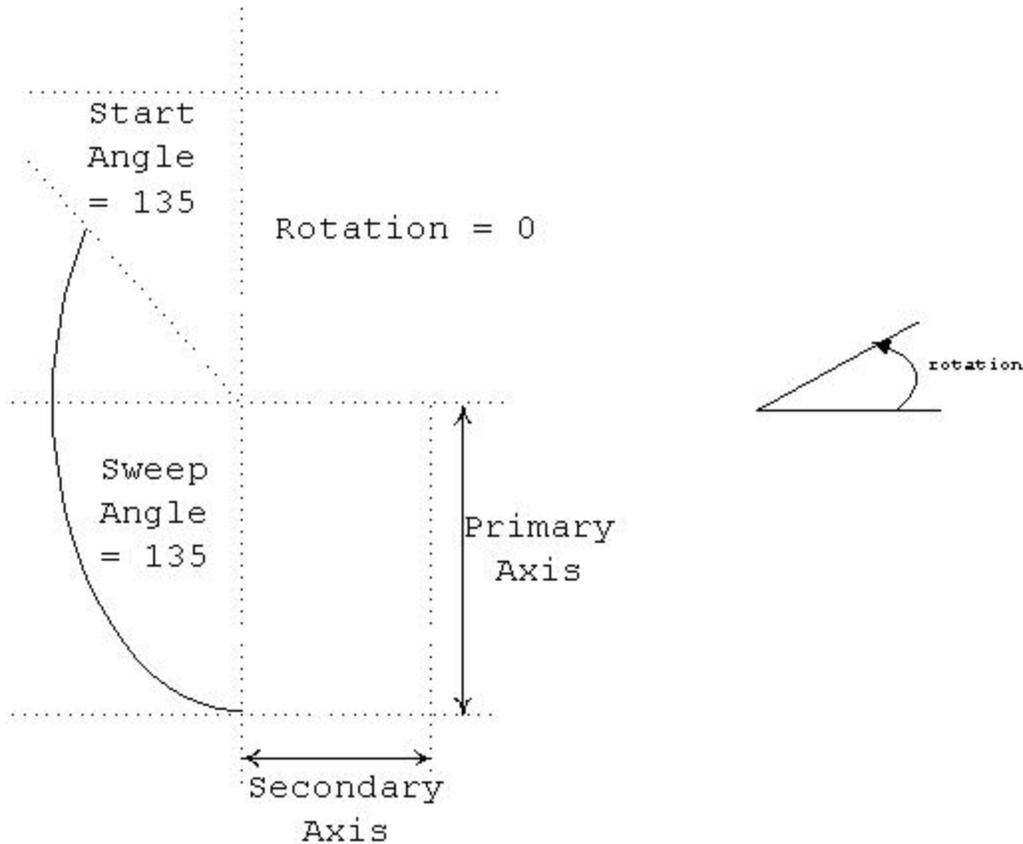
Attribute Name	Contents
autocad_edge_1	The visibility of the first edge of the Face. Range: visible invisible Default: visible
autocad_edge_2	The visibility of the second edge of the Face. Range: visible invisible Default: visible
autocad_edge_3	The visibility of the third edge of the Face. Range: visible invisible Default: visible
autocad_edge_4	The visibility of the final edge of the Face. Range: visible invisible Default: visible

Arcs

autocad_entity: autocad_arc

This geometry type is stored in an AutoCAD arc entity. Arc features are like ellipse features, except two additional angles control the portion of the ellipse boundary which is drawn. There are several properties of an FME arc geometry that may result in it being written as an AutoCAD entity other than an arc. If this arc geometry is circular and has a sweep angle of 360 degrees it will be stored in an AutoCAD circle entity instead of an arc entity. If this arc geometry is not circular, it will be stored in an AutoCAD ellipse entity.

Tip: The Function @Arc() can be used to convert an arc to a line. This is useful for representing arcs in systems that do not support them directly.



Attribute Name	Contents
autocad_primary_axis	<p>The length of the semi-major axis in ground units. Currently the value of the primary axis is always equal to the value of the secondary axis as AutoCAD arcs must be circular. When writing to an AutoCAD file, only the primary axis value is used.</p> <p>Range: Any real number > 0 Default: No default</p>
autocad_secondary_axis	<p>The length of the semi-minor axis in ground units. Currently the value of the primary axis is always equal to the value of the secondary axis as AutoCAD arcs must be circular. When writing to an AutoCAD file, only the primary axis value is used.</p> <p>Range: Any real number > 0 Default: No default</p>
autocad_start_angle	<p>Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of start_angle.</p> <p>Range: 0.0..360.0 Default: 0</p>

Attribute Name	Contents
autocad_sweep_angle	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of sweep_angle. Range: 0.0..360.0 Default: No default
autocad_rotation	The rotation of the ellipse that defines the arc. The rotation angle specifies the angle in degrees from the horizontal axis to the primary axis in a counter-clockwise direction. This value is fixed at 0 as AutoCAD doesn't support rotation of arcs at this time. Range: 0 Default: 0

Traces

autocad_entity: autocad_trace

Features with **autocad_entity** set to **autocad_trace** are stored in and read from drawing files as a 4-coordinate AutoCAD trace entity.

There are no attributes specific to this type of entity.

Solids

autocad_entity: autocad_solid

Features with **autocad_entity** set to **autocad_solid** are stored in and read from drawing files as a 3- or 4-coordinate AutoCAD solid entity. These represent 2D solids in comparison to 3D solids which are represented by **autocad_solid3d**.

There are no attributes specific to this type of entity.

Rays

autocad_entity: autocad_ray

Features with **autocad_entity** set to **autocad_ray** are stored in and read from drawing files as a two coordinate line. The reader and writer modules automatically convert the ray to and from its unit vector representation into a line.

There are no attributes specific to this type of entity.

Text Entities

autocad_entity: autocad_text

Features with **autocad_entity** set to **autocad_text** are stored in and read from drawing files as text entities. A text entity is represented by a single coordinate and the following attributes.

Attribute Name	Contents
autocad_text_string	The text string. Range: char[1024]

Attribute Name	Contents
	Default: No default
autocad_rotation	The rotation of the text for this entity. Range: -360.0..360.0 Default: 0
autocad_true_type_font	The name of the TrueType font used to display the text string. This attribute is only used by the AutoCAD Writer, since single-line text entities do not have support for TrueType fonts. If this attribute is specified on an <code>autocad_text</code> feature, a multiline text entity will be created and written instead of a single-line text entity. Default: No default
autocad_text_size	The text height. Range: Any real number > 0 Default: 10
autocad_width_factor	The scaling applied in the x direction which makes the text wider or narrower. However, this doesn't affect the weight (i.e., boldness) of the text string. Range: Any real number > 0 and <= 10000 Default: 1
autocad_oblique	The oblique angle of the text. Range: -85.0 ..85.0 Default: 0
autocad_alignment_x autocad_alignment_y autocad_alignment_z	The alignment coordinate of the text. This location is used in conjunction with the justification and the feature location to place the text correctly. This is primarily of use when performing an AutoCAD-to-AutoCAD translation. In order to override the justification of source AutoCAD dataset, you need to remove these alignment attributes from the feature and then set the <code>autocad_justification</code> attribute. This is because when performing an AutoCAD-to-AutoCAD translation, these alignment attributes override the <code>autocad_justification</code> attribute. Range: any 64-bit floating point value Default: x, y, and z value of text alignment point
autocad_big_fontname	The name of the file which contains fonts for large character sets.

Attribute Name	Contents
	Range: char[65] Default: NULL
autocad_shape_name	The name of the shape which contains the text font definition. Range: char[33] Default: STANDARD
autocad_shape_filename	The name of the file which contains the text font ^a definition. Range: char[65]; Default: txt
autocad_shape_rotation	The angle for the text as defined in shape file. Range: Any real number Default: 0
autocad_shape_height	The height of the text as defined in shape file. Range: Any real number Default: 0
autocad_shape_width	The width of the text as defined in shape file. Range: Any real number Default: 1
autocad_generation	The generation of the text entry. Range: autocad_normal autocad_upside_down autocad_backwards autocad_upsidedown_backwards Default: autocad_normal
autocad_justification	The justification of the text relative to its insert point. Range: autocad_top_left autocad_top_center autocad_top_right autocad_top_middle autocad_top_aligned autocad_top_fit autocad_middle_left autocad_middle_center autocad_middle_right

^aAutoCAD shape files should not be confused with ESRI Shapefiles. AutoCAD shape files hold font and symbol definitions; ESRI Shapefiles hold spatial features.

Attribute Name	Contents
	autocad_middle_middle autocad_middle_aligned autocad_middle_fit autocad_bottom_left autocad_bottom_center autocad_bottom_right autocad_bottom_middle autocad_bottom_aligned autocad_bottom_fit autocad_baseline_left autocad_baseline_center autocad_baseline_right autocad_baseline_middle autocad_baseline_aligned autocad_baseline_fit Default: autocad_baseline_left
autocad_tracking_percent	The tracking percent. Only used by the Reader. This attribute will only exist if reading a graphical text entity. Default: N/A since this is a Reader-only attribute.
autocad_backwards	Indicates whether the text is backwards. Only used by the Reader. This attribute will only exist if reading a graphical text entity. Default: N/A since this is a Reader-only attribute.
autocad_upside_down	Indicates if the text is upside down. Used only by the Reader. This attribute will only exist if reading a graphical text entity. Default: N/A since this is a Reader-only attribute.
autocad_vertical	Indicates if the text is vertical. Used only by the Reader. This attribute will only exist if reading a graphical text entity. Default: N/A since this is a Reader-only attribute.
autocad_underlined	Indicates if the text is underlined. Used only by the Reader. This attribute will only exist if reading a graphical text entity. Default: N/A since this is a Reader-only attribute.
autocad_overlined	Indicates if the text is overlined. Used only by the Reader. This attribute will only exist if reading a graphical text entity. Default: N/A since this is a Reader-only attribute.

Multi-Text Entities

autocad_entity: autocad_multi_text

Features with **autocad_entity** set to **autocad_multi_text** are stored in and read from drawing files as text entities. A text entity is represented by a single coordinate and the following attributes.

Attribute Name	Contents
autocad_text_string	The text string. Range: char[1024] Default: No default
autocad_rotation	The rotation of the text for this entity. Range: -360.0..360.0 Default: 0
autocad_text_size	The text height. When reading, this value is calculated using the height of the bounding box of the feature and the estimated number of lines. Range: Any real number Default: 10
autocad_mtext_string	The original formatted mtext string. Writer will use this attribute's value to set the destination mtext entity. When performing an AutoCAD-to-AutoCAD translation, this attribute will ensure that the exact formatting is carried over to the destination. Range: char[1024] Default: None
autocad_mtext_text_height	The starting text size of the multi-text feature. This attribute is useful mainly for AutoCAD-to-AutoCAD translations. If this attribute doesn't exist when writing, then a value is calculated for it. Range: Any real number Default: None
autocad_true_type_font	The name of the TrueType font used to display the text string. If this attribute is not specified when writing, the text will still be written, but not using a TrueType font. Default: No default
autocad_linespace_factor	The percentage of default line spacing used. Range: 0.25..4.0 Default: 1

Attribute Name	Contents
autocad_mtext_ref_rect_width	<p>The width of the reference rectangle in which the text is contained.</p> <p>Range: Any real number > 0</p>
autocad_attach_point	<p>The attach point for the multi-text. Use this attribute to set the justification of Multi-Text entities. Note that there is no <code>autocad_justification</code> attribute for this type of entity.</p> <p>Range:</p> <p><code>autocad_top_left </code> <code>autocad_top_center </code> <code>autocad_top_right </code> <code>autocad_middle_left </code> <code>autocad_middle_center </code> <code>autocad_middle_right </code> <code>autocad_bottom_left </code> <code>autocad_bottom_center </code> <code>autocad_bottom_right </code></p>
autocad_draw_direction	<p>The direction the text is drawn.</p> <p>Range:</p> <p><code>autocad_draw_left_to_right </code> <code>autocad_draw_right_to_left </code> <code>autocad_draw_top_to_bottom </code> <code>autocad_draw_bottom_to_top</code></p>
autocad_box_width	<p>The width of the box which the multi text is located.</p> <p>Range: Any real number > 0</p>
autocad_box_height	<p>The height of the box which the multi text is located.</p> <p>Range: Any real number > 0</p>
autocad_big_fontname	<p>The name of the file which contains fonts for large character sets.</p> <p>Range: char[65] Default: NULL</p>
autocad_shape_name	<p>The name of the shape which contains the text font definition.</p> <p>Range: char[33] Default: STANDARD</p>
autocad_shape_filename	<p>The name of the file which contains the text font^a</p>

^aAutoCAD shape files should not be confused with ESRI Shapefiles. AutoCAD shape files hold font and symbol definitions; ESRI Shapefiles hold spatial features.

Attribute Name	Contents
	definition. Range: char[65]; Default: txt
autocad_shape_rotation	The angle for the text as defined in shape file. Range: Any real number Default: 0
autocad_shape_height	The height of the text as defined in shape file. Range: Any real number Default: 0
autocad_shape_width	The width of the text as defined in shape file. Range: Any real number Default: 1
autocad_generation	The generation of the text entry. Range: autocad_normal autocad_upside_down autocad_backwards autocad_upsidedown_backwards Default: autocad_normal
autocad_estimated_num_lines (used by the reader only)	The estimated number of lines in the multi-text feature. This is a calculated value that uses the starting height of the multi-text feature, the height of the bounding box of the feature, and the linespacing factor. Range: Any integer Default: Not applicable since this is a reader-only attribute

Multi-Line

autocad_entity: autocad_multi_line

Multi-line features are linear features that represent a set of parallel lines. They are not capable of representing an arbitrary set of lines such as can be done with GIS systems. When reading a multi-line feature, the FME will output an aggregate of lines thereby hiding all AutoCAD format peculiarities. This entity is only supported by the AutoCAD Reader.

The following attributes are set when reading the multi-lines.

Attribute Name	Content
autocad_scale	The scale of the multi-line. Range: Any real number > 0.t
autocad_justification	The justification of the multi-line object is set to one of: Range:

Attribute Name	Content
	autocad_top autocad_middle autocad_bottom
autocad_num_verts	The number of vertices in one of the parallel multi-lines. All of the multi-lines have this number of coordinates. The feature thereby has <code>autocad_num_verts * autocad_num_lines</code> : Range: Number of vertices in one line.
autocad_num_lines	The number of lines in the multi-line set.: Range: Number of lines.
autocad_mline_type	The type of multi-line object. Range: autocad_open autocad_closed
autocad_base_x	The x coordinate of the base point of the multi-line entity. Range: Any real number.
autocad_base_y	The y coordinate of the base point of the multi-line entity. Range: Any real number.
autocad_base_z	The z coordinate of the base point of the multi-line entity. Range: Any real number.

Inserts

autocad_entity: autocad_insert

Inserts are point features used in AutoCAD to specify block locations and associated attribution. Inserts are another way in which attribution is stored within an AutoCAD drawing file. The features returned from the AutoCAD reader encapsulate all the information from the AutoCAD insert entity and all attribute entities that are associated with the insert entity.

Insert features can be thought of as block references. They represent the location of an instance of a block definition. When passing features to the AutoCAD writer, those features with the type autocad_insert can be used to match existing block definitions in the TEMPLATE file.

If an attribute passed to the writer is defined by an AutoCAD Attribute Definition in the TEMPLATE file, then the placement of the attribute is taken from the TEMPLATE file unless it is overridden by the attributes shown in the table below.

If the position of the attribute is not specified in a TEMPLATE file and is not specified in the attributes below, then the attributes are placed at the insert location. Apart from the user-defined attributes specified within it, each insert entity also has the following attributes.

Attribute Name	Contents
autocad_xscale	<p>The scale factor for the inserted block in the x direction.</p> <p>Range: Any real number. Default: 1</p>
autocad_yscale	<p>The scale factor for the inserted block in the y direction.</p> <p>Range: Any real number Default: 1</p>
autocad_zscale	<p>The scale factor for the inserted block in the z direction.</p> <p>Range: Any real number Default: 1</p>
autocad_size_x	<p>The size of the inserted block in ground units in the x direction. This value will be used to set the scale factor of the inserted block, and takes precedence over the value for autocad_xscale</p> <p>Range: Any positive real number. Default: No default</p>
autocad_size_y	<p>The size of the inserted block in ground units in the y direction. This value will be used to set the scale factor of the inserted block, and takes precedence over the value for autocad_yscale.</p> <p>Range: Any positive real number. Default: No default</p>
autocad_size_z	<p>The size of the inserted block in ground units in the z direction. This value will be used to set the scale factor of the inserted block, and takes precedence over the value for autocad_zscale.</p> <p>Range: Any positive real number. Default: No default</p>
autocad_rotation	<p>The rotation of the inserted block, counterclockwise from horizontal.</p> <p>Range: -360.0 ..360.0 Default: 0</p>
autocad_number_columns	<p>The column count for the insert.</p> <p>Range: 0..65536 Default: 1</p>

Attribute Name	Contents
autocad_number_rows	<p>The row count for the insert.</p> <p>Range: 0..65536</p> <p>Default: 1</p>
autocad_column_distance	<p>The column spacing for the insert.</p> <p>Range: Any real number > 0</p> <p>Default: 0</p>
autocad_row_distance	<p>The row spacing for the insert.</p> <p>Range: Any real number > 0</p> <p>Default: 0</p>
autocad_block_name	<p>The name of the block entity which is to be inserted.</p> <p>Range: char[33]</p> <p>Default: FMEBLOCK<block_number> where block_number is some unique positive integer > 0.</p>
autocad_block_insert_x autocad_block_insert_y autocad_block_insert_z	<p>The insert point location of the block reference entity.</p> <p>When reading, these attributes are added to the block reference component entities when the reader directives RESOLVE_BLOCKS and STORE_INSERT_POINT are both set to YES.</p> <p>When writing, these attributes set the insert point of blocks during automatic block creation.</p> <p>Range: Any real number</p> <p>Default: 0</p>
autocad_attributes_follow	<p>Used during writing to indicate if attributes are also to be stored with the insert entity. This must be specified if feature attributes are to be written to the AutoCAD output file.</p> <p>This attribute affects both the creation of block attributes when the DEFAULT_ATTR_STORAGE directive is set to INSERT_ATTRIBUTES and the creation of blocks during automatic block creation.</p> <p>Range: true false</p> <p>Default: true</p>
autocad_attribute_display	<p>Indicates if the attribute values are to be visible or invisible. This will not override the visibility flag found in an existing template file attribute definition.</p>

Attribute Name	Contents
	<p>Range: visible invisible Default: invisible</p>
autocad_attr_def_tag	<p>Indicates the tag used for "Attribute" definition. (read-only) Range: text string</p>
autocad_attr_def_prompt	<p>Specifies the prompt that is displayed when you insert a block containing this attribute definition. (read-only) Range: text string</p>
autocad_attr_def_default	<p>Specifies the default attribute value. (read-only) Range: text string</p>
fme_attr_info{N}.field_name	<p>This list attribute hold the name of the Nth attribute. Range: text string</p>
fme_attr_info{N}.field_size	<p>This list attribute hold the size of the Nth attribute. Range: integer Default: 0</p>
fme_attr_info{N}.field_value	<p>This list attribute holds the value of the Nth attribute. Range: text string</p>
autocad_<attr_name>_x autocad_<attr_name>_y autocad_<attr_name>_z fme_attr_info{N}.location_x fme_attr_info{N}.location_y fme_attr_info{N}.location_z	<p>Used when attributes are associated with the insert elements, enabling the location of the attributes to be specified for display purposes. This specifies the exact location where the attributes are to be placed. Note that the FME attributes may contain different values than the AutoCAD attributes. This is because the values of the FME attributes will be modified if the justification is not base-line_left in an attempt to be more useful when translating into or out of other formats that support these FME attributes. Range: any 64-bit floating point value Default: x, y, and z value of insert coordinate (for the AutoCAD attributes) 0,0,0 (for the FME attributes)</p>

Attribute Name	Contents
autocad_<attr_name>_alignment_x autocad_<attr_name>_alignment_y autocad_<attr_name>_alignment_z fme_attrib_info{N}.align_x fme_attrib_info{N}.align_y	<p>Used when attributes are associated with the insert elements, enabling the location of the attributes to be specified for display purposes. This specifies the alignment location where the attributes are to be placed. Note that the FME attributes may contain different values than the AutoCAD attributes. This is because the values of the FME attributes will be modified if the justification is not <code>baseline_left</code> in an attempt to be more useful when translating into or out of other formats that support these FME attributes.</p> <p>Range: any 64-bit floating point value Default: 0,0,0 (for the AutoCAD attributes) the value of <code>fme_attrib_info{N}.location_[x y z]</code> (for the FME attributes)</p>
autocad_<attr_name>_justification	<p>The justification of the attribute relative to its insertion point (not its alignment point).</p> <p>Range: <code>autocad_top_left autocad_top_center autocad_top_right autocad_top_middle autocad_top_aligned autocad_top_fit autocad_middle_left autocad_middle_center autocad_middle_right autocad_middle_middle autocad_middle_aligned autocad_middle_fit autocad_bottom_left autocad_bottom_center autocad_bottom_right autocad_bottom_middle autocad_bottom_aligned autocad_bottom_fit autocad_baseline_left autocad_baseline_center autocad_baseline_right autocad_baseline_middle autocad_baseline_aligned </code></p>

Attribute Name	Contents
	autocad_baseline_fit Default: No default
fme_attrib_info{N}.justification	The justification of the attribute relative to its insertion point (not its alignment point). Range: top_left top_center top_right top_middle top_aligned top_fit middle_left middle_center middle_right middle_middle middle_aligned middle_fit bottom_left bottom_center bottom_right bottom_middle bottom_aligned bottom_fit baseline_left baseline_center baseline_right baseline_middle baseline_aligned baseline_fit Default: baseline_left
fme_attrib_info{N}.generation	The generation of the N th attribute. Range: normal upside_down backwards upsidedown_backwards Default: normal
autocad_<attr_name>_style fme_attrib_info{N}.style	The name of the text style for the attribute. For this attribute to be used by the Writer, a template file containing the text style must be specified. Default: No default
fme_attrib_info{N}.width_factor	The scaling applied in the x direction, which makes the text wider or narrower. However, this doesn't affect the

Attribute Name	Contents
	<p>weight (i.e., boldness) of the text string.</p> <p>Range: Any real number > 0 and <= 10000</p> <p>Default: 1</p>
autocad_<attr_name>_color	<p>The color of the attribute. When reading, this value is always 256, meaning COLOR_BYLAYER.</p> <p>Default: The color of the layer on which the insert is placed (not the same as COLOR_BYLAYER).</p>
fme_attrib_info{N}.color.red fme_attrib_info{N}.color.green fme_attrib_info{N}.color.blue	<p>These list attributes hold the color of the Nth attribute in RGB values, ranged between 0.0 and 1.0.</p> <p>Range: real number 0.0-1.0 (inclusive)</p> <p>Default: No default</p>
fme_attrib_info{N}.color.source	<p>The source of the color. If the attribute's color does not come from the block or layer, then it is given the value explicit.</p> <p>Range: use_layer use_block explicit</p> <p>Default: explicit</p>
autocad_<attr_name>_rotation fme_attrib_info{N}.rotation	<p>This specifies the rotation of the attribute, measured in degrees.</p> <p>Range: degree of rotation measured counter-clockwise from the horizontal.</p> <p>Default: 0</p>
fme_attrib_info{N}.oblique	<p>This specifies the obliquing angle of the attribute, which causes the text to lean to the right or left.</p> <p>Range: -85.0 ..85.0</p> <p>Default: 0</p>
autocad_<attr_name>_height fme_attrib_info{N}.height	<p>This specifies the height of the attribute in ground units.</p> <p>Range: any 64-bit floating point value.</p> <p>Default: 1</p>
autocad_<attr_name>_attribute_flag fme_attrib_info{N}.attribute_flag	<p>A bit-coded value. Values can be combined by using addition. Possible values are:</p> <p>1 = The attribute is invisible (does not</p>

Attribute Name	Contents
	<p>appear).</p> <p>2 = The attribute contains a constant value.</p> <p>4 = Verification is required on input of this attribute.</p> <p>8 = Attribute is preset (no prompt during insertion).</p> <p>If the <code>fme_attrib_info{N}</code>-<code>.attribute_flag</code> attribute is used, the invisibility bit will get overwritten by the <code>fme_attrib_info{N}.isvisible</code> attribute.</p> <p>If the <code>autocad_<attr_name>_attribute_flag</code> attribute is used, the invisibility bit will get overwritten by <code>autocad_attribute_display</code> if the attributes <code>autocad_visible_attributes{}</code> and <code>autocad_invisible_attributes{}</code> are specified.</p> <p>Default: 0 if no attribute definitions exist for the attribute (i.e., when <code>DEFAULT_ATTR_STORAGE</code> keyword or <code>autocad_attributes</code> attribute is set to <code>insert_attributes</code>), or value from the attribute definition if attribute definitions are used.</p>
<p><code>autocad_<attr_name>_layer</code> <code>fme_attrib_info{N}.layer</code></p>	<p>The layer on which the attribute definition, corresponding to this attribute, was created.</p> <p>Default: none</p>
<p><code>fme_attrib_info{N}.isVisible</code></p>	<p>These list attributes indicate whether or not the Nth attribute should be displayed.</p> <p>Range: TRUE FALSE</p> <p>Default: FALSE</p>
<p><code>autocad_visible_attributes{}</code></p>	<p>The list of attributes that are set to be visible. This is a list attribute. This list will override the visibility flag found in an existing template file attribute definition.</p>
<p><code>autocad_invisible_attributes{}</code></p>	<p>The list of attributes that are set to be invisible. This is a list attribute. This list</p>

Attribute Name	Contents
	will override the visibility flag found in an existing template file attribute definition.
autocad_<attr_name>_x_off autocad_<attr_name>_y_off autocad_<attr_name>_z_off	Used when attributes are associated with the insert elements enabling the location of the attributes to be specified for display purposes. This specifies the offset from the location of the insert. Range: any 64-bit floating point value Default: x, y and z value of insert coordinate

Dimensions

autocad_entity: autocad_dimension

Dimensions are aggregate features used in AutoCAD to specify dimensions within an AutoCAD drawing. The dimension features have the attributes described below.

Rotated (linear) dimensions:

- The first extension line is specified by **defpt2**.
- The second extension line is specified by **defpt3**.
- The dimension line is specified by **dimlinedefpt**.

Angular dimensions:

- **defpt2** and **defpt3** are the endpoints of the first extension line.
- **dimlinedefpt** and **defpt4** are the endpoints of the second extension line.
- **arcdefpt** specifies the dimension line arc.

Angular 3-point dimensions:

- **defpt4** is the vertex of the angle.
- **defpt2** is the endpoint of the first extension line.
- **defpt3** is the endpoint of the second extension line.
- **dimlinedefpt** specifies the arc for the dimension line.

Diameter dimensions:

- **defpt4** is the point selected on the circle or arc being dimensioned.
- **dimlinedefpt** is the point on the circle exactly across from the selection point.

Ordinate dimensions:

- **defpt3** is the point which was selected.
- **defpt4** is the point indicating the endpoint of the leader.

Attribute Name	Contents
autocad_actual_measurement	The scale factor for the inserted block in the x direction.

Attribute Name	Contents
	<p>Range: Any real number Default: 1</p>
autocad_arc_defpt.x autocad_arc_defpt.y autocad_arc_defpt.z	<p>This defines the dimension arc for an angular dimension. This is equivalent to the 16,26,36 group in DXF. Range: Any real number Default: none</p>
autocad_arc_defpt2.x autocad_arc_defpt2.y autocad_arc_defpt2.z	<p>Definition Point. This is equivalent to the 13,23,33 group in DXF. Range: Any real number Default: none</p>
autocad_arc_defpt3.x autocad_arc_defpt3.y autocad_arc_defpt3.z	<p>Definition Point. This is equivalent to the 14,24,34 group in DXF. Range: Any real number Default: none</p>
autocad_arc_defpt4.x autocad_arc_defpt4.y autocad_arc_defpt4.z	<p>Definition Point. This is equivalent to the 15,25,35 group in DXF. Range: Any real number Default: none</p>
autocad_dimension_attach_point	<p>This is the attachment point of dimension, as defined</p> <ul style="list-style-type: none"> 1 = Top Left 2 = Top Center 3 = Top Right 4 = Middle Left 5 = Middle Center 6 = Middle Right 7 = Bottom Left 8 = Bottom Center 9 = Bottom Right
autocad_dim_arrowhead_pt_<number>.x autocad_dim_arrowhead_pt_<number>.y	<p>These attributes, provided by the reader but not used by the writer, describe the x,y coordinates of the vertex, on each arrowhead, that could be called the "tip of the arrow." <number> is some number between 1 and the number of arrowheads in the dimension. If there are no arrowheads, then this attribute will not be supplied.</p>

Attribute Name	Contents
autocad_dimension_flag	This is the raw value from the AutoCAD file that indicates the type of dimension. See autocad_dimension_type for the decoded version of this.
autocad_dimension_style_name	The name of the dimension style used. When using the AutoCAD writer, it is important that the dimension style used is defined in the template file; otherwise, no style will be set and the dimension text will not be displayed. Even if the standard dimension style is used, the template file must hold a definition for it.
autocad_dimension_type	This indicates the type of the autocad dimension. Possible values are: autocad_rotated autocad_aligned autocad_angular autocad_diameter autocad_radius autocad_angular3Pt autocad_ordinate autocad_xordinate
autocad_leader_length	This is the length of the dimension leader line.
autocad_linespace_style	This is the style of the line spacing. It is either 1 (at least) or 2 (exact).
autocad_linespace_factor	The percentage of default line spacing used. Range: 0.25..4.0 Default: 1
autocad_rotation_angle	The rotation angle of the dimension.
autocad_text_midpoint.x autocad_text_midpoint.y autocad_text_midpoint.z	The midpoint of the text.
autocad_text_rotation	The rotation of the dimension text.
autocad_text_size	The size of the text in ground units.
autocad_text_string	The dimension text value.
autocad_ucs_xangle	The angle of the ucs (user coordinate system) when the dimension was created.

Group

autocad_entity: autocad_group

Group features are features with no geometry. This feature merely identifies the feature handles that are part of the group. This entity is only supported by the AutoCAD Reader. The feature type is set to AUTOCAD_GROUP. The following attributes are set when reading groups.

Attribute Name	Content
autocad_group_description	The descriptive name of the group. Range: Character string.
autocad_group_name	Name of the group. Range: Character string.
autocad_group_anonymous	Whether or not group is anonymous Range: yes/no.
autocad_group_accessible	Whether or not group is accessible. Range: yes/no.
autocad_group_num_entities	Number of entities in the group. Range: Numeric.
autocad_group_selectable	Whether or not group is selectable. Range: yes/no.
autocad_entity_handle{}	The list attribute which contains the hexadecimal values of the entities that make up the group. Range: Hexadecimal value.

Hatches

autocad_entity: autocad_hatch

Hatch features represent AutoCAD hatch entities. They are composed of two dimensional boundary loops that define areas which can be filled with line patterns or color gradients. The loops of each hatch are closed, simple, continuous, and are not self-intersecting except at their endpoints.

The AutoCAD Reader creates features with varied geometry depending on the geometry of the loops that compose them. Features created may be of polygon, donut or aggregate geometry, where the aggregates may contain either donuts and polygons or just ordered polygons depending on the usage of the PRESERVE_COMPLEX_HATCHES keyword. The AutoCAD reader will also preserve polyline bulge information.

Note: The AutoCAD Reader has the following limitations when reading hatch features: associative hatches are not preserved, one pattern is allowed per hatch, only one or two color gradients are supported, unclosed hatch boundary loops are closed, and splines are not supported for hatch boundary loops. In addition, elliptical and circular arcs in boundary loops are stroked unless the reader is using enhanced geometry.

The AutoCAD Writer has the following limitations when writing hatch features: hatch features must be closed area features of polygon, donut, or aggregate geometry according to how the AutoCAD Reader created them. The AutoCAD writer will try to reconstruct polyline bulge information from the feature, but closed and elliptical arcs will be stroked into line segments.

The following attributes may be set when reading hatches.

Attribute Name	Content
autocad_hatch_associative <i>Used only for reading</i>	The flag indicating if the hatch is associative. Range: 0 for no 1 for yes Default: 0
autocad_hatch_complex_mode	The flag that represents whether the hatch feature was created to preserve complex hatches. This indicates how the structure of loops is created. Range: 0 for no 1 for yes Default: 0
autocad_hatch_gradient_angle	The angle of the gradient fill for the hatch feature in degrees. Range: any 64-bit floating point value. Default: 0
autocad_hatch_gradient_color1	The first color used to interpolate a two color gradient fill. It the single color used in a one color gradient fill. Specified as a character string of comma-separated red, green and blue values. Range: 0..255,0..255,0..255. Default: None.
autocad_hatch_gradient_color2	The second color used to interpolate a two color gradient fill. Specified as a character string of comma-separated red, green and blue values. Range: 0..255,0..255,0..255 Default: None.
autocad_hatch_gradient_name	The name of the gradient. Must be a predefined value for predefined gradients. This is mandatory for hatches with gradients. Range: Curved Cylinder Hemispherical Linear Spherical Invcurved Invcylinder Invhemispherical Invspherical Default: None
autocad_hatch_gradient_one_color_mode	The flag indicating whether only one color should be used in gradient calculation. Gradients can be two color, or one color with a luminance value set by autocad_hatch_shade_tint_value.

Attribute Name	Content
	<p>Range: 0 for no 1 for yes Default: 0</p>
autocad_hatch_gradient_shift	<p>The interpolation value between the default and shifted values of the gradient's definition. Range: 0..1 Default: None</p>
autocad_hatch_gradient_type	<p>The type of the gradient. Currently this is set to 0 for predefined gradient. In the future a value of 1 may be supported for user-defined gradients. This is only used for hatches with gradients. Range: 0 for pre-defined 1 for user-defined Default: None</p>
autocad_hatch_loop{}.autocad_hatch_bulge{} <i>Applicable only with classic geometry</i>	<p>The list of bulge values for polyline bulge arcs in each of a list of hatch boundary loops. The list of bulge values parallel the vertices in each loop. A bulge value represents the tangent of 1/4 the included angle in the arc measured counterclockwise. A value of 0 represents a line, and a value of 1 represents a semicircle. Range: 0..1 Default: 0</p>
autocad_hatch_loop{}.autocad_hatch_bulges_present <i>Applicable only with classic geometry</i>	<p>The flag indicating if polyline bulge arcs exist in each of a list of hatch boundary loops. Range: 0 for no 1 for yes Default: 0</p>
autocad_hatch_loop{}.autocad_hatch_type <i>Applicable only with classic geometry</i>	<p>The type of the hatch loop. This is an integer representing the addition of applicable type flags. Range: numeric value Default: None</p>
autocad_hatch_object_type	<p>The general type of the hatch, specifying the usage of either patterns or gradients. Range: 0 for classic hatch 1 for color gradient</p>

Attribute Name	Content
	Default: 0
autocad_hatch_origin_point_x	The x-axis coordinate of the origin of the hatch in world coordinates. Range: any 64-bit floating point value Default: 0
autocad_hatch_origin_point_y	The y-axis coordinate of the origin of the hatch in world coordinates. Range: any 64-bit floating point value Default: 0.
autocad_hatch_pattern_angle	The angle of the pattern fill for the hatch feature in degrees. Range: any 64-bit floating point value Default: 0
autocad_hatch_pattern_double	The flag indicating if the hatch pattern is doubled by adding a second set of lines at 90 degrees to the first. This is only used for user-defined patterns. Range: 0 for no 1 for yes Default: 0
autocad_hatch_pattern_name	The name of the pattern. May be a predefined value for predefined patterns, a predefined value for custom patterns, or any string for user-defined patterns. Range: Pre-defined or custom pattern name any string Default: SOLID
autocad_hatch_pattern_scale	This represents the scaled size of the pattern for pre-defined and custom-defined patterns. Range: positive floating point value > 0 Default: 1
autocad_hatch_pattern_space	This represents the space between the parallel lines of the hatch pattern. This is only used for user-defined patterns. Range: positive floating point value > 0 Default: 1

Attribute Name	Content
autocad_hatch_pattern_type	The type of the hatch pattern. Custom-defined patterns are pre-created patterns that must be present in the location of the predefined patterns. This is only used for hatches with patterns. Range: 0 for user-defined 1 for predefined 2 for custom-defined Default: None
autocad_hatch_pixel_size	The size of pixels for intersection and ray casting when drawing the hatch. Range: positive floating point value > 0 Default: 1
autocad_hatch_shade_tint_value	The luminance value of the hatch. If the hatch has a gradient and is using one color mode, this value is applied to the first color. Range: 0.0..1.0 Default: 0

MPolygons

autocad_entity: autocad_mpolygon

MPolygon features represent AutoCAD mpolygon entities. They are composed of two-dimensional polyline loops defining areas that can be filled with line patterns or color gradients. The loops of each mpolygon are closed, simple, continuous, and are not self-intersecting except at their endpoints. This is very similar to the definition of hatch entities.

The AutoCAD Reader creates features with varied geometry depending on the geometry of the loops that compose each mpolygon. Features created may be of polygon, donut or aggregate geometry, where the aggregates may contain a combination of donuts and polygons.

Note: The AutoCAD Reader has the following limitations when reading mpolygon features: one pattern is allowed per hatch, only one or two color gradients are supported, unclosed boundary loops are closed, and splines are not supported for boundary loops.

The AutoCAD Writer has the following limitations when writing mpolygon features: the features must be closed area features of polygon, donut, or aggregate geometry according to how they were created by the AutoCAD Reader.

The following attributes may be set when reading mpolygons.

Attribute Name	Content
autocad_mpolygon_associative <i>Used only for reading</i>	The flag indicating if the mpolygon is associative. Range: 0 for no 1 for yes Default: 0

Attribute Name	Content
autocad_mpolygon_gradient_angle	<p>The angle of the gradient fill for the mpolygon feature in degrees.</p> <p>Range: any 64-bit floating point value.</p> <p>Default: 0</p>
autocad_mpolygon_gradient_color1	<p>The first color used to interpolate a two color gradient fill. It the single color used in a one color gradient fill. Specified as a character string of comma-separated red, green and blue values.</p> <p>Range: 0..255,0..255,0..255.</p> <p>Default: None.</p>
autocad_mpolygon_gradient_color2	<p>The second color used to interpolate a two color gradient fill. Specified as a character string of comma-separated red, green and blue values.</p> <p>Range: 0..255,0..255,0..255</p> <p>Default: None.</p>
autocad_mpolygon_gradient_name	<p>The name of the gradient. Must be a predefined value for predefined gradients. This is mandatory for mpolygons with gradients.</p> <p>Range: Curved Cylinder Hemispherical Linear Spherical Invcurved Invcylinder Invhemispherical Invspherical</p> <p>Default: None</p>
autocad_mpolygon_gradient_one_color_mode	<p>The flag indicating whether only one color should be used in gradient calculation. Gradients can be two color, or one color with a luminance value set by autocad_mpolygon_shade_tint_value.</p> <p>Range: 0 for no 1 for yes</p> <p>Default: 0</p>
autocad_mpolygon_gradient_shift	<p>The interpolation value between the default and shifted values of the gradient's definition.</p> <p>Range: 0..1</p> <p>Default: None</p>
autocad_mpolygon_gradient_type	<p>The type of the gradient. Currently this is set to 0 for predefined gradient. In the future a value of 1 may be supported for user-defined gradients. This is mandatory for mpolygons with gradients.</p> <p>Range: 0 for pre-defined 1 for user-defined</p> <p>Default: None</p>
autocad_mpolygon_object_type	<p>The type of the mpolygon. This is mandatory for mpolygons with gradients and patterns other than</p>

Attribute Name	Content
	<p>SOLID.</p> <p>Range: 0 for classic mpolygon 1 for color gradient</p> <p>Default: 0</p>
autocad_mpolygon_origin_point_x	<p>The x-axis coordinate of the origin of the mpolygon in world coordinates.</p> <p>Range: any 64-bit floating point value</p> <p>Default: 0</p>
autocad_mpolygon_origin_point_y	<p>The y-axis coordinate of the origin of the mpolygon in world coordinates.</p> <p>Range: any 64-bit floating point value</p> <p>Default: 0.</p>
autocad_mpolygon_offset_x Used only for reading	<p>The offset along the x-axis coordinate of the center point of the mpolygon extents in world coordinates.</p> <p>Range: any 64-bit floating point value</p> <p>Default: 0</p>
autocad_mpolygon_offset_y Used only for reading	<p>The offset along the y-axis coordinate of the center point of the mpolygon extents in world coordinates.</p> <p>Range: any 64-bit floating point value</p> <p>Default: 0.</p>
autocad_mpolygon_pattern_angle	<p>The angle of the pattern fill for the mpolygon feature in degrees.</p> <p>Range: any 64-bit floating point value</p> <p>Default: 0</p>
autocad_mpolygon_pattern_double	<p>The flag indicating if the mpolygon pattern is doubled by adding a second set of lines at 90 degrees to the first. This is only used for user-defined patterns.</p> <p>Range: 0 for no 1 for yes</p> <p>Default: 0</p>
autocad_mpolygon_pattern_name	<p>The name of the pattern. May be a predefined value for predefined patterns, a predefined value for custom patterns, or any string for user-defined patterns.</p> <p>Range: Pre-defined or custom pattern name any string</p> <p>Default: SOLID</p>
autocad_mpolygon_pattern_scale	<p>This represents the scaled size of the pattern for predefined and custom-defined patterns.</p> <p>Range: positive floating point value > 0</p> <p>Default: 1</p>
autocad_mpolygon_pattern_space	<p>This represents the space between the parallel lines</p>

Attribute Name	Content
	<p>of the mpolygon pattern. This is only used for user-defined patterns.</p> <p>Range: positive floating point value > 0</p> <p>Default: 1</p>
autocad_mpolygon_pattern_type	<p>The type of the mpolygon pattern. Custom-defined patterns are pre-created patterns that must be present in the location of the predefined patterns. This is mandatory for mpolygons with patterns other than SOLID.</p> <p>Range: 0 for user-defined 1 for pre-defined 2 for custom-defined</p> <p>Default: 1</p>
autocad_mpolygon_pixel_size	<p>The size of pixels for intersection and ray casting when drawing the mpolygon.</p> <p>Range: positive floating point value > 0</p> <p>Default: 1</p>
autocad_mpolygon_shade_tint_value	<p>The luminance value of the mpolygon. If the mpolygon has a gradient and is using one color mode, this value is applied to the first color.</p> <p>Range: 0.0..1.0</p> <p>Default: 0</p>

Surfaces

autocad_entity: autocad_surface

Features with this value are used to store several AutoCAD entities including face, region, polygon mesh, polyface mesh, or surface entities. Extruded, planar, revolved, lofted, and swept surfaces are represented as autocad_surface, which may contain multiple unconnected surfaces, each of which is composed of faces, which may or may not be planar. This value is used by both the reader and the writer.

Surfaces support appearances, but only one appearance per surface. For two-sided surfaces, the writer will split the surface into one surface per side. Furthermore, only subdivision mesh entities support textured appearances. Subdivision meshes will preferentially be written from surfaces with textured appearances. (Textured appearances are only supported by RealDWG.)

For writing, all 3D surface geometry types are supported. Any types of 3D geometry which are not directly supported as entities are decomposed into triangulated mesh representation prior to writing.

Attribute Name	Content
autocad_subdmesh_base_faces	<p>This is used by the Reader only and indicates the number of faces in a subdivision mesh at the base smoothness level of 0.</p> <p>Range: A 32 bit integer value. Default: None</p>

<code>autocad_subdmesh_base_vertices</code>	This is used by the Reader only and indicates the number of vertices in a subdivision mesh at the base smoothness level of 0. Range: A 32 bit integer value. Default: None
<code>autocad_subdmesh_smooth_level</code>	This is used by the Reader only and indicates the smoothness level of a subdivision mesh. A value of 0 represents the base smoothness of the mesh geometry and higher values indicate greater subdivisions which increase smoothness. Range: 0-4. Default: 0
<code>autocad_subdmesh_smoothed_faces</code>	This is used by the Reader only and indicates the number of faces in a subdivision mesh at the current smoothness level. Range: A 32 bit integer value. Default: None
<code>autocad_subdmesh_smoothed_vertices</code>	This is used by the Reader only and indicates the number of vertices in a subdivision mesh at the current smoothness level. Range: A 32 bit integer value. Default: None
<code>autocad_subdmesh_watertight</code>	This is used by the Reader only and indicates whether a subdivision mesh is watertight, based on adjacent polygons sharing common vertices. Range: Yes No. Default: None

3D Solids

`autocad_entity: autocad_solid3d`

Features with this value are used to store both AutoCAD 3D solid and body entities. Cone, elliptical cone, cylinder, elliptical cylinder, revolve, sphere, and torus 3D solids may be represented by an `autocad_solid3d`. This value is used by both the reader and the writer.

Closed surface boundary representations of 3D geometric volumes which may contain representations of multiple unconnected 3D solids may be stored as surfaces or multi-surfaces on read. Most 3D solids will be represented as surface boundaries and will be handled as `autocad_surface` type features on write.

For writing, all 3D solid geometry types are supported. Any types of 3D geometry which are not directly supported as entities are decomposed into triangulated mesh representation prior to writing.

Autodesk MapGuide SDL Reader/Writer

The Autodesk® MapGuide SDL Reader and Writer modules allow FME to read and write SDL files. The SDL file format is an ASCII format used with AutoDesk's MapGuide and other World Wide Web map authoring tools.

Overview

SDL data can be either two-dimensional (2D) or three-dimensional (3D).

SDL files store both geometry and attributions. A logical SDL dataset consists of one or more files in the same directory with the extension .sdl. This extension is added to the basename of the SDL files.

The SDL reader and writer support the storage of *point*, *line*, and *polygon* geometric data in **.sdl** files. Output files contain only one geometry type to conform with MapGuide. The SDL format can also store features with no geometry. Features that have no geometry are referred to as having a geometry of none.

SDL Quick Facts

Format Type Identifier	SDL
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	Directory or File
Feature Type	File base name
Typical File Extensions	.sdl
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type Attribute	sdl_type

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	point	yes
circles	no	polygon	yes
circular arc	no	raster	no
donut polygon	yes	solid	no
elliptical arc	no	surface	no
ellipses	no	text	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
line	yes		z values	yes (reader only)
none	yes			

Reader Overview

The SDL reader first scans the directory it is given for SDL files that have been defined in the mapping file. The SDL reader then extracts features from the files one at a time, and passes them on to the rest of the FME for further processing. Optionally a single SDL file can be given as the dataset. In this case, only that SDL file is read.

Reader Directives

The directives processed by the SDL reader are listed below. The suffixes shown are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the SDL reader is **SDL**.

DATASET

Required/Optional: *Required*

The value for this keyword is the directory containing the SDL files to be read, or a single SDL file. A typical mapping file fragment specifying an input SDL dataset looks like:

```
SDL_DATASET /usr/data/sdl/92i080
```

Workbench Parameter: *Source Autodesk MapGuide SDL File(s)*

IDs

Required/Optional: *Optional*

This specification is used to limit the available and defined SDL files read. The syntax of the **IDs** keyword is:

```
<ReaderKeyword>_IDs <baseName1> \
                    <baseName2> \
                    <baseNameN>
```

The basenames must match those used in **DEF** lines. The example below selects only the **roads** SDL file for input during a translation:

```
SDL_IDS roads
```

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the **mitab.dll** in the FME home directory to **mapinfo.dll**.

The syntax of the **MAPINFO_SEARCH_ENVELOPE** directive is:

```
MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

```
MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The SDL writer outputs each feature type into a separate file in order to comply with AutoDesk MapGuide. Each feature has the following associations: vertices, a name, an ID, and a Universal Resource Locator (URL).

Writer Directives

The directives that are processed by the SDL writer are listed below. The suffixes shown are prefixed by the current `<WriterKeyword>_` in a mapping file. By default, the `<WriterKeyword>` for the SDL writer is `SDL`.

DATASET

Required/Optional: *Required*

The value for this keyword is the name of the created SDL directory. If a directory of this name exists, it is replaced by the new SDL. A typical mapping file fragment specifying an output SDL dataset looks like:

```
SDL_DATASET /tmp
```

Workbench Parameter: *Destination Autodesk MapGuide SDL Directory*

DEF

Required/Optional: *Required*

The SDL writer uses SDL_DEF lines to define files to write features to. A typical mapping file fragment specifying an output SDL file looks like:

```
SDL_DEF roads
```

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), special FME feature attributes direct the SDL writer as it renders the feature into the image. The most important of these is the `sdl_type` attribute, which controls the overall interpretation of the feature. The correct values for `sdl_type` are `sdl_line`, `sdl_point`, and `sdl_polygon`. The parameters specified for each of these are described in the following subsections, and the attributes common to each are given in the following table:

Attribute Name	Contents
<code>sdl_url</code>	Specifies a URL for the line, polygon or point. Required: No Default: NULL
<code>sdl_name</code>	Specifies an internal name for the line, polygon or point. Required: No Default: NULL
<code>sdl_id</code>	Specifies an ID for the line, polygon or point. Required: No Default: NULL

Lines

sdl_type: `sdl_line`

The SDL writer outputs a line object containing the points as specified in the input file. Also, the SDL writer outputs a URL, a name, and an ID number associated with the line object as attributes.

Points

sdl_type: sdl_point

The SDL writer will output a point object containing the points as specified in the input file. Also, the SDL writer will output a URL, a name, and an ID number associated with the point object as attributes.

Polygons

sdl_type: sdl_polygon

The SDL writer outputs a polygon object containing the points as specified in the input file. Also, the SDL writer outputs a URL, a name, and an ID number associated with the polygon object as attributes.

BC MOEP Reader/Writer

The British Columbia (BC) Ministry of Environment and Parks (MOEP) format is a compact binary format used in the province of B.C., Canada. MOEP features have few attributes, one of which is a feature code which encodes the feature's properties. MOEP files can store only integer coordinates.

The MOEP Reader and Writer enables FME to read and write files in binary MOEP format, with either 16-bit or 32-bit integer coordinates. Support for ASCII MOEP files is not provided.

Note: Throughout this section, a binary MOEP file will be referred to simply as an MOEP file; this reader/writer provides no support for ASCII MOEP files.

Overview

Each MOEP file starts with a small header, which is immediately followed by a sequence of geometric features. The header contains information which is global to the MOEP file, including a file type, a name for the content of the file such as, a mapsheet ID, and whether the coordinates are specified with 16-bit or 32-bit integers. Each feature has a feature code, a single optional attribute, a geometric type, such as point, line, text, etc., and some type-specific information, like coordinates, rotation, text size, etc.

The FME considers an MOEP data set to be a collection of MOEP files in a single directory.

MOEP files are referred to in the mapping file by IDs rather than by physical file names. The mapping between IDs and physical names is defined by the MOEP file definition lines within the mapping file.

BC MOEP Quick Facts

Format Type Identifier	MOEP
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	Directory or File
Feature Type	File base name
Typical File Extensions	.arc, .bin
Automated Translation Support	Yes for Reader No for Writer
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type Attribute	moep_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	no
circular arc	no		raster	no
donut polygon	no		solid	no
elliptical arc	no		surface	no
ellipses	no		text	yes
line	yes		z values	yes
none	no			

Reader Overview

The MOEP reader produces FME features for all the feature data held in MOEP files residing in a given directory. The MOEP reader first scans the directory it is given for the MOEP files which have been defined in the mapping file. For each MOEP file that it finds, it checks to see if it the ID corresponding to the file is requested by looking at the list of IDs specified in the mapping file. If a match is found or if no IDs were specified in the mapping file, the MOEP file is opened for read. The MOEP reader extracts features from the file one at a time, and passes them on to the rest of the FME for further processing. When the file is exhausted, the MOEP reader starts on the next file in the directory.

Reader Directives

The suffixes shown below are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the MOEP reader is **MOEP**.

DATASET

Required/Optional: *Required*

The value for this directive is the directory name of the input MOEP files, or a single MOEP file to be read. A typical mapping file fragment specifying an input MOEP data set looks like:

```
MOEP_DATASET /usr/data/moep/92i080
```

or

```
MOEP_DATASET /usr/data/moep/92i080/92i080a.bin
```

Workbench Parameter: *Source B.C. MOEP File(s)*

DEF

Required/Optional: *Optional*

The definition specifies the ID to use to refer to the file, along with the physical file name and its extension. In addition to the file name, other global attributes from the table below can be specified in the definition. When additional attributes are specified for an MOEP file being read, the reader will generate warnings if the specified values do not match those specified in the file's header. The writer uses the global attributes to fill in the header of the MOEP file being written.

The syntax of an MOEP **DEF** line is:

```
<ReaderKeyword>_DEF <fileID> \
    MOEP_FILENAME <physFileName> \
    [<attrName> <attrVal>]*
```

The following table shows the supported global attributes:

Attribute Name	Description
MOEP_FILENAME	Name of physical file within MOEP data set.
MOEP_RESOLUTION	The size of integer used to represent each X and Y coordinate value within the MOEP file. This can be either 16 or 32, indicating 16-bit or 32-bit integers, respectively. Z coordinates are always 16 bits, regardless of this attribute's value.
MOEP_FILE_TYPE	An integer in the range 0..9 denoting the type of data this file contains.
MOEP_NAME	An ASCII string 0 to 11 characters in length, providing a logical name for the file. This is stored in the file's header; it typically contains a mapsheet ID.
MOEP_FORCE_TYPE5	This optional attribute must be set to either yes or no. The default is no. If it is yes, a type 5 attribute record is written with each feature, even if it is empty.
MOEP_DATE	The date of submission of the MOEP file. The format for this date is YYMMDD, where YY is the last two digits of the year, MM is the month (01-12), and DD is the day within the month (01-31).
MOEP_OFFSET_MINIMUM	The MOEP writer module uses this value to determine the origin from which 16-bit (X,Y) coordinates are measured. As features are written to the MOEP file, their minimum bounding rectangle is maintained; once the MBR is larger than MOEP_OFFSET_MINIMUM in both the X and Y directions, its centre point is chosen as the origin for all coordinates written to the file. This attribute has no effect on 32-bit coordinates, which are always measured from (0,0).

The following mapping file fragment defines two MOEP files, one containing DEM data with 16-bit coordinates, and one containing contours, with 32-bit coordinates:

```
MOEP_DEF dem_data MOEP_FILENAME 92b053d.arc \
    MOEP_FILE_TYPE 1 \
    MOEP_RESOLUTION 16 \
    MOEP_NAME 92b053d \
    MOEP_DATE 960913\
    MOEP_OFFSET_MINIMUM 1000
MOEP_DEF contour_data MOEP_FILENAME 92b053t \
    MOEP_FILE_TYPE 2 \
    MOEP_RESOLUTION 32 \
    MOEP_NAME 92b053t \
    MOEP_DATE 960913
```

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined MOEP files read. If no IDs are specified, then all defined and available MOEP files are read. The syntax of the **IDs** keyword is:

```
<ReaderKeyword>_IDs    <fileID1> \  
    <fileID1> ... \  
    <fileIDn>
```

The fileIDs must match those used in DEF lines.

The example below selects only the **dem_data** MOEP file for input during a translation:

```
MOEP_IDS dem_data
```

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the **mitab.dll** in the FME home directory to **mapinfo.dll**.

The syntax of the **MAPINFO_SEARCH_ENVELOPE** directive is:

```
MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

```
MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The **COORDINATE_SYSTEM** directive, which specifies the coordinate system associated with the data to be read, must always be set if the **SEARCH_ENVELOPE_COORDINATE_SYSTEM** directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the **SEARCH_ENVELOPE_COORDINATE_SYSTEM** to the reader **COORDINATE_SYSTEM** prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

✳ Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

✳ Workbench Parameter

Additional Attributes to Expose

Writer Overview

The MOEP writer creates and writes feature data to MOEP files in the directory specified by the DATASET keyword. If the directory did not exist before the translation, the writer will create it. Any old MOEP files in the directory will be overwritten with the new feature data. The FME determines which file features are to be written to as they are routed to the MOEP writer. Many MOEP files can be written during a single FME session.

Writer Directives

The MOEP writer processes the DATASET and DEF directives as described in the *Reader Directives* section. Unlike the reader, the MOEP writer requires DEF directives to be specified. It does not make use of the IDs directive.

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Special FME attributes are used to hold the parameters specific to MOEP features. The MOEP writer uses these attributes to define aspects of the geometries of the features it writes out, and the MOEP reader will define these attributes from the MOEP features it reads.

One of these attributes is an optional user attribute which can contain up to 66 characters of arbitrary data.

FME considers the ID of the MOEP file to be the *FME feature type* of an MOEP feature. The feature type of an MOEP feature must match the ID of an MOEP file defined by an MOEP DEF line.

Every MOEP feature, regardless of its geometry type, shares the parameters shown in the following table. Subsequent subsections will describe parameters specific to each feature type.

Attribute Name	Contents
moep_type	The type of the geometry for the feature. This attribute will contain one of: moep_line moep_contour_line moep_point moep_text moep_arc
moep_code	Character string with up to 10 characters designating the feature code of the feature. If this is not specified for a feature being written, the feature will have the same feature code as the feature which was most recently written to the MOEP file.
moep_attribute	An optional attribute (MOEP type 05 feature) which can contain up to 66 characters of arbitrary text. (See also the moep_font, moep_weight, and moep_text_group attributes defined on moep_text features.)

Line Features

moep_type: moep_line

MOEP line features have two or more coordinates. FME features with an moep_type of moep_line correspond to non-contour MOEP features with a type of 02, 03, 12, or 13; the moep_display_type and moep_line_type differentiate between the different types.

The following attributes are defined for moep_line features:

Attribute Name	Contents
moep_display_type	Determines whether the line is a primary line or a duplicate. Legal values are primary and construction. The default is primary.
moep_line_type	Determines whether the MOEP feature is simple or complex (curvilinear). Legal values are curve and line. The default is line.

Contour Features

moep_type: moep_contour_line

MOEP contour line features have three or more coordinates. FME features with an moep_type of moep_contour correspond to MOEP features with a type of 02, 03, 12, or 13 which are represent contour data; the moep_display_type and moep_line_type differentiate between the different types.

Aside from the `moep_line_type` and `moep_display_type` attributes that contour lines inherit from `moep_line` features, the following attribute is defined for `moep_contour_line` features:

Attribute Name	Contents
<code>moep_contour_elevation</code>	The elevation of the contour line.

Point Features

moep_type: `moep_point`

In addition to an (X,Y,Z) location, an MOEP point has some additional attributes which affect the display of its point symbol. The symbol will always be centred around its location, but can be rotated and/or scaled, in both the X and Y directions.

Attribute Name	Contents
<code>moep_rotation</code>	Determines the rotation applied to the point symbol, measured in degrees counterclockwise from horizontal. The default is 0.0 degrees.
<code>moep_scale_x</code>	Multiplier applied to scale the point symbol in the X direction. (If this is not provided, it defaults to 1.0.
<code>moep_scale_y</code>	Multiplier applied to scale the point symbol in the Y direction. If this is not provided, it defaults to 1.0.

Arc Features

moep_type: `moep_arc`

MOEP arc features represent a directed circular segment between two points on an ellipse. The representation of an arc is a set of three (X,Y,Z) coordinates—start of arc, end of arc, and origin of arc—along with a the direction of the arc.

Attribute Name	Contents
<code>moep_sweep_direction</code>	The direction in which the arc is drawn. Legal values are <code>clockwise</code> and <code>counterclockwise</code> . The default is <code>clockwise</code> .

Text Features

moep_type: `moep_text`

MOEP text features represent textual annotation placed at specific world coordinates. The full specification of the geometry includes an (X,Y,Z) position, the rotation of the text, the text string itself, the size of the text, and a specification of font, weight, and text group number.

Attribute Name	Contents
<code>moep_rotation</code>	Determines the rotation applied to the text, measured in degrees counterclockwise from horizontal.

Attribute Name	Contents
moep_text_string	The characters which make up a line of the text feature. The maximum length of a line of text is 66 characters. Several text features can be grouped into a single feature using the moep_text_group attribute.
moep_text_size	The size of the text feature, measured in ground metres.
moep_font	Specifies a font number for the text, an integer in the range 0..99. See the discussion below this table regarding the encoding of font, weight, and text group.
moep_weight	Specifies the weight of the text, an integer in the range 0..99. See the discussion below this table regarding the encoding of font, weight, and text group.
moep_text_group	Specifies a group number; several text features can be logically grouped together by giving them the same group number. This number is a five digit, decimal integer. See the discussion below this table regarding the encoding of font, weight, and text group.

It is important to note the relationship between the font, weight, text group, and the optional attribute for the feature. If font, weight, and text group attributes are specified, MOEP uses the optional attribute of a text feature to store their values. When these are specified, the format of the attribute string is **FFFWWWGGGGGG**, where **FFF** is the font number, **WWW** is the weight, and **GGGGGG** is the text group number. Each number is right-justified in its field, padded to the left with spaces as necessary.

Similarly, when reading a text feature the optional attribute, if present, is broken down into a font, weight, and text group.

BC MoF Electronic Submission Framework (ESF) - Reader/Writer

Format Notes: This format is not supported by FME Base Edition.

Overview

BC Ministry of Forests (MoF) Electronic Submission Framework (ESF) is a set of XML/GML formats that allow clients to submit data electronically to BC Ministry of Forests and Range and Ministry of Agriculture and Lands. Four ESF formats are supported:

- **ESF_ABR: Electronic Submission Framework - As Built Roads**
- **ESF_FSP: Electronic Submission Framework - Forest Stewardship Plan**
- **ESF_FTA: Electronic Submission Framework - Forest Tenure Application:** The BC Ministry of Forests (MoF) Electronic Submission Framework (ESF) FTA is a GML format specifying Forest Tenures (FTA) submissions for the British Columbia Ministry of Forests Electronic Submission Framework.
- **ESF_RESULTS: Electronic Submission Framework - RESULTS:** The BC Ministry of Forests (MoF) Electronic Submission Framework (ESF) RESULTS is a GML format specifying silviculture (RESULTS) submissions for the British Columbia Ministry of Forests Electronic Submission Framework.

For more information, go to:

<http://www.safe.com/support/resources/esf/index.php>

<http://www.for.gov.bc.ca/his/esf/index.htm>

Reader Directives

The suffixes shown are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the ESF reader is `ESF`.

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Directives

The suffixes shown are prefixed by the current `<writerKeyword>` in a mapping file. By default, the `<writerKeyword>` for the ESF writer is `ESF`.

DESTINATION_DATASET

Required/Optional: Required

The file to which the should output the ESF text. If the file does not exist, it will be created.

Example:

```
DESTINATION_DATASET c:\fta_file.xml
```

Workbench Parameter: *Destination BC MoF ESF File*

SUBMISSION_FRAMEWORK

Required/Optional: Required

A submission framework must be selected in the settings box when adding a destination dataset. Once this selection has been set, it will be displayed in the destination parameters in Workbench. Note, however, that it cannot be changed after it has been set.

Example:

```
SUBMISSION_FRAMEWORK "ESF_ABR: Electronic Submission Framework - As Built Roads"
```

Workbench Parameter: Submission Framework

Bentley MicroStation Design Reader/Writer

The Bentley® MicroStation Design Reader/Writer allows FME to access files used by the MicroStation and Intergraph Interactive Graphics Design System (IGDS).

Intergraph made public the specification for this file format, which they call the Intergraph Standard File Format (ISFF)¹. This chapter assumes familiarity with this format.

Overview

Design files consist of a header, followed by a series of *elements*. The header contains global information including the transformation equation from design units to user coordinates, as well as the dimension of the elements in the file. Each element contains standard display information, such as its color, level, class, and style, as well as a number of attributes specific to its element type. For example, a text element has fields for font, size, and the text string in addition to the standard display attributes.

Tip: The IGDS reader and writer modules support both two- and three-dimensional Design files and cell libraries.

Individual design file elements must be less than a system-imposed maximum number of bytes. *Complex elements* solve this problem by physically grouping individual elements together into an object that will be manipulated as a whole. The FME transparently handles such complex elements as single FME features. This situation occurs when text elements are grouped together into a single complex element headed up by a text node, and when linear or polygonal features have more than 101 vertices (Microstation V7) or 5000 vertices (Microstation V8). *Cells* are complex elements used as symbols, and are treated as atomic entities by the FME.

Each IGDS file element may have one or more *attribute linkages* associated with it. The IGDS reader and writer support both user data and database linkages. (Note, however, that the DGN V8 reader and writer do not support the interpretation of user linkages. Database linkages and MSLINKS are supported. FRAMME linkages are supported for reading.) The linkage values may be used to join elements with attributes stored in relational tables through the use of the @Relate FME Transformation Function. Linkages may also be used to specify fill information for fillable IGDS area geometries such as Shape elements, and other application-specific data. (Note, however, that the `igds_fill_color` attribute will override any solid fill color linkage specification if both are present.)

Because Design files support three interpretations of units, the IGDS reader and writer must be told how to interpret the feature coordinate units and how they will be converted to and from Units of Resolution (UORs). The feature coordinate units may be interpreted as Master Units, SubUnits, or as raw UORs, depending on the setting of `IGDS_UNITS` in the mapping file. However, when writing to DGN V8 files, the writer ignores these settings from the mapping file and adopts the settings as read from the seed file chosen. This means that if you want to do something special with the working units, you have to do that in the V8 seed file.

The IGDS reader and writer use symbolic names for the IGDS element types rather than the IGDS numeric values. This greatly simplifies element type specification. The following table maps the IGDS element type number to its corresponding FME feature `igds_type` attribute value that is used by the IGDS reader and writer. Subsequent subsections describe the handling of each of these element types in detail.

IGDS Element Type	FME <code>igds_type</code>
2	<code>igds_cell</code>
3	<code>igds_point</code>
3,4,12	<code>igds_line</code>
6,14	<code>igds_shape</code>

¹Throughout this chapter, the terms *IGDS file* and *Design file* are used interchangeably to refer to the ISFF format.

7	igds_text_node
11,12	igds_curve
12	igds_complex_string
14	igds_complex_shape
15	igds_ellipse
16	igds_arc
17	igds_text
7,17	igds_multi_text
2,6,14	igds_solid
34,35	igds_shared_cell
19	igds_3d_solid
100	igds_ref

The Reader/Writer has been enhanced to support enhanced geometry. When features are read/written using enhanced geometry then all the complex chains, complex shapes and solids (unnamed cells) will preserve arcs and ellipses within them.

Design File Quick Facts

Format Type Identifier	IGDS
Reader/Writer	Version 7 Both Version 8 Both
Licensing Level	Base
Dependencies	None
Dataset Type	File
Feature Type	Level number
Typical File Extensions	.dgn
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	Yes
Spatial Index	Never
Schema Required	No
Transaction Support	No
Enhanced Geometry	Yes
Geometry Type Attribute	igds_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	yes		polygon	yes
circular arc	yes		raster	no
donut polygon	yes		solid	yes
elliptical arc	yes		surface	no
ellipses	yes		text	yes
line	yes		z values	yes
none	no			

Reader Overview

The FME reader detects the version of the source dataset (version 7 or 8) internally and handles it accordingly. There is no difference to users in terms of the reader keyword or attribute names of the elements.

The IGDS reader first reads the header information from the Design file being processed, and extracts the conversion parameters required to translate coordinates from internal IGDS UORs to ground units. It also determines the dimension of the input file.

It then extracts each individual element, one at a time, and passes it on to the rest of the FME for processing. Complex elements are extracted as single FME features. If a complex element contains an arc, then the reader automatically converts it to a linestring enabling it to be processed by all other readers and writers within the FME. If the element had any attribute linkages attached to it, these are read and added as attributes to the FME feature being created.

When the IGDS reader encounters an element type it does not know how to process, it simply ignores it and moves on to read the next element.

DGN Version 8 also reads the models to which the features belong. All the models read retain their respective working units and global origin values.

Reader Directives

The IGDS reader processes the <ReaderKeyword>_DATASET directive in the mapping file. The value for this directive is the file name of the IGDS file to be read. By default, the <ReaderKeyword> for the IGDS reader is IGDS, so a typical mapping file fragment specifying an input IGDS file looks like:

```
IGDS_DATASET /usr/data/dgn/92b034.dgn
```

The IGDS reader also processes the <ReaderKeyword>_UNITS directive in the mapping file. This directive controls the conversion between UORs in the Design file and FME coordinates. There are three possibilities, outlined in the table below. If no UNITS directive is specified, then IGDS_SUB_UNITS is assumed.

IGDS_UNITS Value	Description
IGDS_MASTER_UNITS	The UORs read from the Design file are converted into master units , according to the conversion factor read from the Design file header, before being stored in an FME feature.
IGDS_SUB_UNITS	The UORs read from the Design file are converted into subunits , according to the conversion factor read from the Design file header, before being stored in an

IGDS_UNITS Value	Description
	FME feature. This is the default.
IGDS_UORS	The UORs read from the Design file are stored directly in an FME feature with no conversion.

The IGDS reader processes several other directives in the mapping file, as shown below. These enable the FME to override the Global Origin and Scaling information. The first four directives are normally used only when reading Design files that have bad header information. If the FME detects a difference between these settings and those read from the Design file, a warning is output to the log file and these settings prevail.

The IGDS reader can also be configured to output all the elements composing cells, or symbols. This is useful if the graphical representation of the Design file is to be preserved. This is true when, for example, a Design file is translated to a GIF image.

UOR_SCALE

Required/Optional: *Optional*

The number of ground units per UOR.

Workbench Parameter: *UOR to FME Feature Coordinate Units scale factor*

UOR_GLOBAL_ORIGIN_X

Required/Optional: *Optional*

The global origin of x measured in UORs.

Workbench Parameter: *UOR X Global Origin*

UOR_GLOBAL_ORIGIN_Y

Required/Optional: *Optional*

The global origin of y measured in UORs.

Workbench Parameter: *UOR Y Global Origin*

UOR_GLOBAL_ORIGIN_Z

Required/Optional: *Optional*

The global origin of z measured in UORs.

Workbench Parameter: *UOR Z Global Origin*

SUBS_PER_MASTER

Required/Optional: *Optional*

The number of sub units per master unit. This is only used if UOR_SCALE is not present.

Workbench Parameter: *SUBS PER MASTER UNIT*

UORS_PER_SUB

Required/Optional: *Optional*

The number of UORs per sub unit. This is only used if UOR_SCALE is not present.

Workbench Parameter: *UORS PER SUB UNIT*

EXPAND_CELLS

Required/Optional: *Optional*

Controls whether or not all components of a cell will be output by the reader.

If the value is YES, then they are and the cell header itself is not output.

If it is NO, then only the cell header is output.

Values: *YES | NO*

Default Value: *NO*

Workbench Parameter: *Expand Named Cells*

EXPAND_UNNAMED_CELLS

Required/Optional: *Optional*

This directive should not be confused with EXPAND_CELL in terms of its usage. It is better understood in relation to igds_solid. When it is set to YES, then no donuts are formed even if they existed and the cell members retain their colors. When it is set to NO, then donuts will be formed if they existed, and the pieces may lose their original colors.

Values: *YES | NO*

Default Value: *NO*

Workbench Parameter: *Expand Unnamed Cells*

PRESERVE_CELL_INSERTS

When EXPAND_CELLS is set to YES, this directive controls whether or not the insert points of the cells are also output.

Values

YES: The cell insert points are output as igds_cell features in addition to the cell components

NO (default): Only the cell components are output.

Required/Optional

Optional

*** Workbench Parameter**

Preserve Named Cell Inserts

PRESERVE_UNNAMEDCELL_INSERTS

Required/Optional: *Optional*

If the value is YES, then the cell insert points are output in addition to the cell components.

If it is NO, then only the cell components are output.

Values: *YES | NO*

Default Value: *NO*

Workbench Parameter: *Preserve Unnamed Cell Insert Points*

PROPAGATE_CHAIN_ELEMENT_LINKAGES (applicable only with classic geometry)

Required/Optional: *Optional*

Controls how the linkages attached to complex chain element features are handled.

If the value is YES, then the linkages attached to the first component of the complex chain are returned on the FME feature, supplementing any existing linkages.

If it is NO, then any linkages on the component elements themselves will be lost and only those linkages attached to the complex chain itself will be returned.

Values: YES | NO

Default Value: NO

Workbench Parameter: *<WorkbenchParameter>*

SPLIT_COMPLEX_CHAINS (applicable only with classic geometry)

Required/Optional: *Optional*

Controls whether or not complex chain elements are returned, merged as a single linear feature or as their component parts.

If SPLIT_COMPLEX_CHAINS is YES, then FME adds the attribute `igds_chain_number` which is added to each element of a chain split. If desired, this can later be used to aggregate chain elements.

If the value is YES, then each component of a complex chain will be returned as its own feature and no feature will be returned for the complex chain as a whole. This is equivalent to dropping the complex chain in MicroStation. If the header had any linkage attributes, these will be propagated to the component elements.

If the value is NO, then all elements of the complex chain will be merged into a single linear feature, any arcs in the complex chain will be converted into linestrings and any linkages on the component elements themselves will be lost.

Values: YES | NO

Default Value: NO

Workbench Parameter: *<WorkbenchParameter>*

AGGREGATE_COMPLEX_CHAINS (applicable only with classic geometry)

Controls whether or not complex chain element features are returned as aggregates.

If the value is YES then the individual element properties are held in the `igds_complex_elements{}` list, and the `igds_type` is set to `igds_complex_string` or `igds_complex_shape`.

If specified, this setting takes precedence over SPLIT_COMPLEX_CHAINS. In other words, if the value of AGGREGATE_COMPLEX_CHAINS is YES, any value specified for SPLIT_COMPLEX_CHAINS is ignored.

Required/Optional

Optional

Values

YES | NO (default)

TAGS_AS_TEXT

Controls whether or not visible tag data elements are output as separate text elements, in addition to having their data attached to the primary graphic element they go with.

Required/Optional

Optional

Values

YES | NO (visible tag data elements are not output as text elements)

*** Workbench Parameter**

Output Tags as Text

PRESERVE_CURVES

Controls whether or not curve elements will be stroked into lines by adding vertices.

Required/Optional

Optional

Values

YES | NO (curves are not preserved and are stroked into lines)

*** Workbench Parameter**

Preserve Curves

ELEVATION_SHIFT_FACTOR

If an elevation shift is desired to build "fake" 3D topology, this is the scaling factor used to generate the shift. Specifically, the Z value is divided by this factor and the result is added to the X value.

Required/Optional

Optional

*** Workbench Parameter**

Elevation Shift Factor

CURVE_VERTICES

This directive is used only when **PRESERVE_CURVES** is set to **NO**. It controls the number of interpolated points per segment when the curve is stroked into a line.

Required/Optional

Optional

Default Value

5

*** Workbench Parameter**

Number of interpolated curve vertices

TRIM_DOWN_TAGS

Removes the tag attributes when set to YES.

Required/Optional

Optional

Values

YES | NO (default)

*** Workbench Parameter**

Strip Off Tag Information

SPLIT_MULTITEXT

When set to YES, the reader splits the multi-text into text nodes and outputs the member text elements as individual text elements.

When set to NO, the text elements are not split.

Required/Optional

Optional

Values

YES (default) | NO

*** Workbench Parameter**

Split multi text

READ_BYTE_OFFSET

Required/Optional: *Optional*

Version: *supported for version 7 only*

If set to YES, adds the `igds_element_byteoffset` attribute (which contains the position of the element in the .dgn file) to the feature. Note, however, that turning this option on might significantly slow down reading on some platforms like UNIX.

Values: *YES | NO*

Default Value: *NO*

Workbench Parameter: *<WorkbenchParameter>*

EXPLODE_DIMENSION_ELEM

Required/Optional: *Optional*

If set to YES, explodes the dimension element into its pieces. If set to NO, then imports the dimension element as an aggregate. When importing as an aggregate, the text members are not output as features but are stored as list attributes of the dimension, and the arc members are stroked.

Values: YES | NO

Default Value: YES

Workbench Parameter: <WorkbenchParameter>

READ_XREF_FILES

If set to YES, reads all the supported elements in the external reference files attached to the source dataset. If the reference file has nested references, they are also imported. Note that this directive does not affect the reading of reference file elements as features with an igds_type of igds_xref.

Required/Optional

Optional

Values

YES | NO (default)

✳ Workbench Parameter

Read Reference Files

READ_XREF_UPTO_FIRST_LVL

If set to YES, reads all the supported elements in the external reference files attached to the source data set up to the first level of nesting only. Note that this directive does not affect the reading of reference file elements as features with an igds_type of igds_xref.

This directive is valid only if READ_XREF_FILES is set to YES.

Required/Optional

Optional

Values

YES | NO (default)

✳ Workbench Parameter

Read Reference up to First Level

USE_XREF_PARENT_MODEL

If set to YES, uses the model of the parent file of the xref file. Note that this directive does not affect the reading of reference file elements as features with an igds_type of igds_xref.

This directive is applicable to version 8 only, since models are supported in version 8 but not in version 7.

Required/Optional

Optional

Values

YES (default) | NO

*** Workbench Parameter**

Use Reference's Parent Model (V8 only)

EXPLODE_MULTI_LINE

Required/Optional: *Optional*

If set to yes, then multilines are exploded into its pieces.

Values: *YES | NO*

Default Value: *NO*

READ_DELETED_ELEMENTS

Required/Optional: *Optional*

This directive is used to read deleted elements.

Note: This directive will not be made available in Workbench Format Parameters. To use this directive, it has to be set to TRUE in the mapping file.

Values: *TRUE | FALSE*

Default Value: *FALSE*

APPLY_WORLD_FILE

Use this directive when you have an ESRI World file (*.wld) that you want FME to use when determining the coordinates for features in your dataset.

When this directive has a value of YES, FME will search the directory of the dataset for a file with the same name as your dataset but with a .wld extension. If it cannot find a file with that name, it will then look for the file "esri_cad.wld" within the dataset directory.

If either of those files exist, FME will use the information in the files to translate the coordinates of the features in the dataset to their new geospatial coordinates.

If the files cannot be found, then the translation will continue, using the coordinate information found in the dataset, without performing any additional transformation.

Required/Optional

Optional

Values

- YES (Workbench default)
- NO (mapping file default)

*** Workbench Parameter**

Apply World File (.wld)

REMOVE_DUPLICATES (applies to classic geometry only)

Set this directive to Yes when it is intended to remove the duplicate points (same x and y coordinates) from the geometry of the feature.

Required/Optional

Optional

Values

- YES (mapping file default)
- NO (Workbench default)

* Workbench Parameter

Remove Duplicate Points

Reader Directives for FME Objects

SCHEMA_INCLUDE_MSLINKS

Required/Optional: *Optional*

This directive can be used for FME Objects only. When set to YES, schema for MSLINKS are added to the feature.

Values: YES | NO

Default Value: NO

Reader Directives for FME Objects

SCHEMA_INCLUDE_MSLINKS

Required/Optional: *Optional*

This directive can be used for FME Objects only. When set to YES, schema for MSLINKS are added to the feature.

Values: YES | NO

Default Value: NO

SCHEMA_INCLUDE_FRAMME

Required/Optional: *Optional*

This directive can be used for FME Objects only. When set to YES, schema for FRAMME linkages are added to the feature.

Values: YES | NO

Default Value: NO

Writer Overview

To create a new Design file, header information is obtained from an existing Design file, called a *seed file*. The IGDS writer first copies the seed file's header information to the destination file, and then extracts the conversion parameters required to translate coordinates from feature coordinate units to internal IGDS UORs¹. This header information

¹Since coordinates in Design files are ultimately stored as integer UORs, it is possible for precision to be lost or overflow to occur when they are output. Care must be taken to ensure that the conversion parameters in the seed file preserve the data precision and range.

includes type 68 FRAMME elements for V7 only, and type 100 external file reference elements. The IGDS writer uses the seed file to determine whether the destination file will be two-dimensional or three-dimensional.

Because seed files with a sufficient ground range and resolution may be difficult to obtain, the IGDS V7 writer allows seed parameters to be overridden in the mapping file. When a seed file with insufficient range available is used, the IGDS V7 writer will report that features were outside of the bounds of the seed file, and suggest values for the global origin and UOR/subunit/master unit ratios to use. The FME can also automatically adjust the V7 Design file by setting the COMPUTE_SEED_FILE_PARAMS flag to yes. Note that this facility has been taken away from the V8 writer – it is no longer necessary since V8 has a much larger design plane than V7.

Note: When translating from DGN version 8 to DGN version 7 or vice versa in FME Workbench, by default a v8 seed file is chosen from the set of seed files as provided by FME. This has to be changed to an appropriate version 7 or version 8 seed file in order to achieve a successful conversion. The seed file is used to determine which version the user intends to write. Also note that if the user picked a v7 seed file at the time of generating the workspace, the same workspace can be used to write to v7 or v8 by changing the seed file accordingly. But if a workspace was initially generated to write to v8, then it cannot be used to write to v7.

A cell library file may optionally be used by both V7 and V8 writer. Cell libraries contain named symbol definitions which can be used to depict point features. If a cell library is specified, the IGDS writer reads in all the cell definitions for later when cell features are output. The IGDS writer can use either 2- or 3-dimensional cell libraries, and will automatically convert the cell definitions to be of the correct dimension for output.

The IGDS writer then outputs each FME feature it is given. Most often, a single FME feature corresponds to a single IGDS element. If any linkages are specified for the element, they are also output. However, some of the IGDS element types cause several elements to be output as a complex unit, with the complex bit turned on. This occurs when a multi-line text object, a cell, or a closed shape or linear feature with more than 101 coordinates (5000 coordinates in V8) is output. The IGDS writer hides all of the details of complex element output.

The IGDS writer can be configured to do one of two things with linear features that have exactly two points. By default, a type 4 linestring will be created for such features. However, if IGDS_CREATE_LINE_ELEMENTS is set to yes in the mapping file, then a type 3 line element will be created for the two-point linear feature.

Note: Design files (V7) can be a maximum of 32 MB in size. Files larger than this will not be completely read by Microstation. The IGDS writer will automatically split any design file it is writing into pieces to avoid overrunning this maximum size. When this happens, features that would have caused the size limit to be exceeded are written to additional design files as necessary. The additional files are named <basename>_#.dgn, where # starts at 1 and increases.

Writer Directives

By default, the <WriterKeyword> for the IGDS writer is IGDS, so a typical mapping file fragment configuring the IGDS writer would be:

```
IGDS_DATASET /usr/data/dgn/92b034.dgn
IGDS_SEED_FILE /usr/data/dgn/2dseed.dgn
IGDS_CELL_LIBRARY /usr/data/dgn/cartog.cel
```

DATASET

The file name of the output IGDS file.

Required/Optional

Required

* Workbench Parameter

Destination Bentley MicroStation Design File

SEED_FILE

The file name of the Design file which will be used to seed the output file's header information. The default seed file (V8) is:

`$(FME_HOME)/design/seed3d_m_v8.dgn`

To write to V7, you will have to select a valid V7 seed file.

It is important to note that the seed file determines which destination version to write.

Required/Optional

Required

*** Workbench Parameter**

V7/V8 Seed File

ALLOW_FILL

Controls whether or not fill linkages will be written out for ellipses, shapes, and solids. This setting does not affect the usable

Required/Optional

Optional

Values

YES (default) | NO

*** Workbench Parameter**

Allow Area Color Fills

CELL_LIBRARY

The file name of an IGDS cell library that contains the definitions of cells which may later be output.

Required/Optional

Optional

*** Workbench Parameter**

Cell Library File

DEFAULT_CELL_NAME

The default cell used in place of any cells requested but not found in the cell library.

Required/Optional

Optional

Version

This directive is not currently supported by the V8 writer.

*** Workbench Parameter**

Default Cell Name

UNITS

Specifies how FME feature coordinates will be interpreted and converted into UORs.

See the [Reader Overview](#) for details.

Version

This directive is not currently supported by the V8 writer.

Required/Optional

Optional

*** Workbench Parameter**

Output Units

CREATE_LINE_ELEMENTS

Controls whether or not type 3 line elements will be created for two point linear features.

Values

YES | NO (default)

If set to NO, then type 4 elements will be created.

Required/Optional

Optional

*** Workbench Parameter**

Type 3 Elements

COMPUTE_SEED_FILE_PARMS

Automatically adjusts the origin and scaling of the seed file to provide an optimum set of parameters for the input data.

Version

This directive is not currently supported by the V8 writer. It is ignored if chosen with a V8 seed file.

Required/Optional

Optional

*** Workbench Parameter**

Compute Optimal Seed File Parameters

UOR_GLOBAL_ORIGIN_X

The global origin of x, measured in UORs. Overrides that read from the seed file.

Version

This directive is not currently supported by the V8 writer.

Required/Optional

Optional

*** Workbench Parameter**

UOR X Global Origin

UOR_GLOBAL_ORIGIN_Y

The global origin of y, measured in UORs. Overrides that read from the seed file.

Version

This directive is not currently supported by the V8 writer.

Required/Optional

Optional

*** Workbench Parameter**

UOR Y Global Origin

UOR_GLOBAL_ORIGIN_Z

The global origin of z, measured in UORs. Overrides that read from the seed file.

Version

This directive is not currently supported by the V8 writer.

Required/Optional

Optional

*** Workbench Parameter**

UOR Z Global Origin

MASTER_UNIT_NAME

The two-character master unit name to use. Overrides that read from the seed file.

Version

This directive is not currently supported by the V8 writer.

Required/Optional

Optional

*** Workbench Parameter**

Master Unit Name

SUB_UNIT_NAME

The two-character sub unit name to use. Overrides that read from the seed file.

Version

This directive is not currently supported by the V8 writer.

Required/Optional

Optional

*** Workbench Parameter**

Sub Unit Name

SUBS_PER_MASTER

The number of sub units per master unit. Overrides that read from the seed file.

Version

This directive is not currently supported by the V8 writer.

Required/Optional

Optional

*** Workbench Parameter**

Subs per Master

UORS_PER_SUB

The number of UORs per sub unit. Overrides that read from the seed file.

Version

This directive is not currently supported by the V8 writer.

Required/Optional

Optional

*** Workbench Parameter**

UOR per Sub

MANGLE_DBCS_TEXT

Controls whether or not double-byte-character-set text is *mangled* when text strings are written.

MicroStation uses special header bytes to single DBCS text. Note that the IGDS reader automatically de-mangles DBCS text.

Values

YES | NO (default)

If this directive is set to Yes in the mapping file, then these special bytes will be output when a DBCS text string is encountered. The default value is No.

Version

This directive is not currently supported by the V8 writer.

Required/Optional

Optional

*** Workbench Parameter**

Mangle DBCS Text

SPLIT_BIG_DGN7_FILES

Allows user to split Version 7 DGN files bigger than 32 MB.

Note that this directive can be manually set to No in the mapping file.

Values

YES (default) | NO

Version

This directive applies to the V7 writer only.

Required/Optional

Optional

*** Workbench Parameter**

Split Files (V7 Only)

SPLIT_SIZE_DGN7_FILES

This directive allows you to set the size of the output file, in MB. It is applicable only if **SPLIT_BIG_DGN7_FILES** is set to YES.

Values

Default value: 32 MB

Version

This directive applies to the V7 writer only.

Required/Optional

Optional

*** Workbench Parameter**

Split Size in MB (V7 Only)

WRITE_TAGS

Controls whether or not tags should be written for the elements which have necessary tag information attached to them as attributes.

Values

YES | NO (default)

Version

This directive currently applies only to the V8 writer.

Required/Optional

Optional

*** Workbench Parameter**

Write Tags

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see About Feature Attributes), this format adds the format-specific attributes described in this section.

Special FME feature attributes are used to hold IGDS element parameters. The IGDS writer will use these attribute values as it fills in an element structure during output. The IGDS reader will set these attributes in the FME feature it creates for each element it reads.

The FME considers the IGDS level to be the *FME feature type* of an IGDS feature. Each IGDS element, regardless of its geometry type, shares a number of other parameters, as described in the following table. Subsequent subsections describe parameters specific to each of the supported element types.

When writing elements, `igds_type` has precedence over the `igds_element_type`, unless there is more than one element type for a given type. For example, for `igds_line` the `igds_element_type` can be used to force the element to be a type 4 line element, even if there are only 2 vertices on the line (that is, by rights it should be a type 3 element).

Attribute Name	Contents
igds_basename	The base filename (without extension) of the design file the elements were read from. This attribute is ignored by the writer. Range: ASCII filename
igds_color	The element's color setting. This is the element's color index into the color table stored in the design file. This attribute will be overridden by the <code>igds_symbology</code> value. Range: 0..255 Default: 0
igds_color.red <i>Reader only</i>	The element's red color intensity, as determined by looking up the element's color index in the color table. Range: 0..255
igds_color.green <i>Reader only</i>	The element's green color intensity, as determined by looking up the element's color index in the color table. Range: 0..255
igds_color.blue <i>Reader only</i>	The element's blue color intensity, as determined by looking up the element's color index in the color table. Range: 0..255
igds_class	The element's class. Range: 0..15 Default: 0
igds_element_type	The numeric Design file element type code of the element. When writing to a Design file, the <code>igds_type</code> field overrides this attribute. This attribute will be overridden by the <code>igds_type</code> value. Range: See the <i>Overview</i> subsection. Default: No default
igds_graphic_group	The element's graphic group number. Range: 0..65535 Default: 0 Tip: By using a common value for graphic group value, several otherwise separate elements may be tied together into a <i>logical</i> super-element for later processing by application programs.

Attribute Name	Contents
igds_hole <i>Writer only</i>	If present, it sets the "hole" bit on the element it is creating. Range: string Default: No default
igds_level	The IGDS level of the feature. The value of this attribute is the same as the feature type. The Writer will use the value of this attribute if the feature's type cannot be converted into a valid IGDS level. Range: 0..64 (There is no upper limit on levels for Version 8 DGN files.) Default: No default
igds_level_comment <i>Reader only</i>	The comment associated with the level from which the element originated. Range: String Default: No default
igds_level_group_id <i>Reader only</i>	The group identification of the level from which the element originated. (Does not exist for Version 8 DGN files.) Range: String Default: No default
igds_level_name	During reading this represents the name of the level from which the element originated. Note: For writing to Version 8 DGN files only, this may be used instead of the feature_type to set the level name. Range: String Default: No default
igds_color_set_bylevel	Set to yes if the element's color is set by level; otherwise it is set to no. If it is set to yes, the writer sets the element's property to pick the color from the level it is on. (Note: Applies to Version 8 DGN files only.) Range: yes/no Default: No default
igds_style_set_bylevel	Set to yes if the element's style is set by level; otherwise it is set to no. If it is set to yes, the writer sets the element's property to pick the style from the level it is on. (Note: Applies to Version 8 DGN files only.) Range: yes/no Default: No default
igds_weight_set_bylevel Note: Applies to Version 8 DGN files <i>(Reader only)</i>	Set to yes if the element's weight is set by level; otherwise it is set to no. If it is set to yes, the writer sets the element's property to pick the weight from the level it is on. Range: yes/no Default: No default

Attribute Name	Contents
igds_snappable	<p>The element's snappability.</p> <p>Range: yes or no</p> <p>Default: yes</p>
igds_style	<p>The element's line style. This attribute will be overridden by the igds_symbology value.</p> <p>Range: 0..7</p> <p>Default: 0</p>
igds_style_name Note: <i>Applies to Version 8 DGN files (Reader only)</i>	<p>The name of the element's line style. This attribute is used by the reader to provide the name of the style used in the igds_style attribute.</p> <p>Range: String</p> <p>Default: No default</p>
igds_symbology	<p>A single integer encoding the element's style, weight, and color according to this formula:</p> $\text{symbology} = \text{style} + 8 * \text{weight} + 256 * \text{color}$ <p>This attribute will override the individual settings for style, weight, and type if it is specified.</p> <p>Range: 0..65536</p> <p>Default: None</p>
igds_type	<p>The FME name for the type of element this feature represents.</p> <p>Range: See the table in the Overview subsection.</p> <p>Default: No default</p>
igds_weight	<p>The element's line weight. This attribute will be overridden by the igds_symbology value.</p> <p>Range: 0..31</p> <p>Default: 0</p>
igds_xlow	<p>The element's minimum X value in ground units. The value of this attribute is ignored when writing.</p> <p>Range: Any Number</p>
igds_xhigh	<p>The element's maximum X value in ground units. The value of this attribute is ignored when writing.</p> <p>Range: Any Number</p>
igds_ylow	<p>The element's minimum Y value in ground units. The value of this attribute is ignored when writing.</p> <p>Range: Any Number</p>
igds_yhigh	<p>The element's maximum Y value in ground units. The value of this attribute is ignored when writing.</p> <p>Range: Any Number</p>
igds_zlow	<p>The element's minimum Z (elevation) value in ground units. The value of this attribute is ignored when writing 3D files to V7, and is ignored</p>

Attribute Name	Contents
	<p>when writing to V8.</p> <p>Range: Any Number</p> <p>Default: No default</p>
igds_zlow_uor	<p>The element's minimum Z (elevation) value in UORs. The value of this attribute takes precedence over igds_zlow when the feature is written. The value of this attribute is ignored when writing 3D files to V7, and is ignored when writing to V8.</p> <p>Range: Any Number</p> <p>Default: No default</p>
igds_zhigh	<p>The element's maximum Z (elevation) value in ground units. The value of this attribute is ignored when writing 3D files to V7, and is ignored when writing to V8.</p> <p>Range: Any Number</p> <p>Default: No default</p>
igds_zhigh_uor	<p>The element's maximum Z (elevation) value in UORs. The value of this attribute takes precedence over igds_zhigh when the feature is written. The value of this attribute is ignored when writing 3D files to V7, and is ignored when writing to V8.</p> <p>Range: Any Number</p> <p>Default: No default</p>
igds_custom_linestyle	<p>If an element has a custom line style, then this attribute will contain the name of the custom line style. It does not appear as part of the attributes of the element in case it does not have any custom line styles defined for it.</p> <p>Range: String</p> <p>Default: No default</p>
igds_custom_linestyle_rbit	<p>This is used to write the custom line styles. This value sets the rbit of the user linkage.</p> <p>Range: 0 or 1</p> <p>Default: 0</p>
igds_custom_linestyle_mbit	<p>This is used to write the custom line styles. This value sets the mbit of the user linkage.</p> <p>Range: 0 or 1</p> <p>Default: 0</p>
igds_custom_linestyle_ibit	<p>This is used to write the custom line styles. This value sets the ibit of the user linkage.</p> <p>Range: 0 or 1</p> <p>Default: 0</p>
igds_custom_linestyle_class	<p>This is used to write the custom line styles. This value sets the class of the user linkage.</p> <p>Range: 0 or 1</p>

Attribute Name	Contents
	Default: 0
igds_element_byteoffset	This is used to tell the position of the element. Range: Any Number Default: No default
igds_model_name Note: <i>Applies to Version 8 DGN files.</i>	The name of the model to which the feature belongs. Range: String Default: No default
igds_model_id Note: <i>Applies to Version 8 DGN files.</i>	The ID of the model to which the feature belongs. Range: Any positive integer Default: No default
igds_element_new	The NEW property of the element. Range: YES or NO Default: No default
igds_element_modified	The MODIFIED property of the element. Range: YES or NO Default: No default
igds_date_last_modified Note: <i>Applies to Version 8 DGN files (Reader only)</i>	Stores the date the element of last modified in the format YYYYMMDD hh:mm:ssAM/PM. Default: No default
igds_element_locked	The LOCKED property of the element. Range: YES or NO Default: No default
igds_element_id	The unique ID of each element in a DGN file. Note: Applies to Version 8 DGN files. Range: Any positive integer Default: No default
mslink_x	Value of mslink key of the corresponding linkage, where x is the linkage number starting with 0. Default: No default
entity_num_x	Value of entity_number of the corresponding linkage where x is the linkage number starting with 0. Default: No default
link_type_x	Value of link type of the corresponding linkage, where x is the linkage number starting with 0. Default: No default
igds_element_association_id	The tags store this ID as the element ID it is attached to.

Attribute Name	Contents
igds_z_value	This attribute is for the writer only and should be used only when 3D is intended to be forced. Default: 0
igds_chain_number	If SPLIT_COMPLEX_CHAINS is YES, then FME adds the attribute igds_chain_number which is added to each element of a chain split. Default: No default
igds_deleted	This attribute is set to yes only when the element read was a deleted element. Default: No default
igds_element_visibility Note: Applies to Version 8 DGN files.	This attribute has the value yes if the level the element is on has its display property set to "on"; otherwise, the value is no. Default: No default
igds_element_view_independent Note: Applies to Version 8 DGN files.	This attribute has the value yes if the element is view-independent; otherwise, the value is no. Default: yes
igds_is_graphic_cell_relative	Note: This attribute is for graphic cells only. It is ignored for point cells and shared cells. If this attribute is set to Yes, then the graphic cell is written as relative graphic cell. This means that the cell member with the lowest level number will be put on the current (feature's) level. All the subsequent ones are offset accordingly. For example, if a cell had members on level 4, 6 and 7 respectively and we are writing this cell feature on level 2, then the member with level 4 gets written on level 2. The members with level 6 and 7 are written on level 4 and 5, respectively. This also applies to members of nested cells. Note that all the offset levels should be provided in the seed file, otherwise the cell would be skipped. Note: When writing to Version 8 DGN files only, if the igds_level is not supplied, the level name will be used to look up the level number in the seed file. Default: No

Attribute Linkages

Each element in an IGDS file may have one or more attribute linkages attached to it. The IGDS reader and writer support both user data and database attribute linkages.

Note, however, that the DGN V8 reader and writer do not support the interpretation of user linkages. Database linkages and MSLINKS are supported. FRAMME linkages are supported for reading.

Because an element may have more than one linkage, linkages are stored in an FME feature attribute list named igds_linkage{#}. As with other feature attribute lists, # starts at zero and increments for each successive linkage.

Currently, only database and DMRS linkages are supported for reading and writing of Version 8 DGN files. However, FRAMME linkages are supported when reading Version 8 DGN files.

Attribute Lists – all linkages

The following attribute list item names are used by all linkages. Note that the class and various bit fields are not used when the linkage type is dmrs.

Linkage Parameter	Contents
type	The type of linkage. Range: user dbase odbc oracle informix ris dmrs framme Default: No default Note: User and FRAMME linkages are not currently supported for Version 8 DGN files.
class	Linkage Class. Range: 0..15 Default: 0
ibit	Linkage <i>ibit</i> value. This bit represents whether the linkage is informational or non-informational. Range: 0 1 Default: 0
mbit	Linkage <i>mbit</i> value. Indicates linkage has been modified. Range: 0 1 Default: 0
rbit	Linkage <i>rbit</i> value. The bit is set for remote linkages. Range: 0 1 Default: 0
ubit	Linkage <i>ubit</i> value. Range: 0 1 If set to 1 then linkage is user data linkage; if set to 0, then the linkage type is always dmrs. Default: 1

Attribute Lists – user linkage

If the linkage is of type user (Version 8 DGN files support user linkages having a *userId* of 2570, 22244, 32000, 32001, or 39030 only), then these attribute list item names are used to specify the values for the user linkage:

Linkage Parameter	Contents
userId	The user ID of the linkage. This is application-specific. Range: 0..65535 Default: No default
long{#}	The user data associated with a user linkage may be specified as a list of 32-bit long integers or as a list of 16-bit

Linkage Parameter	Contents
	words. If 32-bit long integers are used to fill out the attribute linkage, they have this suffix and are numbered sequentially starting from 0. Range: 32-bit integer Default: 0
word{#}	If 16-bit words are used to fill out the attribute linkage, they have this suffix and are numbered sequentially starting from 0. Range: 0..65535 Default: 0

User linkages with a userId of 2570, 22234, 32000, 32001, or 39030 (Extended entity data linkage)

In V7 these linkages are supported like any other user linkages, but in V8 FME stores them as a blob and that gets carried over to V8 as a blob.

Note that in order to get Extended Entity Data linkages to carry over correctly, the original file containing these linkages should be picked as the seed file. The support for these linkages would work for V8 to V8 only. Any attempt to transfer them from V7 to V8 (or vice versa) will not work. In V8, they store linkage attributes as follows:

Linkage Parameter	Contents
userId	The user -ID of the linkage. This value would be any of 2570, 22244, 32000, 32001, or 39030. Default: No default
blob	This stores the linkage as binary data. Default: No default
blobsize	Stores the size of the blob. Range: 0..256 Default: No default
flags	Flags for the user linkage. Range: 0..256 Default: No default
type	Type of linkage. Default: user

Attribute Lists – dbase, odbc, oracle, ris, dmrs, informix linkages

If the linkage is of type dbase, odbc, oracle, ris, dmrs, or informix, then these attribute list item names are used to specify the values for the database linkage.

Linkage Parameter	Contents
entity_number	The entity number of the linkage. Range: 0..65535 Default: 1
key	The key value of the database linkage. This value corresponds to the value in a field in the attribute row associated with the element in the database. Range: 32-bit integer for 8 word linkage formats (i.e., Oracle, ODBC) and 24-bit integer for 4 word linkage formats (i.e., DMRS) Default: No default
readonly	This applies to the dmrs linkages and indicates whether or not the linkage is readonly. MGE systems also use this to differentiate between feature (which are readonly) and attribute (which are not) linkages. Range: yes no Default: yes
trailing_flags	The trailing flags of the database linkage. This can be used to set the "daskey". (Not supported for Version 8 DGN files.) Range: signed 32-bit integer Default: 0
firstword	This is the actual value of the first word of the dmrs linkage. It is stored for dmrs linkages only. Range: Unsigned 16-bit integer Default: 0
ltype	This attribute is used internally by the V8 writer and is not intended for users.
key2	This attribute is used internally by the V8 writer and is not intended for users.
suspectlinkage	This attribute is stored only if the reader detected that a certain linkage had an odd number of words, rather than an even number.

Attribute Lists – framme linkage

If the linkage is a framme type, the following attribute list item names are used to specify the values for the Facilities Rulebase Application Model Management Environment (FRAMME) linkage.

Familiarity with the FRAMME system is necessary to fully understand the meaning of these attributes. Note that FRAMME linkages are only supported when reading Version 8 DGN files.

Linkage Parameter	Contents
ufid	<p>The unique feature ID of the linkage. This is part of the database key used by FRAMME.</p> <p>Range: unsigned 32-bit integer</p> <p>Default: 0</p>
ufid_low	<p>The high low order bytes of the unique feature ID of the linkage. This is part of the database key used by FRAMME.</p> <p>Range: unsigned 16-bit integer</p> <p>Default: 0</p>
ufid_high	<p>The low order bytes of the unique feature ID of the linkage. This is part of the database key used by FRAMME.</p> <p>Range: unsigned 16-bit integer</p> <p>Default: 0</p>
design_file	<p>The base name of the design file holding the linkage. This makes up the second part of the database key used by FRAMME.</p> <p>Range: character string</p> <p>Default: No default – not used when writing</p>
state_num	<p>The state number of the FRAMME feature</p> <p>Range: unsigned 16-bit integer</p> <p>Default: 0</p>
rule_base_id	<p>The FRAMME rule base identifier which is fixed at 0x20.</p> <p>Range: 0x20 (32 decimal)</p> <p>Default: 0x20</p>
component_num	<p>The component number of the FRAMME feature.</p> <p>Range: unsigned 16-bit integer</p> <p>Default: 0</p>
component_count	<p>The component count, or occurrence, of the FRAMME feature.</p> <p>Range: unsigned 16-bit integer</p> <p>Default: 0</p>
feature_num	<p>The feature number of the FRAMME feature.</p> <p>Range: unsigned 16-bit integer</p> <p>Default: 0</p>
long{#}	<p>A list of 16-bit words that associated with "long" FRAMME linkages.</p> <p>Range: unsigned 16-bit integer</p> <p>Default: 0</p>

For convenience, some of the list item names above are provided as non-list attributes which represent only the first framme linkage found per element. These attributes are listed below.

Linkage Parameter	Contents
ufid	The unique feature ID of the linkage. This is part of the database key used by FRAMME. Range: unsigned 32-bit integer Default: 0
ufid_low	The high low order bytes of the unique feature ID of the linkage. This is part of the database key used by FRAMME. Range: unsigned 16-bit integer Default: 0
ufid_high	The low order bytes of the unique feature ID of the linkage. This is part of the database key used by FRAMME. Range: unsigned 16-bit integer Default: 0
dgnfile	The base name of the design file holding the linkage. This makes up the second part of the database key used by FRAMME. Range: character string Default: No default – not used when writing
state_num	The state number of the FRAMME feature Range: unsigned 16-bit integer Default: 0
comp_num	The component number of the FRAMME feature. Range: unsigned 16-bit integer Default: 0
comp_count	The component count, or occurrence, of the FRAMME feature. Range: unsigned 16-bit integer Default: 0
feat_num	The feature number of the FRAMME feature. Range: unsigned 16-bit integer Default: 0

Attribute Lists – incosada linkage

If the linkage is an incosada type, then the following attribute names are added by the reader to hold the values for the British Columbia Forestry File (INCOSADA) linkage. Note that INCOSADA linkages are not supported in Version 8 DGN files, nor are they supported by the Design file writer.

Linkage Parameter	Contents
incosada_fid	The unique feature ID of the linkage. Range: Character string of size 32 consisting of hex digits

Linkage Parameter	Contents
incosada_sequence_num	Sequence number of the linkage. Range: integer
incosada_feature_code	The feature code of the INCOSADA feature Range: unsigned 32-bit integer

Example

For example, the FME feature specified by the partial transfer specification below would have two linkages. The first linkage is a user linkage which specifies that the shape is to be filled with color 12, and the second linkage is a dBASE linkage which links the element to the record with the key value of 1001. Note that if the same feature were to have an `igds_fill_color` attribute, its value would override the color specified in any solid fill.

```
MACRO fillUserId 65
MACRO fillMagic 67586
IGDS 32 igds_type igds_shape \
    igds_color 8 igds_weight 1 \
    igds_linkage{0}.type user \
    igds_linkage{0}.userId $(fillUserId) \
    igds_linkage{0}.long{0} $(fillMagic) \
    igds_linkage{0}.long{1} 12 \
    igds_linkage{1}.type dbase \
    igds_linkage{1}.key 1001
```

Custom Line Styles

The custom line styles are stored as linkages of the element.

In V7, each custom line style name has an ID, which is a negative integer, and is stored as `igds_linkage{n}.long{0}`. In V8, each custom line style name has an ID, which is a negative integer, and is stored as `igds_style`. (Note that there are some limitations to V8 custom line style support.)

It is fairly simple to write custom line styles while translating from `dgn` to `dgn`. However, it is the user's responsibility to provide the correct seed file containing the definitions of the custom line styles and to copy the `.rsc` file into the directory containing other Microstation resource files. Here are the steps to configure Microstation for Custom Line Styles:

1. Open the seed file or destination file.
2. Select Workspace > Configuration > Symbology.
3. Click Select.
4. Select the `.rsc` file that you want to use.
5. Click Add to add the `.rsc` file to the list.
6. Click OK and then Done.
7. Close the file and then reopen it. It is important close the file; otherwise, the changes in the configuration just made are not reflected.
8. From the Active Line Style pull-down menu, select Custom. Then select the name of line style that you want to use, and double click to activate it.
9. Select File > SaveSettings.

It is also possible to write a new custom line style when writing to a `dgn` file in cases where it was not originally provided in the source dataset. The user has to provide the value of `igds_custom_linestyle` whereas it is optional to provide the values for `igds_custom_linestyle_rbit`, `igds_custom_linestyle_mbit`, `igds_custom_linestyle_ibit` and `igds_custom_linestyle_class`. If they are not provided, the writer uses the default values.

Note: When translating from DGN to DGN where a complex chain in the source data set has custom line styles, then you have to set the keyword **PROPAGATE_CHAIN_ELEMENT_LINKAGES** to **true** to translate the custom line styles properly to the destination format.

If a complex chain has different line styles, then in order to retain those line styles, set "Drop Complex Chain" to yes.

Arcs

igds_type: igds_arc

This geometry type is stored in an IGDS type 16 element. Arc features are just like ellipse features, except that two additional angles control the portion of the ellipse boundary that is drawn. Arcs with 3D rotations will be stroked into lines and returned as igds_line elements.

Tip: The function @Arc() can be used to convert an arc to a linestring. This is useful for storing arcs in systems which do not support them directly.

Attribute Name	Contents
igds_primary_axis	The length of the semi-major axis in ground units. Range: Any real number > 0 Default: No default
igds_secondary_axis	The length of the semi-minor axis in ground units. Range: Any real number > 0 Default: No default
igds_start_angle	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of start_angle. Range: 0.0..360.0 Default: No default
igds_sweep_angle	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of sweep_angle. Range: Any real number > 0 Default: No default
igds_rotation	The rotation of the major axis. The rotation is measured in degrees counterclockwise up from horizontal. Range: -360.0..360.0 Default: 0
igds_arc_orientation	The orientation of the arc. As the sweep angle is always returned as positive, this field can be used to determine the original orientation of the arc. This attribute is only used during reading. Range: clockwise counterclockwise Default: none
igds_quat_p igds_quat_q igds_quat_r	Values of quaternion for 3D arcs Default: none

Attribute Name	Contents
igds_quat_s	

Cells

igds_type: igds_cell

Cells correspond to IGDS element type 2. The FME feature used to hold a cell element does **not** contain the complete set of elements which make up the cell's definition. Instead, FME features representing IGDS cells contain only the cell's name, as well as rotation and scaling parameters. The IGDS reader skips all elements that define the cell (extracting only the text strings from any text elements in the cell), and the IGDS writer extracts the cell description from the supplied cell library to be output. Cell features are point features and have only a single coordinate. Writing of named cells is not currently supported by the V8 writer. However, the V8 writer can successfully handle unnamed cells (groups).

The IGDS reader may be set to expand cells. If the mapping file contains a yes setting for IGDS_EXPAND_CELLS, then each member element of the cell is read and output. However, the cell insertion point itself is **not** output. In addition, the cell members are assigned a unique cell sequence number in the igds_cell_sequence_number. This number can be used to later regroup the cell components if that is required.

If the setting for IGDS_EXPAND_CELLS is no, then only the cell insertion point is output.

Both *graphic* and *point* cells are supported. Graphic cells use the level, color, and style information from the cell library, and must always have a feature type of 0. Point cells use the level, color, and style information provided in the mapping file. Note that for point cells, when cell header is assigned igds_color then this color is assigned as fill_color to all of its members capable of having fill_color i.e. shapes etc. If the cell header had igds_fill_color then it gets ignored.

Both V7 and V8 can write cells. V8 can also preserve the cell structure. For example, if the cell had any nested cells, complex chains or complex shapes, then the whole nesting is preserved.

The IGDS reader also supports orphan or unnamed cells and is controlled by the keyword IGDS_EXPAND_UNNAMED_CELLS. A named/unnamed cell can have further nested named/unnamed cells. The way the IGDS Reader treats them, depending on their respective keywords, is explained in these sections:

- Case-I: Named cell (root) nested named cell
- Case-II: Named cells (root) nested unnamed cells
- Case-III: Unnamed cells (root) nested unnamed cells
- Case-IV: Unnamed cells (root) nested named cells

Case-I: Named cell (root) nested named cell:

IGDS_EXPAND_CELLS (YES): The cell insertion point is not stored. Members of root and nested cells are stored as independent features.

IGDS_EXPAND_CELLS (NO): Only the root cell's insertion point is stored.

Case-II: Named cells (root) nested unnamed cells:

IGDS_EXPAND_CELLS (YES) AND IGDS_EXPAND_UNNAMED_CELLS(YES): Neither of the two cells is preserved. All members of both cells are output as independent features. No donuts are formed in case the unnamed cell contained overlapping polygons.

IGDS_EXPAND_CELLS (YES) AND IGDS_EXPAND_UNNAMED_CELLS(NO): Neither of the two cells is preserved. All members of both cells are output as independent features. No donuts are formed in case the unnamed cell contained overlapping polygons.

IGDS_EXPAND_CELLS (NO) AND IGDS_EXPAND_UNNAMED_CELLS(YES): Only the root cell is output. Nested unnamed cells are ignored.

IGDS_EXPAND_CELLS (NO) AND IGDS_EXPAND_UNNAMED_CELLS(NO): Only the root cell is output. Nested unnamed cells are ignored.

Case-III: Unnamed cells (root) nested unnamed cells:

IGDS_EXPAND_UNNAMED_CELLS(YES): The insertion point of the root cell is not preserved. No donuts are formed if existed. All elements of the root and the nested cells are given.

IGDS_EXPAND_UNNAMED_CELLS(NO): Make donuts of all members of the root and nested cells.

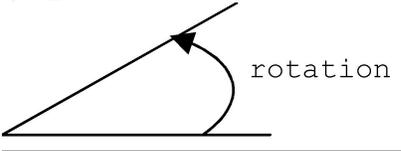
Case-IV: Unnamed cells (root) nested named cells:

IGDS_EXPAND_UNNAMED_CELLS (YES) AND IGDS_EXPAND_CELLS(YES): Neither of the cells is preserved. No donuts are formed. All elements of both cells are output as independent features.

IGDS_EXPAND_UNNAMED_CELLS (YES) AND IGDS_EXPAND_CELLS(NO): Only elements of the root unnamed cell are output. Nested cells are preserved and output as points.

IGDS_EXPAND_UNNAMED_CELLS (NO) AND IGDS_EXPAND_CELLS(YES): All elements of nested named cells are output. Donuts are formed. If both cells have donuts then an aggregate of donuts is formed.

IGDS_EXPAND_UNNAMED_CELLS (NO) AND IGDS_EXPAND_CELLS(NO): The insertion point of the root unnamed cell is preserved. Donuts are formed from the root cell only. The nested cell is ignored and so are its members.

Attribute Name	Contents
igds_cell_name	The name of the cell. Corresponds to the name of the cell in a cell library. Range: Character String Default: No default
igds_cell_x_scale igds_cell_y_scale igds_cell_z_scale	The scaling factors to apply to the cell. Range: Any real number > 0 Default: 1
igds_cell_size	The size in ground units of the maximum span of the cell. If this is specified, the settings for igds_cell_x_scale, igds_cell_y_scale, and igds_cell_z_scale are ignored. If it is not specified, then the scaling factors described above are used. This attribute is not assigned any value by the reader. Range: Any real numbers > 0 Default: No default
igds_rotation 	The rotation of the entire cell. The rotation is measured in degrees counterclockwise up from horizontal. Range: -360.0..360.0 Default: 0
igds_text_string{#}	When reading only, this contains the text string of the # th text element in the cell. Range: Any string
igds_cell_sequence_number	When reading only with IGDS_EXPAND_CELLS set to yes, this contains a unique number that can be used

Attribute Name	Contents
	to regroup a cell with its component elements.
igds_cell_size_x	This is the difference of minX and maxX stored in ground units. Note: If igds_cell_size_x and igds_cell_size_y are both specified, then igds_cell_size_x_scale, igds_cell_size_y_scale and igds_cell_size_z_scale values are ignored.
igds_cell_size_y	This is the difference of minY and maxY stored in ground units.
igds_cell_num_members	Stores a cell's total number of members. Range: Any real numbers > 0 Default: No default
igds_unnamedcell_num_of_elements	Stores number of elements of an unnamed cell (group) Range: Any real numbers > 0 Default: No default
igds_cell_insertion_x igds_cell_insertion_y igds_cell_insertion_z	Stores cell insertion point Range: Any real number Default: No default
igds_cell_element_class igds_cell_element_style igds_cell_element_color igds_cell_element_weight igds_cell_element_level	Stores properties of the cell if the cell is graphic. Default: No default
igds_cell2DTMat11 igds_cell2DTMat12 igds_cell2DTMat21 igds_cell2DTMat22	Cell's 2D matrix containing rotation and scale information. Default: No default
igds_cell3DTMat11 igds_cell3DTMat12 igds_cell3DTMat13 igds_cell3DTMat21 igds_cell3DTMat22 igds_cell3DTMat23 igds_cell3DTMat31 igds_cell3DTMat32 igds_cell3DTMat33	Cell's 3D matrix containing rotation and scale information. Default: No default

Cells (Shared)

igds_type: igds_shared_cell

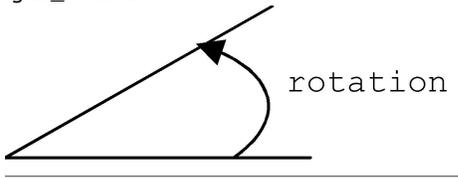
Shared cells correspond to IGDS element type 34 and 35. They consist of two parts: the definition (Type 34) and the element (Type 35). The definitions list the component elements of the cells. The elements are made up of an insertion point as well as rotation and scaling parameters. The IGDS reader skips all elements that define the cell and only processes the element features as point features that have only a single coordinate.

If IGDS_EXPAND_CELLS is set to yes, then the shared cells are expanded into its pieces; otherwise only the cell insertion point is output for each shared cell instance. Expansion of shared cells is supported by both V7 and V8.

The IGDS V8 writer can write shared cells.

Shared cell instances have the following attributes:

Attribute Name	Contents
igds_cell_name	The name of the cell. Range: Character String Default: No default
igds_cell_x_scale igds_cell_y_scale igds_cell_z_scale	The scaling factors to apply to the cell. Range: Any real number > 0 Default: 1
igds_rotation	The rotation of the entire cell. The rotation is measured in degrees counterclockwise up from horizontal. Range: -360.0..360.0 Default: 0
igds_sharedcell_description	The description of the cell. (Supported for version 8 DGN files only.) Range: Character String Default: No default
igds_cell_num_members	Stores a cell's total number of members. Range: Any real numbers > 0 Default: No default



Complex Shapes/Strings

igds_type: igds_complex_shape

igds_type: igds_complex_string

Complex shape/string elements are normally treated the same as shape/linestring elements by the IGDS reader. However, if the exact original composition of the complex shape is required, the IGDS_AGGREGATE_COMPLEX_CHAINS directive can be set to yes and then complex shape/string elements will be returned as single FME features with igds_complex_shape/igds_complex_string as their igds_type. This allows preservation of any arc elements that made up the boundary of the shape, for example.

The IGDS writer will accept and write out complex shape/string elements at any time.

The complex shape/string feature consists of an aggregate geometry. Each aggregate geometry corresponds to an entry in an attribute list. The list is called `igds_complex_elements{#}`, where # starts at 0 and increments for each aggregate element. The list's item names are identical to the component feature's attributes.

Splitting of FME geometries such as IFMEPaths into complex shape and complex string elements will occur automatically on write, such that the size limitations of complex shape and complex string elements are not exceeded. Similarly, each line string element component of complex shape and complex string elements will be created so as not to exceed the maximum number of coordinates for a line string.

Curves

igds_type: `igds_curve`

Curve features are used in Design files to represent smooth bezier curves. Curve features have four extra points which are used to determine the slope at the starting and ending points of the curve. These points are not part of the real coordinates of the feature, and are stored in the attribute list `igds_curve_slope{}`. The first two entries in the list define the slope points for the start of the feature, and the last two define the slope points for the end of the feature. The IGDS reader and writer interpret the curves coordinates as the points which define the curve. If the `PRESERVE_CURVES` directive is YES, then the reader does not interpolate points along the curve. If curves are not preserved, they will have interpolated points added to them and `igds_curve` elements will be returned as `igds_line` elements.

A curve feature has these attributes:

Attribute Name	Contents
<code>igds_curve_slope{0}.x</code> <code>igds_curve_slope{0}.y</code> <code>igds_curve_slope{0}.z</code> <code>igds_curve_slope{1}.x</code> <code>igds_curve_slope{1}.y</code> <code>igds_curve_slope{1}.z</code>	<p>The ground coordinates of the slope points for the beginning of the feature.</p> <p>If the design file was two-dimensional (2D), then the <code>.z</code> attributes will not be present.</p>
<code>igds_curve_slope{2}.x</code> <code>igds_curve_slope{2}.y</code> <code>igds_curve_slope{2}.z</code> <code>igds_curve_slope{3}.x</code> <code>igds_curve_slope{3}.y</code> <code>igds_curve_slope{3}.z</code>	<p>The ground coordinates of the slope points for the end of the feature.</p> <p>If the design file was 2D, then the <code>.z</code> attributes will not be present.</p>

Tip: When a curve feature is reprojected, its slope points are automatically reprojected.

BSpline Curves

igds_type: `igds_line`

This is stored as an IGDS type 27 element. The information of the poles, knots and weights of a spline are stored in element types 21, 26 and 28 respectively. Currently, only *reading* of bsplines is supported. The bsplines are read and stroked into segments (which is why its `igds_type` is stored as `igds_line`).

External Reference Files

Reference files can be read in two ways. In order to read the supported elements stored in reference files, the keyword `READ_XREF_FILES` has to be set to yes. The default is no. For this method of reading the following is true: All the reference files inherit the working units and offsets from the parent file and their respective units and offsets are ignored. The V8 reader can read both v7 and v8 attachments, whereas V7 will read only V7 references. Both V7 and V8 can read nested references. The nesting can be restricted to first level only by setting the keyword `READ_XREF_UPTO_FIRST_LVL` as true.

Alternatively or in addition, the V8 reader is able to read reference files are also read as individual features with an `igds_type` of `igds_xref`. These features are a non-graphical representation of the XREF elements themselves, not the elements stored within them. Though the V8 writer does not create external reference elements, when the V8 writer uses a seed file, the external reference file elements in the seed file are preserved and put into the destination dataset.

An external reference element has the attributes shown below.

Attribute Name	Contents
igds_xref_camera_focal_len	The focal length value for the camera used for the view of the external file reference data. Range: Real Number Default: 0.0
igds_xref_camera_pos_x	The position in the x dimension for the camera used for the view of the external reference file data. Range: Real Number Default: 0.0
igds_xref_camera_pos_y	The position in the y dimension for the camera used for the view of the external reference file data. Range: Real Number Default: 0.0
igds_xref_camera_pos_z	The position in the z dimension for the camera used for the view of the external reference file data. Range: Real Number Default: 0.0
igds_xref_desc	The position in the z dimension for the camera used for the view of the external file reference data. Range: String Default: No default
igds_xref_file_build_opts	The file builder option mask for the external reference file. Range: Integer Default: 15
igds_xref_file_disp_opts	The file displayer option mask for the external reference file. Range: Integer Default: 73858
igds_xref_file_name	The filename of the external reference file. This is a basename and extension, not a path. Range: String Default: No default

igds_xref_file_num	The file number for the external reference file. Range: Integer Default: 1
igds_xref_group_id	The number of the group to which this external reference file belongs. Range: Integer Default: 0
igds_xref_master_origin_x	The x dimension value for the origin of the external reference file position in master file UORs. Range: Real Number Default: 0.0
igds_xref_master_origin_y	The y dimension value for the origin of the external reference file position in master file UORs. Range: Real Number Default: 0.0
igds_xref_master_origin_z	The z dimension value for the origin of the external reference file position in master file UORs. Range: Real Number Default: 0.0
igds_xref_model_name	The name of the model of the external reference file. This value may be empty for the default model. Range: String Default: No default
igds_xref_name	The logical name of the external reference file. This value may distinguish between multiple references to the same reference file. Range: String Default: No default
igds_xref_nest_depth	The depth of nested reference of the external reference file. Range: 0..65536 Default: 0
igds_xref_parent_attach_id	The id of the parent attachment of the external reference file. Range: Numeric string Default: "0"
igds_xref_file_path	The file path of the external reference file. Range: String Default: No default

igds_xref_reference_origin_x	The x dimension value for the origin of the external reference file position in reference UORs. Range: Real Number Default: 0.0
igds_xref_reference_origin_y	The y dimension value for the origin of the external reference file position in reference UORs. Range: Real Number Default: 0.0
igds_xref_reference_origin_z	The z dimension value for the origin of the external reference file position in reference UORs Range: Real Number Default: 0.0
igds_xref_rotation	The rotation value for the external reference file. Range: Real Number Default: 0.0
igds_xref_scale	The conversion factor value for the external reference file. Range: Real Number Default: 1.0
igds_xref_version	The version number value for the external reference file. Range: Integer Default: 1
igds_xref_z_back	The back z clip value for the external reference file. Range: Real Number Default: 0.0
igds_xref_z_front	The front z clip value for the external reference file. Range: Real Number Default: 0.0

Multilines

igds_type: igds_line

The multilines are stored with their igds_type as igds_line, but the fact that they are multilines can be detected from igds_element_type, which is stored as type 36. The multilines are stored as lines, therefore they are written as lines when performing a DGN to DGN translation. Currently, the multilines are imported with their centerlines only. However, the attributes such as offset and symbology (style, weight, color) of the pieces are stored in the list attribute igds_multiline{ }. When reading multilines from a V7 dataset, the multilines are ignored if they are part of a cell. In addition, if the keyword READ_BYTE_OFFSET is set to true, then it gets ignored for multilines.

A multiline has the attributes shown below.

Attribute Name	Contents
igds_mlineStyle{#}.offset igds_mlineStyle{#}.style igds_mlineStyle{#}.color igds_mlineStyle{#}.weight igds_mlineStyle{#}.level	Where offset is the perpendicular distance of the piece from the centerline, and {}.style, {}.color, {}.weight and {}.level are the line styles, color, weight and level of the individual pieces.
igds_mlinehdr_num_lines	Number of pieces of the multiline. Range: Any real number > 0 Default: No default
igds_mlinehdr_num_breaks	Number of breaks of the multiline. Range: Any real number > 0 Default: No default
igds_mlinehdr_num_nodes	Number of nodes of the multiline. Range: Any real number > 0 Default: No default
igds_mlinehdr_startcap_angle	Angle in degrees of the start cap. Range: -360.0..360.0 Default: 0
igds_mlinehdr_endcap_angle	Angle in degrees of the end cap. Range: -360.0..360.0 Default: 0
igds_mlinehdr_freeze_group	Multiline header attribute - value is always 0 (for internal use by the toolkit)
igds_mlinehdr_version	Multiline header version - currently it is 3 (for internal use by the toolkit)
igds_mlinehdr_closed	Whether or not multiline is closed. Range: 0 or 1 Default: No default
igds_mlinehdr_arc_cap_by_profile_line	Multiline header flags (for internal use by the toolkit)
igds_mlinehdr_offset_model_valid	Multiline header flags (for internal use by the toolkit)
igds_mlinehdr_offset_mode	Multiline header flags (for internal use by the toolkit)
igds_mlinehdr_placement_offset	Global offset from definition points. Range: Real Number Default: No default

Attribute Name	Contents
igds_mlinehdr_style_id.lo igds_mlinehdr_style_id.hi	ID of multiline style element. Range: Integer Default: No default
igds_mlinehdr_styleScale	Scale of multiline style element. Range: Real Number Default: No default
igds_mlinehdr_updv.x igds_mlinehdr_updv.y igds_mlinehdr_updv.z	Up direction vector for 3D to determine side orientation. Range: Real Number Default: No default
igds_mlattrib_startcap.usestyle igds_mlattrib_startcap.useweight igds_mlattrib_startcap.usecolor igds_mlattrib_startcap.cap_on_arc igds_mlattrib_startcap.cap_out_arc igds_mlattrib_startcap.cap_line igds_mlattrib_startcap.use_class igds_mlattrib_startcap.customstyle igds_mlattrib_startcap.cap_color_from_segment igds_mlattrib_startcap.construction_class The same list of attributes is repeated for end_cap and joint; for example: igds_mlattrib_endcap.usestyle etc. and igds_mlattrib_joint.usestyle etc.	Flags specifying the properties of the multiline. Range: 0 or 1 Default: No default
igds_mlattrib_startcap.style	Style of start cap. Range: 0..7 Default: 0
igds_mlattrib_startcap.weight	Weight of start cap. Range: 0..31 Default: 0
igds_mlattrib_startcap.color	Color of start cap. Range: 0..254 Default: 0

Attribute Name	Contents
igds_mlattrib_endcap.style	Style of end cap. Range: 0..7 Default: 0
igds_mlattrib_endcap.weight	Weight of end cap. Range: 0..31 Default: 0
igds_mlattrib_endcap.color	Color of end cap. Range: 0..254 Default: 0
igds_mlattrib_joint.style	Style of joint cap. Range: 0..7 Default: 0
igds_mlattrib_joint.weight	Weight of joint cap. Range: 0..31 Default: 0
igds_mlattrib_joint.color	Color of joint cap. Range: 0..254 Default: 0
igds_mlineStyle{#}.offset	Offset of each member of multiline Default: No Default
igds_mlineStyle{#}.lineattrib.usestyle igds_mlineStyle{#}.lineattrib.useweight igds_mlineStyle{#}.lineattrib.usecolor igds_mlineStyle{#}.lineattrib.cap_on_arc igds_mlineStyle{#}.lineattrib.cap_out_arc igds_mlineStyle{#}.lineattrib.cap_line igds_mlineStyle{#}.lineattrib.use_class igds_mlineStyle{#}.lineattrib.customstyle igds_mlineStyle{#}.lineattrib.cap_color_from_segment igds_mlineStyle{#}.lineattrib.construction_class	Values defining the flag of line attribute Default: No Default
igds_mlineStyle{#}.lineattrib.style	Style of line attribute Range: 0..7 Default: 0

Attribute Name	Contents
igds_mLineStyle{#}.lineattrib.weight	Weight of line attribute Range: 0..31 Default: 0
igds_mLineStyle{#}.lineattrib.color	Color of line attribute Range: 0..254 Default: 0
igds_mLineStyle{#}.lineattrib.level	Level of line attribute Default: 0
igds_mlinenode_props	If a multiline has a large number of nodes, which is very likely, it will need to be cleaned up before viewing it in the Universal Viewer. Store these two attributes as strings with comma-separated values. This will be the protocol b_index1, bCount1, b_index2, bCount2, b_indexn, bCountn The writer will parse them in the same order and use them. Default: No default
igds_mlinebreak{#}.segmask	Mask bit set for each line that is broken. Default: No default
igds_mlinebreak{#}.from_joint igds_mlinebreak{#}.to_joint	Flags setting the line break properties Default: No default
igds_mlinebreak{#}.point_offset	Offset from point Default: No default
igds_mlinebreak{#}.length	Break length Default: No default
igds_mlinebreak{#}.angle	Reserved - should be 0.0 Default: No default

Dimensions

igds_type: igds_line

The dimensions are stored with their igds_type as igds_line but the fact that they are dimensions can be detected from igds_element_type which is stored as type 33. The keyword EXPLODE_DIMENSION_ELEM controls the way the dimensions are imported. When it is set to yes, the dimensions are exploded into its pieces; when it is set to no, it is imported as an aggregate. The default is yes. When dimensions are imported as aggregates, the arcs are stroked and text features are output as list attributes only. Therefore, when performing a DGN-to-DGN translation with the option EXPLODE_DIMENSION_ELEM set to no, the text features will be lost. When reading dimensions from a V7 dataset, the dimensions are ignored if they are part of a cell. If the keyword READ_BYTE_OFFSET is set to true, then the dimensions will also be ignored.

A dimension element has the following attributes

Attribute Name	Contents
igds_dim_text{#}.font igds_dim_text{#}.original_justification igds_dim_text{#}.text_size igds_dim_text{#}.text_width_multiplier igds_dim_text{#}.text_used_string_len igds_dim_text{#}.text_rotation igds_dim_text{#}.text_insertion_x igds_dim_text{#}.text_insertion_y igds_dim_text{#}.text_string	Stores the standard attributes of the text member of the dimension. For example, font, original_justification , size and width etc.
igds_dim_refx{#}.pt.x igds_dim_refx{#}.pt.y igds_dim_refx{#}.pt.z igds_dim_refx{#}.base_offset igds_dim_refx{#}.segment_text_offset igds_dim_refx{#}.flags igds_dim_refx{#}.rxflags.mode igds_dim_refx{#}.rxflags.hide_extension igds_dim_refx{#}.rxflags.use_text_margin igds_dim_refx{#}.rxflags.primary_text_exists igds_dim_refx{#}.rxflags.display_primary_plus_tolerance igds_dim_refx{#}.rxflags.display_primary_minus_tolerance igds_dim_refx{#}.rxflags.secondary_text_exists igds_dim_refx{#}.rxflags.display_secondary_plus_tolerance igds_dim_refx{#}.rxflags.display_secondary_minus_tolerance	Stores the attributes of reference points as list attributes of the dimension.
igds_dim_type	Type of dimension to draw. Range: 1..53 Default: No default
igds_dim_scale	Scale of dimension Range: Any real number > 0 Default: No default
igds_dim_style	Style of dimension Range: Any real number > 0 Default: No default
igds_dim_weight	Weight of dimension Range: 0..31

Attribute Name	Contents
	Default: 0
igds_dim_color	Color of dimension Range: 0..254 Default: 0
igds_dim_primary_accuracy	Primary accuracy of dimension Range: 0..254 Default: None
igds_dim_secondary_accuracy	Secondary accuracy of dimension Range: 0..254 Default: None
igds_witness_line_offset	Offset of witness lines Range: Real Number >0 Default: None
igds_witness_line_extension	Witness line extension Range: Real Number >0 Default: None
igds_base_to_text_dist	Base to text distance Range: Real Number >0 Default: None
igds_leader_to_text_dist	Leader to text distance Range: Real Number >0 Default: None
igds_text_min_leader	Text minimum leader Range: Real Number >0 Default: None
igds_arrow_width	Dimension arrow width Range: Real Number >0 Default: None
igds_arrow_height	Dimension arrow height Range: Real Number >0 Default: None
igds_center_mark_size	Center mark size Range: Real Number >0
igds_base_line_cos_value	Cos value of bearing angle of base line. Range: -1..1

Attribute Name	Contents
igds_witness_line_cos_value	Cos value for bearing angle of witness line. Range: -1..1
igds_base_line_sin_value	Sin value of bearing angle of base line. Range: -1..1
igds_witness_line_sin_value	Sin value of bearing angle of witness line. Range: -1..1
igds_quat.w igds_quat.x igds_quat.y igds_quat.z	Components of quaternion for 3D orientation Range: Any real number
igds_view_id	To allow to align by view Range: 1 character (1 byte)
igds_num_ref_points	Number of reference points Range: Any real number
igds_options_count	Count of options Range: Any real number
igds_dim_text_width	Width of text member Range: Real number > 0
igds_dim_text_height	Height of text member Range: Real number > 0
igds_text_font	Font of text member. See <i>Text Nodes</i> and <i>Text Strings</i> : igds_font. Range: 0
igds_text_color	Color of text member Range: 0..254
igds_text_weight	Weight of text member Range: 0..31
igds_use_text_color	Use text color and not the dimension element color Range: 1/0 (1 bit)
igds_use_text_weight	Use text weight and not the dimension element weight Range: 1/0 (1 bit)

Attribute Name	Contents
igds_measure_angle	Measure angle and not the distance Range: 1/0 (1 bit)

Ellipses

igds_type: igds_ellipse

This geometry type is stored in an IGDS type 15 element.

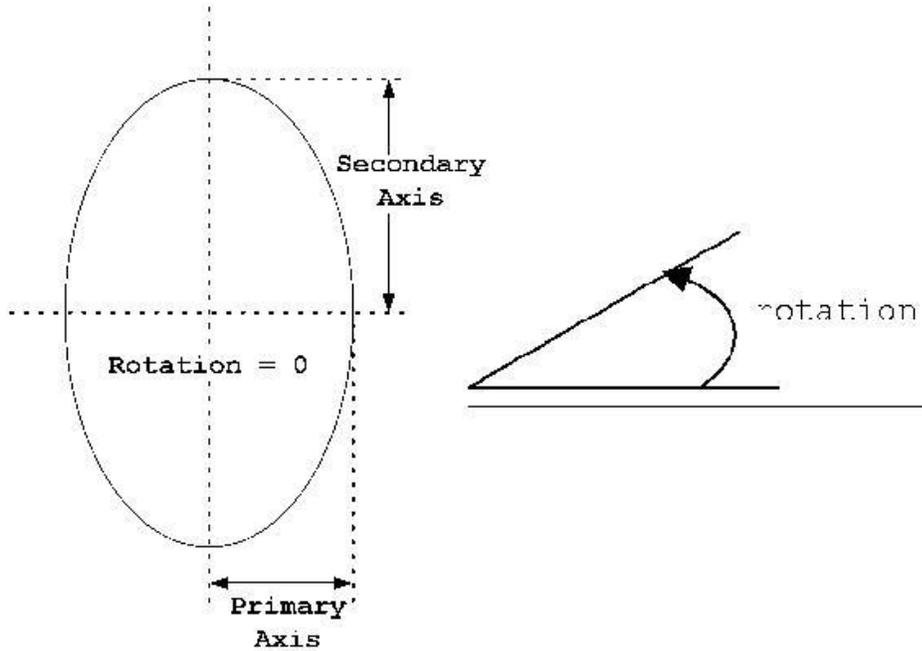
Ellipse features are point features, and only have a single coordinate. This point serves as the center of the ellipse. Additional attributes specify the rotation, major axis, and minor axis of the ellipse.

Tip: The function @Arc() can be used to convert an ellipse to a polygon. This is useful for storing ellipses in systems which do not support them directly.

Attribute Name	Version Info	Contents
igds_primary_axis		The length of the semi-major axis in ground units. Range: Any real number > 0 Default: No default
igds_secondary_axis		The length of the semi-minor axis in ground units. Range: Any real number > 0 Default: No default
igds_rotation		The rotation of the major axis. The rotation is measured in degrees counter-clockwise up from horizontal. Range: -360.0..360.0 Default: 0
igds_fill_color		The color used to fill the shape. This will override any solid fill linkage that may be present on the feature. Range: 0..255 Default: No fill
igds_fill_color.red		The fill's red color intensity, as determined by looking up the fill color index in the color table. Reader only. Range: 0..255
igds_fill_color.green		The fill's green color intensity, as determined by looking up the fill color index in the color table. Reader only. Range: 0..255
igds_fill_color.blue		The fill's blue color intensity, as determined by looking up the fill color index in the color table. Reader only. Range: 0..255
igds_fill_angle1	used in V8 only	The angle of the primary hatch lines in a hatch fill. The angle is measured in degrees counterclockwise up from hor-

Attribute Name	Version Info	Contents
		<p>izontal. Range: -360.0..360.0 Default: 0</p>
igds_fill_angle2	used in V8 only	<p>The angle of the secondary hatch lines in a hatch fill. The angle is measured in degrees counterclockwise up from horizontal. Range: -360.0..360.0 Default: 0</p>
igds_fill_hole_style_id	used in V8 only	<p>The style id of the holes of a filled area element. Default: 0</p>
igds_fill_island_style_id	used in V8 only	<p>The style id of the island areas of holes of a filled area element. Default: 0</p>
igds_fill_offset_x	used in V8 only	<p>The coordinate offset in the x dimension for the hatch or pattern fill of an area element. Range: Any real number Default: 0</p>
igds_fill_offset_y	used in V8 only	<p>The coordinate offset in the y dimension for the hatchv fill of an area element. Range: Any real number Default: 0</p>
igds_fill_offset_z	used in V8 only	<p>The coordinate offset in the z dimension for the hatch or pattern fill of an area element. Range: Any real number Default: 0</p>
igds_fill_pattern_color	used in V8 only	<p>The color used to fill the area element. This corresponds to the color of the hatch or pattern fill linkage. Range: 0..2545</p>
igds_fill_pattern_color.red	used in V8 only	<p>The pattern_fill's red color intensity, as determined by looking up the fill color index in the color table. Reader only. Range: 0..255</p>
igds_fill_pattern_color.green	used in V8 only	<p>The pattern_fill's green color intensity, as determined by looking up the fill color index in the color table. Reader only. Range: 0..255</p>
igds_fill_pattern_color.blue	used in V8 only	<p>The pattern_fill's blue color intensity, as determined by looking up the fill color index in the color table. Reader only. Range: 0..255</p>
igds_fill_pattern_type	used in V8 only	<p>The type of pattern fill for an area element. Currently only hatch is supported, and this indicates that there is one on an area element.</p>

Attribute Name	Version Info	Contents
		Range: hatch Default: hatch
igds_fill_pixel_size	used in V8 only	The size of pixels for the hatch or pattern fill of an area element. Range: Any real number
igds_fill_rotation	used in V8 only	The rotation angle of the hatch or pattern fill within an area element. The rotation is measured in degrees counterclockwise up from horizontal. Range: -360.0..360.0 Default: 0
igds_fill_scale	used in V8 only	The scale of the hatch fill or pattern fill for an area element. Range: Any positive real number
igds_fill_snappable	used in V8 only	This flag indicates if the hatch or pattern fill for an area element is snappable. Range: yes or no Default: yes
igds_fill_spacing1	used in V8 only	The spacing between the primary hatch lines in a hatch fill. Range: Any positive real number
igds_fill_spacing2	used in V8 only	The spacing between the secondary hatch lines in a hatch fill. Range: Any positive real number
igds_fill_style_id	used in V8 only	The line style id of the hatch fill or pattern fill for an area element. Default: 0
igds_fill_tolerance	used in V8 only	The maximum distance between curved element boundaries and line segments in a hatch fill. Range: Any positive real number
igds_fill_weight	used in V8 only	The line weight of the hatch fill or pattern fill for an area element. Default: 0



Note: The primary ellipse axis is not necessarily the longest axis, but rather the one whose orientation is specified by the rotation value.

Lines

igds_type: igds_line

Features with their `igds_type` set to `igds_line` are stored in and read from IGDS files in one of three ways, depending on the number of coordinates they have.

Number of Coordinates	IGDS Element Type	Description
2	3	If the feature contained exactly two points, then an IGDS type 3 element is used to store the data if <code>IGDS_CREATE_LINE_ELEMENTS</code> was yes; otherwise, a type 4 element will be created.
In V7: Between 3 and 101 In V8: Between 3 and 5000	4	If the coordinates can fit in a single element, then an IGDS type 4 element is used to store the line.
In V7: Greater than 101 In V8: Greater than 5000	12, 4	If the coordinates cannot fit into a single element, then they are grouped together into a complex line string element (type 12). This consists of a single type 12 element, followed by as many type 4 elements as required to hold all the coordinate data. The type 4 elements have their <i>complex</i> bit turned on. Furthermore, such a complex line string element

Number of Coordinates	IGDS Element Type	Description
		will be split into multiple complex line string elements if the size of the data exceeds the maximum size that the header can address, which is approximately 65K words.

There are no attributes specific to this type of element.

Points

igds_type: igds_point

Strictly speaking, the IGDS file format does not support point data. However, for easier interoperability with other formats, the IGDS reader and writer define a synthetic IGDS type for point data. Such features have only a single coordinate, and are stored in an IGDS type 3 element¹ as a zero length line with the start and the end point the same. When the IGDS reader encounters such an element, it assigns an *igds_type* of *igds_point*. If the IGDS reader encounters a type 3 element with a different start and end point, it will assign an *igds_type* of *igds_line*.

There are no attributes specific to this type of element.

Shapes

igds_type: igds_shape

Shape features are used in IGDS to represent closed polygons. The first coordinate in a shape feature must be equal to the last coordinate. Such features are stored in and read from IGDS files in one of two ways, depending on the number of coordinates in their boundaries:

Number of Coordinates	IGDS Element Type	Description
In V7: Between 3 and 101 In V8: Between 3 and 5000	6	If the coordinates can fit in a single element, then an IGDS type 6 element is used to store the shape.
In V7: Greater than 101 In V8: Greater than 5000	14, 4	If the coordinates cannot fit into a single element, then they are grouped together into a complex shape element (type 14). This consists of a single type 14 element, followed by as many type 4 elements as required to hold all the coordinate data. The type 4 elements have their <i>complex</i> bit turned on. Furthermore, such a complex shape element will be split into multiple complex line string elements if the size of the data exceeds the maximum size that the header can address, which is approximately 65K words.

Shape elements have the following attributes.

Attribute Name	Contents
<i>igds_fill_color</i>	The color used to fill the shape. This will override any solid fill

¹FME treats IGDS type 3 elements with different start and end points as *igds_line* features.

Attribute Name	Contents
	linkage that may be present on the feature. Range: 0..255 Default: no fill
igds_fill_color.red	The fill's red color intensity, as determined by looking up the fill color index in the color table. Reader only. Range: 0..255
igds_fill_color.green	The fill's green color intensity, as determined by looking up the fill color index in the color table. Reader only. Range: 0..255
igds_fill_color.blue	The fill's blue color intensity, as determined by looking up the fill color index in the color table. Reader only. Range: 0..255

Tip: Shapes will not be filled in MicroStation unless the 'View Attributes: Fill' checkbox is ticked, and a fill color is specified.

Solids

igds_type: igds_solid

Solids correspond to the grouped shapes in MicroStation. Solids consist of polygons or donut polygons. When a donut polygon is written out as a solid, all holes are output with the hole bit turned on, and are grouped together with the enclosing polygon. Groups are created in the Design file by creating an unnamed cell header element, and making each shape in the donut polygon a member of the group.

If EXPAND_UNNAMED_CELLS is set to yes, then unnamed cell components are output but the cell header itself is not output. In this case, donut polygons will not be formed from member shape elements. All member elements will retain their original colors. If it is NO (which is the default), then the cell is not exploded into its components and only the cell header is output. Donut polygons may be formed if multiple intersecting polygons existed.

If a solid consists of polygons without holes, then it is written out as igds_shape.

Solids are always filled, and accept an additional parameter to define the fill color. Holes within a solid will not be filled with the color.

The IGDS file format imposes a limit on the number of coordinates which can be present in a solid. This limit is around 16,000 for two-dimensional IGDS files, and around 10,000 for three-dimensional IGDS files. If a solid with more than an allowable number of coordinates is encountered, it is rejected and a message to that effect is logged to the FME log file.

Tip: Solids will not be filled in MicroStation unless the 'View Attributes: Fill' checkbox is ticked.

Solid elements have the following attributes.

Attribute Name	Contents
igds_fill_color	The color used to fill the solid. This will override any solid fill linkage that may be present on the feature. Range: 0..255 Default: 0
igds_fill_color.red	The fill's red color intensity, as determined by looking up the fill color index in the color table. Reader only.

Attribute Name	Contents
	Range: 0..255
igds_fill_color.green	The fill's green color intensity, as determined by looking up the fill color index in the color table. Reader only. Range: 0..255
igds_fill_color.blue	The fill's blue color intensity, as determined by looking up the fill color index in the color table. Reader only. Range: 0..255

3D Solids

igds_type: igds_3d_solid

This is supported for v8 only. This element should not be confused with igds_solid as it correspond to dgn element type 19 and not to unnamed cell. FME supports both reading and writing, however, its support is limited to only extrusions for now. Work to add support for other solid types is currently underway.

Tags

Elements in a design file may have user-defined attributes attached to them. Such attributes are called *tags*, and these may be read and written (DGNV8 only) by FME. In addition, to supply a value for a user-defined attribute, tags may also be displayed as text in the original design file. The TAGS_AS_TEXT directive controls whether or not tag data elements will be returned as text elements.

When reading a design file, FME first scans for all the tag data elements and tag set definition elements. Then as it reads each graphical element from the design file, it uses the element association ID to reconnect the data and attribute names with the graphical element. All the tag data values are then added to the feature returned into FME.

The attributes shown in the following table are added to an element for each associated tag.

Note: <tag name> is replaced by each TAG NAME that may be associated with the element. For example, if the element is associated with tags called "NUMLANES" and "PAVETYPE", then the feature would have attributes like "NUMLANES", NUMLANES.height, PAVETYPE, PAVETYPE.rotation, etc.

Note that most of the tag attributes are same as those of text. For example, igds_tag_names{}.height is the same as igds_text_height and is therefore not explicitly documented. All the other tag attributes are documented as follows:

Attribute Name	Contents
igds_tag_names{}	List of tag names attached to an element. Default: No default Required when writing tags through list attributes.
<tag name>.tagset_name	The name of the tagset the tag belongs to. Default: No default Required when writing tags to DGNV8
<tag name>.tagtype	The unique tag id Default: 1 Range: 1 = tag of type character string 3 = tag of type integer

	4 = tag of type double Optional when writing tags to DGNV8. In case tag-type is not provided, it always defaults to character string (i.e., type 1).
<tag name>.prompt	The value of tag prompt as defined in the tagset Default: No default Optional when writing tags to DGNV8
<tag name>.default_value	The value of tag default as defined in the tagset Default: No default Optional when writing tags to DGNV8
<tag name>.display	The display value of tag as defined in the tagset. Note that the writer will always set it to NO if tag offsets are not found on the feature Default: no Range: yes/no Optional when writing tags to DGNV8
<tag name>.x_offset <tag name>.y_offset <tag name>.z_offset	Tag offset from the element. In case these values are not provided the writer uses some default values to offset tags from the element Default: No default Optional when writing tags to DGNV8
<tag name>.urx <tag name>.ury <tag name>.urz	Tag upper right range of rectangle. Default: No default Not required when writing tags to DGNV8
<tag name>.llx <tag name>.lly <tag name>.llz	Tag lower left range of rectangle. Default: No default Not required when writing tags to DGNV8

Note that tag writing is supported by DGNV8 only. In order to attach tags to an element, set writer keyword WRITE_TAGS to yes. There are two ways tagset and tags definitions can be carried over to V8 writer:

1. *Through DEF lines:* This is the default behavior. The writer looks at the DEF lines to extract the information of tagsets and tag names. The feature type is assigned as the tagset whereas the user attributes become its tags. For instance, if the DEF line looks like this:

```
DGNV8_DEF Roads
      Name char(50)
      Type integer
```

then a tagset gets written with the name "Road" consisting two tags namely "Name" of data type string and "Type" of data type integer. The possible data types are char(n), integer and double. Note that this approach is introduced to automate the tag writing process and to avoid the amount of work involved using the "list attribute" approach as explained later. This approach has the following limitations:

- a. a. can write one tagset per feature only
- b. b. can define tag names and their corresponding data types only. The tag default, prompt and display properties cannot be set. The display property is automatically set to no and in order to turn it on the user need to set <tag_name>.display attribute on the feature with the value of yes.

If someone wants to write multiple tagsets then he can do so by using the "list attribute" approach. Also note that in case the writer sees list attributes under the name igds_tag_names{} it ignores the tagset definitions provided on the DEF lines.

2. *Through list attributes:* Another way of writing tags is by providing all tag names as list attributes to igds_tag_names{} on the feature.

The DGNV8 writer looks for the following attributes only when writing tags and uses them to calculate all other values. Therefore, any tag related attribute provided other than the following will be ignored.

```
igds_tag_names{}
<tag name>.tagset_name
<tag name>.tagtype
<tag name>.prompt
<tag name>.display
<tag name>.default_value
<tag name>.x_offset
<tag name>.y_offset
<tag name>.z_offset
```

In case tag offsets are not provided then the writer uses some default values for the offsets and turns off tag's display property.

Some tips on tag writing to avoid surprises:

When going from dgn->dgn, it is advised to ensure that the option "TAG_AS_TEXT" is turned off to avoid getting extra text element on top of the tags being written. Note that this option is set to "no" by default.

When going from dgn->dgn, if the source has tags attached to a cell then note that exploding cell will result into attaching tags to each cell member. Thus, each cell member will have same tags written in the output file.

Multi-text Strings

igds_type: igds_multi_text

Multi-text string features correspond to an IGDS text node (element type 7) grouping together a series of IGDS text elements (element type 17), each of which have their complex bit turned on. This feature uses the same attribute names as a text node, plus it has a feature attribute list of text string attributes. The list is called igds_text_elements{#}, where # starts at 0 and increments for each text element. The list's item names are identical to the text string features attributes.

Tip: Multi-text strings can be used to group together text so that it will be manipulated as a single entity with MicroStation.

Multi-text features are point features, and only have a single coordinate. This coordinate is used when the text node is created. If the feature had no coordinates of its own, the text node is created with the coordinates of the first text string. The coordinates for each of the text strings are stored in the FME feature using the following attribute names.

Attribute Name	Contents
igds_text_elements{#}.x	The x coordinate of the # th text element. Range: Any real number Default: No Default

Attribute Name	Contents
igds_text_elements{#}.y	The y coordinate of the # th text element. Range: Any real number Default: No Default
igds_text_elements{#}.z	The z coordinate of the # th text element. Range: Any real number Default: 0
igds_number_of_strings	The number of text elements in the multi-text feature
igds_split_multitext	Is added to the feature with the value "yes" if splitting multi-text. Default: No Default

If a setting for a particular text element is not present in the `igds_text_elements` list, then the setting specified for the previous text element will be used. If the first element does not have some settings specified, then the corresponding settings will be borrowed from the text node.

Tip: When a multi-text string feature is reprojected, its rotation and text size are also automatically adjusted to be correct in the new coordinate system.

For example, the FME feature specified by the below partial transfer specification would create a text node, followed by two text strings, as a single complex element.

```
IGDS 32 igds_type          igds_multi_text \
  igds_node_number      15 \
  igds_font              31 \
  igds_rotation          0 \
  igds_text_size         40 \
  igds_color              2 \
  igds_justification     1 \
  igds_text_elements{0}.igds_font      33 \
  igds_text_elements{0}.igds_rotation  3.1 \
  igds_text_elements{0}.igds_text_size  52 \
  igds_text_elements{0}.igds_color      4 \
  igds_text_elements{0}.igds_text_string Hello \
  igds_text_elements{0}.x                477556 \
  igds_text_elements{0}.y                5360183 \
  igds_text_elements{0}.z                20 \
  igds_text_elements{1}.igds_text_string world \
  igds_text_elements{1}.x                47755 \
  igds_text_elements{1}.y                5359177 \
  igds_text_elements{1}.z                20
```

Note that in this example, the justification code (1) used for the text node would be propagated to each of the text elements, but that the color used in the text node (2) would not be used in any of the text elements because the first one set the color to 4.

The in-memory snapshot of the FME feature created by the IGDS writer from this transfer specification is shown below.

Feature Type: 32	
Attribute Name	Value
igds_type	igds_multi_text

Feature Type: 32	
Attribute Name	Value
igds_node_number	15
igds_font	31
igds_weight	1
igds_text_size	40
igds_color	2
igds_rotation	0
igds_justification	1
igds_text_elements{0}.igds_text_string	Hello
igds_text_elements{0}.igds_font	33
igds_text_elements{0}.igds_rotation	3.1
igds_text_elements{0}.igds_justification	1
igds_text_elements{0}.igds_text_size	52
igds_text_elements{0}.x	477556
igds_text_elements{0}.y	536018
igds_text_elements{0}.z	20
igds_text_elements{1}.igds_text_string	World
igds_text_elements{1}.igds_font	33
igds_text_elements{1}.igds_rotation	3.1
igds_text_elements{1}.igds_justification	1
igds_text_elements{1}.igds_text_size	52
igds_text_elements{1}.x	477556
igds_text_elements{1}.y	5359177
igds_text_elements{1}.z	20
Coordinates: (477553,5360181,20)	

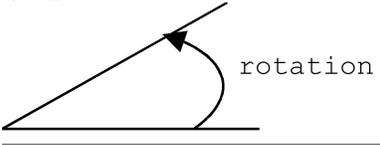
Text Nodes

igds_type: igds_text_node

Text nodes correspond to IGDS element type 7. Text node features are point features, and only have a single coordinate. Normally, text nodes are used to group together lines of text into a single complex element. However, such text groups are handled by the igds_multi_text type and not by this type, which is used only for text nodes with no attached text.

Tip: Free standing text nodes are often used as point features in IGDS files, with the text node number holding a key to related attribution.

Text node elements have the following attributes.

Attribute Name	Contents
igds_node_number	The node number assigned to the text node. Range: 0..65535 Default: 0
igds_font	The IGDS font number for the text node. For free standing text nodes, this value is relatively meaningless. Values from 0..511 are RSC fonts, while values from 512..1023 are SHX font, and values above 1023 are True Type fonts. Range: 0 Default: 25
igds_rotation 	The rotation of the text node. The rotation is measured in degrees counter clockwise up from horizontal. For free standing text nodes, this value is relatively meaningless. Range: -360.0..360.0 Default: 0
igds_justification	The justification code for the text node. Range: 0..14 0 is Left/Top 8 is Center/Bottom 1 is Left/Center 9 is Right Margin/Top 2 is Left/Bottom 10 is Right Margin/Center 3 is Left Margin/Top 11 is Right Margin/Bottom 4 is Left Margin/Center 12 is Right/Top 5 is Left Margin/Bottom 13 is Right/Center 6 is Center/Top 14 is Right/Bottom 7 is Center/Center Default: 5
igds_text_size	The text size of the text in the node. This is stored as the text height in the element. The text size is measured in ground units. Range: Any real number > 0 Default: 20
igds_text_width_multiplier	The text width of the text in the node. The text width is measured in ground units. If this is not supplied, then igds_text_size is used. Range: Any real number > 0 Default: Value of igds_text_size
igds_line_spacing	The line spacing between lines of text associated with the text node. It is measured in ground units.

Attribute Name	Contents
	Range: Any real number > 0 Default: 0
igds_number_of_strings	The number of text elements associated with the text node. If the number is greater than 0, then the text node is returned as an igds_multi_text. This is only used by the IGDS reader. Range: integer >= 0 Default: Not applicable. This field is only used by the reader.
igds_max_string_length	The maximum length of the strings associated with the text node. Range: integer >= 0 Default: 255
igds_max_used_string_length	The actual length of the strings associated with the text node. (Not included in Version 8 DGN files.) Range: integer >= 0 Default: 0

Tip: When a text node feature is reprojected, its rotation and text size are also automatically adjusted to be correct in the new coordinate system.

Text Strings

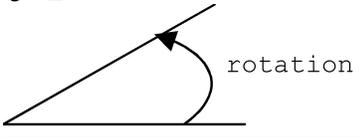
igds_type: igds_text

Text string features correspond to IGDS element type 17. Normally, text strings are grouped together into a single complex element within MicroStation by text nodes. However, such text groups are handled in the FME by the igds_multi_text type and not by this type, which is used only for single, free standing text strings. Text string features are point features, and only have a single coordinate. Note that V8 reader is capable of reading texts in unicode (UTF-16) in Windows only.

Tip: Some applications may use the graphic group field to logically group related text elements together.

Text strings have the following attributes.

Attribute Name	Contents
igds_original_justification	This attribute contains the original justification of the element when it was placed in the Design file. Once placed, all text elements are stored in the Design file using lower left corner (code 2) justification. Therefore, all text elements returned by the reader have an igds_justification of 2. The IDGS reader returns the original justification code in this attribute. The IGDS writer stores the value of this attribute in the justification bits of the placed text element, but it does NOT use its contents to deter-

Attribute Name	Contents
	<p>mine placement of the text. Range: 0..12</p>
igds_text_string	<p>The text string to be output. Text strings longer than 255 characters cannot be stored in a Design file and will be broken into multiple separate text elements. Range: Any string Default: No Default</p>
igds_font	<p>The IGDS font number for the text string. Values from 0..511 are RSC fonts, while values from 512..1023 are SHX font, and values above 1023 are True Type fonts. Range: 0 Default: 25</p>
igds_rotation 	<p>The rotation of the text string. The rotation is measured in degrees counterclockwise up from horizontal. Range: -360.0..360.0 Default: 0</p>
igds_justification	<p>The justification code for the text string. See the <i>Text Node</i> section for documentation on the mapping of numbers to alignments. Note that if this is specified, the IGDS writer will compute the lower left corner of the text as best it can and use that when the element is written to the Design file. Text elements in a Design file are always stored using a lower left corner. Range: 0..2,6..8, and 12..14 Default: 2</p>
igds_text_size	<p>The text size of the text, measured in ground units. This is stored as the height of the text element. Range: Any real number > 0 Default: 20</p>
igds_text_width_multiplier	<p>The text width of the text. The text width is measured in ground units. If this is not supplied, then igds_text_size is used. Range: Any real number > 0 Default: value of igds_text_size</p>
igds_text_num_lines	<p>If this is specified and is greater than 1, it will cause the text string to be broken into the number</p>

Attribute Name	Contents
	<p>of lines specified, and output as that number of text elements stacked vertically. If this is not supplied, then <code>igds_text_size</code> is used. Range: Any integer > 0 Default: 1</p>
<code>igds_text_horizontal_flip</code>	<p>Indicates whether or not the text should be flipped horizontally when it is displayed. This is represented in a Design file by storing the text width as a negative number if the text should be flipped. (Not included in Version 8 DGN files.) Range: Yes No Default: No</p>
<code>igds_text_vertical_flip</code>	<p>Indicates whether or not the text should be flipped vertically when it is displayed. This is represented in a Design file by storing the text height as a negative number if the text should be flipped. (Not included in Version 8 DGN files.) Range: Yes No Default: No</p>
<code>igds_insertion_x</code> <code>igds_insertion_y</code> <code>igds_insertion_z</code>	<p>The x, y, and z location of the original insertion point for the text, before the justification was applied. Reader only. Range: Any real number > 0</p>
<code>igds_lower_x</code> <code>igds_lower_y</code> <code>igds_upper_x</code> <code>igds_upper_y</code>	<p>The lower left and upper right x and y coordinates of the bounding box of the text. Reader only. Range: Any real number > 0</p>
<code>igds_textstyle_id</code>	<p>The ID of the textstyle being used. Default: No</p>
<code>igds_textstyle_char_spacing</code>	<p>Textstyle character spacing. Default: No</p>
<code>igds_textstyle_slant</code>	<p>Textstyle slant Default: No</p>
<code>igds_textstyle_underline_spacing</code>	<p>Textstyle underline spacing Default: No</p>
<code>igds_textstyle_underline_color</code>	<p>Textstyle underline color Default: No</p>
<code>igds_textstyle_underline_style</code>	<p>Textstyle underline style</p>

Attribute Name	Contents
	Default: No
igds_textstyle_underline_weight	Textstyle underline weight Default: No
igds_textstyle_overline_spacing	Textstyle overline spacing Default: No
igds_textstyle_overline_color	Textstyle overline color Default: No
igds_textstyle_overline_style	Textstyle overline style Default: No
igds_textstyle_overline_weight	Textstyle overline weight Default: No
igds_textstyle_line_offset.x igds_textstyle_line_offset.y	Coordinates of textstyle line offset Default: No
igds_textstyle_codepage	Textstyle codepage. . Specified as a the integer portion of a Microsoft Windows code page, i.e., 1252 for Latin I. Default: No
igds_textstyle_bg_color	Textstyle background color Default: No
igds_textstyle_bg_style	Textstyle background style Default: No
igds_textstyle_bg_weight	Textstyle background weight Default: No
igds_textstyle_bg_border.x igds_textstyle_bg_border.y	Coordinates of textstyle background border Default: No
igds_textstyle_bg_fill_color	Textstyle background fill color Default: No
igds_textstyle_color	Textstyle color Default: No
igds_textstyle_font	Textstyle font Default: No
igds_textstyle_tnode_word_wrap_len	Textstyle word wrap length Default: No
igds_textstyle_overrides_style1	Textstyle override styles Default: No

Attribute Name	Contents
igds_textstyle_overrides_style2	
igds_textstyle_txflags	Textstyle flags. A reasonable setting to enable text style on write is 512. Default: No
igds_textstyle_exflags	Textstyle extended flags

Notes: When a text string feature is reprojected, its rotation and text size are also automatically adjusted to be correct in the new coordinate system.

When writing textstyles, make sure that all the textstyles are added to the seed file being used.

Writing Levels in V8 (DEFLine Params)

In V8, the feature type is always taken as the level name. When WRITE_TAGS is set to yes then feature type will also be the tagset name. For more information refer to section under TAGS. Levels are created with this name and the level numbers are assigned from the DEF line. However, for backward compatibility, a feature's igds_level and igds_level_name overwrite the level_number and feature_type.

The following protocol is used when processing levels:

1. If the level is already provided in the seed file, then it is left as-is.
2. If the feature type has a corresponding DEF line parameter, and if that level is not already in the seed file, then that level is created with symbology as defined on the DEF line. This allows users to create levels with the desired symbology. Note that in order to apply the level symbology to the features belonging to the level, the attributes igds_color_set_bylevel, igds_style_set_bylevel and igds_weight_set_bylevel must be set to Yes. If none of the above are provided, then symbology of the first feature appearing in a level is assigned as its symbology.
3. If the feature type has a corresponding DEF line parameter, and if the value of igds_level is undefined and the feature type (level Name) is not "Default" then the DGN writer assigns level number 1 to it.

The DEFLine Params for defining levels are as follows:

Parameter Name	Contents
igds_level	The level number corresponding to the feature type. Note that feature type is treated as level name.
igds_level_comment	The comment of the destination level.
igds_level_color	The color of the destination level.
igds_level_style	The style of the destination level.
igds_level_weight	The weight of the destination level.

Example

```
DGNV8_1_DEF test2 \
igds_level 4 \
igds_level_comment "This is a test" \
igds_level_color 3 \
igds_level_style 4 \
igds_level_weight 2
```

In this case, the level will be created with the level name "test2", the corresponding level number of 4 and the comment and symbology as defined above. Note that if any feature being written to this level is intended to have symbology by_level then the attributes igds_color_set_bylevel, igds_style_set_bylevel and igds_weight_set_bylevel must be set to Yes.

Note that destination feature types are treated differently for V7 and V8. Version 7 always sees the destination feature types as level numbers, whereas V8 sees them as level names. For V8, the feature attribute igds_level_name overwrites feature_type, and feature attribute igds_level overwrites DEF line parameter igds_level.

Note that a workspace originally created using a V8 seed file can only be used to write to V8. A workspace originally created using a V7 seed file can be used to write to both V7 (although there will be a difference in the way destination feature types are handled) or V8. Two additional limitations are applied to V8:

1. Limitation of level numbers from 1 to 63 will also be applied to V8.
2. Feature types will always be generated as level numbers just like V7, but those level numbers will be treated as level_names by V8. For instance, if you tried to write to level_number 3, the level would be written as level number 3 for V7, but the level name would be written as "3" for V8 (when its number may or may not be 3). This can be overcome by specifying values for igds_level_name and igds_level.

Bentley MicroStation GeoGraphics Reader/Writer

Format Notes: This format is not supported by FME Base Edition.

The (Bentley) MicroStation GeoGraphics Reader/Writer is nearly identical to the Intergraph MGE Reader/Writer. The only difference is that by default, the <ReaderKeyword> for the MGE reader is MGE.

Refer to the chapter *Intergraph MGE Reader/Writer* for specific information about both the GG and MGE formats.

Overview

The MicroStation GeoGraphics Reader and Writer Modules provide the FME with the ability to read and write design files and their associated databases. This chapter assumes an operational knowledge of MicroStation GeoGraphics.

GeoGraphics Quick Facts

Format Type Identifier	GG
Reader/Writer	Both
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	GeoGraphics Feature Name
Typical File Extensions	.dgn, .cad
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	Yes
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Enhanced Geometry	Yes
Geometry Type	igds_type

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	point	yes
circles	yes	polygon	yes
circular arc	yes	raster	no
donut polygon	yes	solid	no
elliptical arc	yes	surface	no
ellipses	yes	text	yes

Geometry Support				
Geometry	Supported?		Geometry	Supported?
line	yes		z values	yes
none	no			

Canadian Council on Geomatics Interchange Format (CCOGIF) Reader/Writer

Format Notes:

This format is not supported by FME Base Edition.

The Canadian Council on Geomatics Interchange Format (CCOGIF) ASCII reader and writer module provides FME with access to the contents of a CCOGIF dataset stored in ASCII format in a single disk file. The structure of this file is discussed in Canadian Council on Geomatics' document, *Standard File Exchange Format for Digital Spatial Data* version #2.3, published October 1994.

The CCOGIF format, a data exchange format, provides a very general medium in which to represent a data model. FME accesses the individual records of a CCOGIF file at a very low level, involving only minimal interpretation of the contents of those records. This allows FME to handle virtually any data encoded with the CCOGIF standard, but requires a somewhat more sophisticated mapping file to make full use of the data.

Overview

The CCOGIF disk file consists of a series of logical records. Each of these records either describes *metadata* which is information about the data contents or structuring, or *entity data* which are geometric features.

The CCOGIF file describes a single data volume, that groups spatial data into *datasets*, *data groups*, and *data themes*. A CCOGIF volume contains one or more datasets. A single CCOGIF dataset contains one or more data groups, and a single data group contains one or more data themes.

At the highest level of grouping, the CCOGIF dataset – not to be confused with FME's concept of a dataset which is referred to as an FME dataset for the remainder of this chapter – groups the entity data by geographic region, such as a map sheet. In other words, all geographic data contained in a single CCOGIF dataset are somehow geographically related. All entity data within a CCOGIF dataset are measured in a single coordinate system.

Each data group provides some conceptual grouping of geographic entities. The criteria of this grouping are entirely data-dependent and are not constrained by the CCOGIF standard. This grouping is somewhat analogous to FME's notion of a feature type. For example, a CCOGIF dataset might contain the data groups *Highway*, *Bridge*, and *Intersection*.

The data within a single CCOGIF data group is divided into data themes. Each data theme represents a certain entity type: point, line, or area. The definition of a theme includes a list of data attributes. All attributes are defined on every entity record within the theme. A single data group may contain more than one theme of a given type – for example, two point themes. The themes are always ordered so that point themes come first, then line themes, and finally area themes.

CCOGIF ASCII Quick Facts

Format Type Identifier	CCOGIF
Reader/Writer	Both
Licensing Level	Professional
Dependencies	Writer requires extra-cost plug-in from Safe Software
Dataset Type	File
Feature Type	Group base name
Typical File Extensions	.asc
Automated Translation Support	Yes for Reader No for Writer
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	Never
Spatial Index	Yes
Schema Required	No
Transaction Support	Yes
Geometry Type	ccogif_entity_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	yes		z values	no
none	no			

Reader Overview

For the most part, the CCOGIF reader simply returns a feature to represent each record it encounters in the CCOGIF file. The reader does not have any requirement for definition statements.

The feature type of a feature returned from the CCOGIF reader depends on whether the feature represents metadata or entity data. Features that represent metadata records are returned with a feature type of **CCOGIF_METADATA**, whereas features that represent entity records are returned with a feature type dependent on the CCOGIF data group and theme within that data group. The feature type will have the format **<GroupName>_<ThemeIndex>**, where

<GroupName> is the name of the group extracted from the Data Group Header Record (DGHR), and <ThemeIndex> is the position of the data theme within the group.

There are different ways to generate mapping files to read CCOGIF data. The generated mapping files run the features through a number of factories, so the actual names of the feature types used in an automatically generated mapping file will depend on which method is used and may not correspond to the feature types returned from the reader itself. The different methods are discussed later in this section, under the heading **Generated Mapping Files**.

Reader Directives

The suffixes shown are prefixed by the current <ReaderKeyword> in a mapping file. By default, the <ReaderKeyword> for the CCOGIF reader is **CCOGIF**.

DATASET

Required/Optional: *Required*

The value for this keyword is the name of the file containing the CCOGIF volume to be read. A typical mapping file fragment specifying an input CCOGIF volume looks like this:

```
CCOGIF_DATASET /usr/data/ntdb/021g01.asc
```

Note: Notice that this refers to the CCOGIF volume and not the CCOGIF dataset. There may be several datasets in a single CCOGIF volume.

Workbench Parameter: *Source CCOGIF File(s)*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

✳ Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>

* **Workbench Parameter**

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]

* **Workbench Parameter**

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* **Workbench Parameter**

Additional Attributes to Expose

Writer Overview

The CCOGIF writer provides the ability to write FME feature data to a single CCOGIF file, the name of which is specified by the **DATASET** keyword. The contents of this file forms a single CCOGIF volume, consisting of exactly one CCOGIF dataset.

Metadata records may be inserted into the output data stream (for example, with the **CreationFactory**) to define the precise contents of all metadata records in the output file. This technique is described later in this chapter, under the heading *Defining Volume Structure*.

Unlike the CCOGIF reader, the writer requires **DEF** lines to define the attributes of the output CCOGIF file. The writer provides a mechanism in the **DEF** lines to precisely specify the attributes and order of every theme within each data group being written.

FME ships with sample mapping files that output CCOGIF from a NULL data source, as well as from Oracle. These are found in the **gallery/ccogif** subdirectory of the FME installation.

Writer Directives

The following directives are processed by the CCOGIF writer.

- DATASET
- DEF
- AREA_RELPOSN_ATTR

The suffixes shown are prefixed by the current `<writerKeyword>` in a mapping file. By default, the `<writerKeyword>` for the CCOGIF writer is `CCOGIF`.

DATASET

The file name of the output CCOGIF data file.

DEF

The CCOGIF `DEF` line is required to specify the contents of a CCOGIF data theme before any geometric entities may be written to that theme. All entities in a given data theme have the same geometric entity type – point, line, or area – and have the same set of attributes defined on them. The `DEF` line for the theme provides this information.

The syntax of a CCOGIF `DEF` line is:

```
<writerKeyword>_DEF <themeName> \  
    [CCOGIF_GROUP_NAME <groupName>] \  
    [CCOGIF_THEME_ENTITY_TYPE <entityType>] \  
    [CCOGIF_THEME_ORDERING <orderIndex>] \  
    [<attrName> <attrType>]+
```

The `<themeName>` is simply the identifier used within the mapping file to refer to the theme. Data themes do not have identifiers within the CCOGIF file, so the chosen `<themeName>` is not actually reflected in the CCOGIF file.

The `CCOGIF_GROUP_NAME` keyword specifies the name of the group containing the data theme. The value `<groupName>` is placed into the `ccogif_data_group_name` attribute for the theme's group's header record. All themes given a common `<groupName>` value belong to the same group.

In general, the `DEF` lines require each theme to be explicitly specified. Exceptions to this are noted later in this section, under the heading *Defining Volume Structure*.

The `CCOGIF_THEME_ENTITY_TYPE` keyword specifies the geometric type of the entity records to be written to the theme and is required in most cases. The value of `<entityType>` must be one of the values `ccogif_point`, `ccogif_line`, or `ccogif_area`.

The optional `CCOGIF_THEME_ORDERING` keyword allows each theme to be assigned a numeric ordering value. When the themes are written out to the CCOGIF file, they are ordered so that:

- All point themes belonging to a given data group are written first, followed by line themes, and finally area themes.
- All of a group's themes of a given entity type – line, point, or area – are written with a numerically increasing `<orderIndex>` value.

Themes for which no ordering index was specified are written with an arbitrary relative ordering after all themes of the same entity type for which a theme ordering was specified.

All attribute names must contain no more than 40 characters. They may be composed of nearly any printable characters including alphanumerics, colons, periods, commas, apostrophes, and accented characters. The following table shows the attribute types that are supported:

Attribute Name	Description
INT	Integers are represented with 16 ASCII characters. They are stored as base 10 numbers, right-justified in the field, with leading zeroes to fill the remaining space. The first character

Record Name	Description
VDR	Volume Descriptor Record: contains information about the entire CCOGIF volume, such as its creation date and generating agency.
UFLR	User Fixed Length Record: contains any user data associated with the CCOGIF volume or dataset. It simply contains free-form ASCII data.
DSHR	Data Set Header Record: describes each dataset contained in a CCOGIF volume. It contains some extra-neous information about the data (name, creation date, geographic location, etc.), as well as information pertaining to the interpretation of the data itself (X, Y, and Z data types, topology information, map projection information, etc.).
EMDR	Entity Metadata Record: describes the source and quality of the entity data contained in a dataset. There are one or more of these records following each DSHR record.
DGHR	Data Group Header Record: is the first record of each data group within a CCOGIF dataset. It provides a name for the group, as well as provides counts of how many point, line, and area themes are contained in the group.
DTHR	Data Theme Header Record: describes the contents of a single data theme. It provides information about the entity data itself (entity type, number of entities), as well as a count of user attributes defined on each entity and a calculation of the length of the fixed-length part of each entity in the theme.
ADR	Attribute Descriptor Record: There is one attribute descriptor record in each data theme. It is a variable-length record that describes the names and types of attributes defined on each entity contained in the theme.
EOVR	End Of Volume Record: marks the logical and physical end of the CCOGIF volume. It is always the last record in a CCOGIF file.

All metadata features have a feature type of **CCOGIF_METADATA**. The CCOGIF record described by a metadata feature is reflected by the value of the attribute **ccogif_record_code**. This attribute has one of three or four character record names listed in the above table.

Each metadata feature has a particular set of attributes, depending on the CCOGIF record it represents. The following sections list the attributes for each record type.

Volume Descriptor Record

The Volume Descriptor Record (VDR) is the first record in a CCOGIF file. The FME represents this record as a **CCOGIF_METADATA** feature with the following attributes:

Attribute Name	Description	Type
ccogif_record_code	Record code (constant VDR).	char(4)
ccogif_log_vol_id	Logical volume identifier.	int(16)
ccogif_phys_vol_num	Physical volume number in this logical volume (numbered sequentially from 1).	int(16)
ccogif_vol_cre_date	Volume creation date.	date
ccogif_vol_data_desc	Logical volume data description.	char(128)
ccogif_vol_gen_cntry	Volume generating country.	char(64)
ccogif_vol_gen_agncy	Volume generating agency.	char(64)
ccogif_vol_gen_fclty	Volume generating facility.	char(64)
ccogif_fmt_ctrl_doc_id	Format control document identifier.	char(64)
ccogif_sw_release_id	Software release identifier.	char(64)
ccogif_feat_code_rev	Feature code revision level.	char(64)
ccogif_num_ufl_recs	Number of user fixed length records immediately following this volume descriptor record.	int(16)
ccogif_bytes_prev_rec	Number of bytes left from last logical record of previous physical volume.	int(16)

User Fixed-Length Record

User Fixed-Length Records (UFLRs) provide a place for the user's software to place any ASCII information for its own purpose. Each UFLR contains up to 2044 bytes of user-defined data. There are zero or more UFLRs immediately following each VDR or DSHR in the CCOGIF file. The **ccogif_num_ufl_recs** attribute of the VDR or DSHR feature tell us how many UFLR records to expect.

A single FME feature is used to represent a sequence of user fixed length records. The data from the entire sequence of UFLRs appears concatenated together in a single attribute on the single FME feature.

FME represents the sequence of UFLRs with a CCOGIF_METADATA feature with the following attributes:

Attribute Name	Description	Type
ccogif_record_code	Record code, constant UFLR .	char(4)
ccogif_user_def_data	User-defined data, concatenated from an entire sequence of UFLR records.	char(n)

Data Set Header Record

The Data Set Header Record (DSHR) defines all information common to all entities contained in a single CCOGIF dataset. A CCOGIF volume may contain more than one dataset.

FME represents a DSHR record with a **CCOGIF_METADATA** feature with the following attributes:

Attribute Name	Description	Type
ccogif_record_code	Record code (constant DSHR).	char(4)
ccogif_data_set_name	Dataset name.	char(64)
ccogif_ds_cre_date	Dataset creation date.	date
ccogif_ds_loc_text	Dataset geographic location text.	char(64)
ccogif_related_ds	Reference to other related datasets.	char(64)
ccogif_data_three_dim	Specifies whether data is three-dimensional (3D). If false, Z coordinate is always zero. Legal values are T (true), F (false) and U (unknown).	char(1)
ccogif_pt_to_ln_topo	Specifies whether point-to-line topology exists in a dataset. Legal values are T (true), F (false) and U (unknown).	char(1)
ccogif_ln_to_pt_topo	Specifies whether line-to-point topology exists in a dataset. Legal values are T (true), F (false) and U (unknown).	char(1)
ccogif_colloc_exists	Specifies whether dataset employs line collocation. Legal values are T (true), F (false) and U (unknown).	char(1)
ccogif_ln_to_area_topo	Specifies whether line-to-area topology exists in a dataset. Legal values are T (true), F (false) and U (unknown).	char(1)
ccogif_area_to_ln_topo	Specifies whether area-to-line topology exists in a dataset. Legal values are T (true), F (false) and U (unknown).	char(1)

Attribute Name	Description	Type
ccogif_known_pt_in_area	Specifies whether there is a known point in each area. Legal values are T (true), F (false) and U (unknown).	char(1)
ccogif_attrs_in_entity	Specifies whether attributes are present in entity records. Legal values are T (true), F (false) and U (unknown).	char(1)
ccogif_feat_classes	Ordered list of feature classes (A to K) for the dataset. Blanks are placed for classes not present; e.g., A, D, GH, J.	char(32)
ccogif_num_data_grp	Number of data groups contained in this CCOGIF dataset (n>=1).	int(16)
ccogif_num_ufl_recs	Number of user fixed length records immediately following this dataset header record (n>=0).	int(16)
ccogif_num_emd_recs	Number of entity metadata records that describe the contents of this CCOGIF dataset (n>=1).	int(16)
ccogif_x_data_type, ccogif_y_data_type, ccogif_z_data_type	Data type of x, y, or z coordinate values. Legal values are INT, REAL or DMS.	char(4)
ccogif_x_data_units, ccogif_y_data_units, ccogif_z_data_units	Units in which x, y, or z coordinate values are measured – for example, METRES, METRES ASL.	char(16)
ccogif_z_min_value, ccogif_z_max_value	Z coordinate minimum and maximum values. Interpretation of this attribute depends on the values of ccogif_z_data_type and ccogif_z_data_units.	variable ^a
ccogif_proj_id	Map projection identifier that describes the map projection in which the entity data in the dataset is encoded. See the discussion below this table for more details.	char(4)
ccogif_geod_datum	Name of geodetic datum.	char(16)
ccogif_adj_name	Name of adjustment.	char(16)
ccogif_vert_datum	Name of vertical datum.	char(16)

^aFields marked with a type of variable are either REAL, INTEGER, or DMS depending on the DSHRs x, y, or z data type corresponding to that field.

The `ccogif_x_data_type`, `ccogif_y_data_type`, and `ccogif_z_data_type` tell what numeric format is used to represent x, y, and z coordinate values. `INTEGER` and `REAL` are obvious representations. `DMS` values for x and y coordinate values only are stored as `+ddd mm ss.sss`, where `ddd` is the degrees portion of the number, `mm` is the number of minutes, and `ss.sss` is the number of seconds. `DMS` values are converted by FME to their corresponding numeric or decimal values.

The `DSHR` feature also contains a number of attributes specific to the map projection in which the dataset's entities are expressed. The selection of map projection is made by the `ccogif_proj_id` attribute. It has one of the following values:

Map Projection ID	Projection Name
0100	Latitude/Longitude
0200	Transverse Mercator—Universal Transverse Mercator (UTM) projections are stored with this ID as well
0203	Mercator
0300	Lambert Conformal
0400	Stereographic
0500	Polyconic

The attributes for each map projection type are listed in the following sections.

Latitude/Longitude Project Parameters

Datasets in the Latitude/Longitude projection have the following attributes defined on their `DSHR` feature:

Attribute Name	Description	Type
<code>ccogif_proj_id</code>	Map projection identifier, constant <code>0100</code> .	char(4)
<code>ccogif_proj_name</code>	Map projection name, constant <code>LATITUDE/LONGITUDE</code> .	char(32)
<code>ccogif_proj_origin_x</code> , <code>ccogif_proj_origin_y</code>	Longitude/latitude origin for x,y coordinates.	DMS
<code>ccogif_proj_num_bnd_crd</code>	Number of coordinate pairs that form a bounding polygon for this dataset (<code>0 <= n <= 12</code>).	int(16)
<code>ccogif_proj_bnd_crd{n}.x</code> , <code>ccogif_proj_bnd_crd{n}.y</code>	Coordinate #n of bounding polygon (<code>0 <= n < config_num_bnd_crd</code>).	DMS

Transverse Mercator Projection Parameters

Datasets in the Transverse Mercator projection have the following attributes defined on their `DSHR` feature:

Attribute Name	Description	Type
ccogif_proj_id	Map projection identifier, constant 0200 .	char(4)
ccogif_proj_name	Map projection name, constant TRANSVERSE MERCATOR .	char(32)
ccogif_proj_cent_merid	Central meridian.	DMS
ccogif_proj_zone_width	Zone width.	DMS
ccogif_proj_sphd_name	Spheroid name.	char(20)
ccogif_proj_semi_major	Semi-major axis.	real(16)
ccogif_proj_semi_minor	Semi-minor axis.	real(16)
ccogif_proj_eccent	Eccentricity.	real(16)
ccogif_proj_scl_fact	Scale factor.	real(16)
ccogif_proj_false_east, ccogif_proj_fals_north	False Easting/Northing.	real(16)
ccogif_proj_zone	Zone number.	int(16)
ccogif_proj_orig_east, ccogif_proj_orig_north	Origin (easting, northing).	variable ^a
ccogif_proj_num_bnd_crd	Number of coordinate pairs that form a bounding polygon for this dataset (0>=n>=12).	int(16)
ccogif_proj_bnd_crd{n}.x, ccogif_proj_bnd_crd{n}.y	Coordinate #n of bounding polygon (0>=n>config_num_bnd_crd).	variable ^a

Mercator Projection Parameters

Datasets in the Mercator projection have the following attributes defined on their **DSHR** feature:

Attribute Name	Description	Type
ccogif_proj_id	Map projection identifier, constant 0203 .	char(4)
ccogif_proj_name	Map projection name, constant MERCATOR .	char(32)
ccogif_proj_mid_lat	Mid latitude.	DMS

^aFields marked with a type of variable are either **REAL**, **INTEGER**, or **DMS** depending on the DSHR's **x**, **y**, or **z** data type corresponding to that field.

Attribute Name	Description	Type
ccogif_proj_sphd_name	Spheroid name.	char(20)
ccogif_proj_semi_major	Semi-major axis.	real(16)
ccogif_proj_semi_minor	Semi-minor axis.	real(16)
ccogif_proj_eccent	Eccentricity.	real(16)
ccogif_proj_orig_east, ccogif_proj_orig_north	Origin (easting, northing).	variable ^a
ccogif_proj_num_bnd_crd	Number of coordinate pairs that form a bounding polygon for this dataset (0>=n>=12).	int(16)
ccogif_proj_bnd_crd{n}.x, ccogif_proj_bnd_crd{n}.y	Coordinate #n of bounding polygon (0>=n>config_num_bnd_crd).	variable ^a

Lambert Conformal Projection Parameters

Datasets in the Mercator projection have the following attributes defined on their **DSHR** feature:

Attribute Name	Description	Type
ccogif_proj_id	Map projection identifier, constant 0300.	char(4)
ccogif_proj_name	Map projection name, constant LAMBERT CONFORMAL.	char(32)
ccogif_proj_frst_scl_par	First scaling parallel.	DMS
ccogif_proj_secnd_scl_par	Second scaling parallel.	DMS
ccogif_proj_sphd_name	Spheroid name.	char(20)
ccogif_proj_semi_major	Semi-major axis.	real(16)
ccogif_proj_semi_minor	Semi-minor axis.	real(16)
ccogif_proj_eccent	Eccentricity.	real(16)
ccogif_proj_orig_east, ccogif_proj_orig_north	Origin (easting, northing).	variable ^b
ccogif_proj_num_bnd_crd	Number of coordinate pairs that form a bounding polygon for this dataset (0>=n>=12).	int(16)

^aFields marked with a type of variable are either **REAL**, **INTEGER**, or **DMS** depending on the DSHR's **x**, **y**, or **z** data type corresponding to that field.

^bFields marked with a type of variable are either **REAL**, **INTEGER**, or **DMS** depending on the DSHR's **x**, **y**, or **z** data type corresponding to that field.

Attribute Name	Description	Type
ccogif_proj_bnd_crd{n}.x, ccogif_proj_bnd_crd{n}.y	Coordinate #n of bounding polygon (0>=n>config_num_bnd_crd).	variable ^a

Stereographic Projection Parameters

Datasets in the Mercator projection will have the following attributes defined on their **DSHR** feature:

Attribute Name	Description	Type
ccogif_proj_id	Map projection identifier, constant 0400	char(4)
ccogif_proj_name	Map projection name, constant STEREOGRAPHIC	char(32)
ccogif_proj_scale_lat	Scaling latitude	DMS
ccogif_proj_sphd_name	Spheroid name	char(20)
ccogif_proj_semi_major	Semi-major axis	real(16)
ccogif_proj_semi_minor	Semi-minor axis	real(16)
ccogif_proj_eccent	Eccentricity	real(16)
ccogif_proj_orig_east, ccogif_proj_orig_north	Origin (easting, northing)	variable ^a
ccogif_proj_num_bnd_crd	Number of coordinate pairs that form a bounding polygon for this dataset (0>=n>=12)	int(16)
ccogif_proj_bnd_crd{n}.x, ccogif_proj_bnd_crd{n}.y	Coordinate #n of bounding polygon (0>=n>config_num_bnd_crd)	variable ^a

Polyconic Projection Parameters

Datasets in the Mercator projection have the following attributes defined on their **DSHR** feature:

Attribute Name	Description	Type
ccogif_proj_id	Map projection identifier, constant 0500 .	char(4)
ccogif_proj_name	Map projection name, constant POLY-CONIC .	char(32)
ccogif_proj_cent_merid	Central meridian.	DMS

^aFields marked with a type of variable are either **REAL**, **INTEGER**, or **DMS** depending on the DSHRs **x**, **y**, or **z** data type corresponding to that field.

Attribute Name	Description	Type
ccogif_proj_sphd_name	Spheroid name.	char(20)
ccogif_proj_semi_major	Semi-major axis.	real(16)
ccogif_proj_semi_minor	Semi-minor axis.	real(16)
ccogif_proj_eccent	Eccentricity.	real(16)
ccogif_proj_orig_east, ccogif_proj_orig_north	Origin (easting, northing).	variable ^a
ccogif_proj_num_bnd_crd	Number of coordinate pairs that form a bounding polygon for this dataset (0>=n>=12).	int(16)
ccogif_proj_bnd_crd{n}.x, ccogif_proj_bnd_crd{n}.y	Coordinate #n of bounding polygon (0>=n>config_num_bnd_crd).	variable ^a

Entity Metadata Record

The Entity Metadata Record (EMDR) describes the source and quality of the data contained in the CCOGIF dataset. There may be multiple EMDRs in a single CCOGIF dataset if there are varying sources and quality of data in the dataset. The entity data records in the CCOGIF dataset refer back to these entity metadata records by ID.

The FME represents an EMDR with a **CCOGIF_METADATA** feature with the following attributes:

Attribute Name	Description	Type
ccogif_record_code	Record code, constant EMDR .	char(4)
ccogif_meta_data_id	Metadata ID number.	int(16)
ccogif_data_gen_agncy	Data generating agency.	char(64)
ccogif_capture_method	Method of data capture or revision.	char(64)
ccogif_col_instrmt	Type of collecting instrument.	char(64)
ccogif_src_mat_type	Type of source material.	char(64)
ccogif_src_mat_scale	Scale of source material.	char(64)
ccogif_src_mat_date	Date of source material.	date
ccogif_fld_comp_date	Date of field completion.	date
ccogif_data_captr_date	Date of data capture or revision.	date
ccogif_src_mat_spec	Reference to specification document on source material and collection or revision methods.	char(192)

^aFields marked with a type of variable are either **REAL**, **INTEGER**, or **DMS** depending on the DSHR's **x**, **y**, or **z** data type corresponding to that field.

Attribute Name	Description	Type
ccogif_feat_code_spec	Reference to specification document on feature coding and attribute assigning procedures.	char(192)
ccogif_data_struct_spec	Reference to specification document on data structuring process.	char(192)
ccogif_qual_ctrl_spec	Reference to specification document on quality control procedures.	char(192)
ccogif_trans_gen_spec	Reference to specification document on transformations and generalization procedures.	char(192)
ccogif_field_cpltn_spec	Reference to specification document on field completion procedures.	char(192)
ccogif_acc_det_proc_spec	Reference to specification document on accuracy determination procedures.	char(192)
ccogif_data_resolution	Resolution of the data.	char(64)
ccogif_x_accuracy, ccogif_y_accuracy, ccogif_z_accuracy	X, Y, Z positional accuracy of the data.	real(16)

Data Group Header Record

Each data group within a CCOGIF dataset starts with a Data Group Header Record (DGHR). This record defines the name of the data group and tells how many point, line, and area themes are contained in the group.

The FME represents a **DGHR** with a **CCOGIF_METADATA** feature with the following attributes:

Attribute Name	Description	Type
ccogif_record_code	Record code, constant DGHR .	char(4)
ccogif_data_group_name	Name of data group.	char(64)
ccogif_num_point_themes, ccogif_num_line_themes, ccogif_num_area_themes	Number of (point, line, area) themes contained in this data group.	int(16)

Data Theme Header Record

Each theme in a data group starts with a Data Theme Header Record (DTHR). This record defines the entity type (point, line, area) contained in the theme, the number of attributes defined on each entity in the theme, and the length of the fixed length entity records within the theme.

All entities within a particular theme must be of the same entity type and must have the same set of attributes.

FME represents a **DTHR** with a **CCOGIF_METADATA** feature with the following attributes:

Attribute Name	Description	Type
ccogif_record_code	Record code, constant DTHR .	char(4)
ccogif_entity_type	Type of entity in theme (POINT , LINE , AREA).	char(8)
ccogif_num_entities	Number of entities in this data theme.	int(16)
ccogif_num_attr_desc	Number of attributes defined on each entity of the theme.	int(16)
ccogif_fixed_len_bytes	Length of the fixed length portion of each entity in the theme.	int(16)

Attribute Descriptor Record

All entities within a given data theme have the same set of attributes defined on them. The Attribute Descriptor Record (ADR) lists the name and type of each attribute defined on the entities of the current data theme. The number of attributes defined on a theme is specified in the **ccogif_num_attr_desc** attribute of the **DTHR**.

The FME represents an **ADR** with a **CCOGIF_METADATA** feature that has the following attributes:

Attribute Name	Description	Type
ccogif_record_code	Record code, constant ADR .	char(4)
ccogif_attr{n}_name	Name of attribute #n.	char(40)
ccogif_attr{n}_type	Type of attribute #n (INT , REAL , DMS , CHAR , or DATA).	char(4)
ccogif_attr{n}_len	String length if attribute #n is a CHAR type, otherwise 0 .	int(16)

Entity Features

The entity features are the features that represent the entities of the CCOGIF dataset. When reading CCOGIF data, an entity feature will have a feature type of **<data_group>_<theme_index>**, where:

<data_group> is the name of the data group, taken from the most recent DGHR metadata record. All special characters—such as spaces, colons, etc.—are replaced with underscores.

<theme_index> is the index within the data group of the theme containing the entity. The themes are numbered sequentially within their data group. The first theme in each data group is number **1**.

Entities may be described in a CCOGIF file as a pair of records. Each entity has a fixed length record and, optionally, a variable-length record. The fixed length portion contains information, such as an ID number, a point's location, a line's topological information, a feature code, and attribute values. In other words, the data present for all entities in the theme.

The variable-length portion contains the data that varies in size between entities within a given theme and may or may not be present for a given entity. It contains, for example, a list of lines attached to a point entity, the coordinates defining a line, or the list of identifiers of lines defining an area's boundaries.

FME's entity features combine the contents of the fixed length and variable length records for a particular entity into a single FME feature.

All entity features have the following attributes defined on them:

Attribute Name	Description	Type
ccogif_record_code	Record code, constant PFLR for point entities, LFLR for line entities, AFLR for area entities.	char(4)
ccogif_data_coll_md_ptr	Data collection metadata pointer (reference number of EMDR corresponding to data collection).	int(16)
ccogif_data_rev_md_ptr	Data collection metadata pointer (reference of EMDR corresponding to data revision or validation).	int(16)
ccogif_prim_feat_code	Primary feature code for the entity.	char(12)
ccogif_data_group_name	Name of the data group containing the entity.	char(40)
ccogif_data_theme_id	Index of the data theme within the data group.	int(16)

Each specific type of entity has additional attributes to describe the entity and are listed in the sections that follow.

In addition to the standard attributes, each entity feature also has values for all attributes listed in its data theme's **ADR**.

Point Entity Features

Point entities are represented in the CCOGIF file as a Point Fixed-Length Record (PFLR) and, optionally, a Point Variable-Length Record (PVLRL). The FME combines the contents of these two records into a single feature – the point entity feature.

The geometry of FME's point entity feature is the point's coordinates from the **PFLR**. A Z-value of **-9999** represents an undefined value, so any **PFLRs** that have a Z-value of **-9999** are translated as an (x,y) coordinate instead of an (x,y,z) coordinate.

In addition to the geometry and the attributes common to all entity features (listed in the previous section), point entity features have the following attributes defined:

Attribute Name	Description	Type
ccogif_record_code	Record code, constant PFLR.	char(4)
ccogif_point_id	Point ID number.	int(16)
ccogif_num_lines	Number of lines attached to this point (n>=0).	int(16)
ccogif_orientation	Orientation of point, measured in degrees counterclockwise from the x-axis.	real(16)
ccogif_line_id{n}	Line identifier of nth line attached to this point.	int(16)

Line Entity Features

Line entities are represented in the CCOGIF file as a Line Fixed-Length Record (LFLR) and optionally, a Line Variable-Length Record (LVLR). The FME combines the contents of these two records into a single feature—the line entity feature.

The geometry of FME's line entity feature is the line's coordinates from the LVLR. A Z-value of **-9999** represents an undefined value, so any LVLRs that have Z-values of **-9999** are translated as (x,y) coordinates instead of (x,y,z) coordinates.

In addition to the geometry and attributes common to all entity features, which were listed in the previous section, line entity features have the following attributes defined:

Attribute Name	Description	Type
ccogif_record_code	Record code, constant LFLR.	char(4)
ccogif_line_id	Line ID number.	int(16)
ccogif_num_lines	Number of lines attached to this point (n>=0).	int(16)
ccogif_coll_line_id	ID number of collocated line (0 if not collocated).	int(16)
ccogif_start_node_id	Start node ID number (0 if not defined).	int(16)
ccogif_end_node_id	End node ID number (0 if not defined).	int(16)
ccogif_left_area_id	Left area ID number (0 if not defined).	int(16)
ccogif_right_area_id	Right area ID number (0 if not defined).	int(16)

Area Entity Features

Area entities are represented in the CCOGIF file as an Area Fixed-Length Record (AFLR) and optionally, as an Area Variable-Length Record (AVLR). The FME combines the contents of these two records into a single feature – the area entity feature.

The geometry of FME's area entity feature is the coordinates of a point inside the area, as defined in the AFLR. A Z-value of **-9999** represents an undefined values so any AFLRs that have a Z-value of **-9999** are translated as an (x,y) coordinate instead of an (x,y,z) coordinate. Note that the area entity feature itself does not contain any polygonal geometry. Instead, it contains a list of attributes pointing to the identifiers of the lines that make up the boundary of the area. To form a polygon from CCOGIF data it is necessary to use the *ReferenceFactory*, or a similar factory, in the mapping file to associate the area feature with its polygonal boundaries.

In addition to the geometry and the attributes common to all entity features (listed in the previous section), area entity features have the following attributes defined:

Attribute Name	Description	Type
ccogif_record_code	Record code, constant AFLR.	char(4)
ccogif_area_id	Area ID number.	int(16)
ccogif_num_bnd_lines	Number of lines that form the boundaries	int(16)

Attribute Name	Description	Type
	of this polygon (n>=0).	
ccogif_line_id{n}	Line identifier of nth boundary line for this area.	int(16)

Defining Volume Structure

The contents of a CCOGIF volume follow a firm structure, provided through the use of metadata records. The sequence and contents of these metadata records is crucial to the correctness of a CCOGIF volume. The CCOGIF writer provides a mechanism for the mapping file to explicitly define the required records and their contents.

There are six areas where the CCOGIF writer provides direct control of the generated CCOGIF records:

- Definition and order of data themes within a data group
- Specification of metadata records, such as:
 - the contents of Volume Descriptor Record (VDR)
 - the contents of User Fixed-Length Record (UFLR)
 - the contents of Data Set Header Record (DSHR)
 - the contents of Entity Metadata Record (EMDR)
- Specification of the contents of entity records
- The following sections cover each of these in more detail.

Theme Definition

Geometric entity records are grouped into themes, where all entities within a given theme have the same geometric type – point, line, or area – and the same set of attributes. Each theme written to a CCOGIF file is defined by a CCOGIF **DEF** line. The **DEF** line specifies the name of the group to which the theme belongs, and allows specification of the relative ordering of the themes within the groups. The header records for data groups (DGHR) are written to the output file for each group mentioned on a **DEF** line.

The group name and theme ordering index may be specified explicitly on the **DEF** line with the **CCOGIF_GROUP_NAME** and **CCOGIF_THEME_ORDERING** keywords, or they may be implied by the theme identifier. If the **DEF** line does not have a **CCOGIF_GROUP_NAME**, the theme identifier, or FME feature type, becomes the implied group name. In this case, the **DEF** line actually defines attributes for the group itself and not a particular theme, and therefore must not include any theme ordering or entity type information. The reasons and implications of assigning attributes to a group instead of a theme within the group are discussed below.

If the **DEF** line contains neither an explicit group name or an explicit theme ordering, and the theme identifier is of the form **<groupName>_<number>**, where **<number>** is any integer, then the group name and theme ordering are implied as **<groupName>** and **<number>** respectively.

The geometric entity type for each theme must be provided on the theme's **DEF** line by using the **CCOGIF_THEME_ENTITY_TYPE** keyword. However, it is not necessary for every **DEF** line to have an entity type provided. If neither entity type nor theme ordering information is specified, then the **DEF** line is considered to define a set of attributes for a data group rather than for a data theme.

Strictly speaking, it makes no sense to talk of attributes being assigned to a CCOGIF data group as attributes are assigned to themes within the CCOGIF file. This mechanism, however, provides the ability to essentially define a point, line, and area theme within a single group, with identical sets of attributes. Any features with the specified feature type are written to the appropriate theme—point, line, or area—depending on the geometry type of the feature. Features with aggregate geometry or no geometry at all will not be written.

The themes implied with this mechanism, called generic themes from this point on, may coexist with other themes in a group, making it possible to define a number of themes for a data group as well as to define generic themes for “the other stuff that we want to write but that doesn’t fit into our predefined themes”. If the generic theme mechanism is

used, its **DEF** line must appear in the mapping file before the **DEF** lines for any non-generic themes within that group.

Specifying Metadata Records

The metadata records in a CCOGIF file contain many pieces of information that must be correctly defined for the file to be accurate. Much of this information is for the benefit of human users of the end product and may vary not only from site to site, but from one dataset to another. This information cannot be coded into FME itself, therefore it must be supplied into the mapping file.

The CCOGIF writer expects this information to be passed in the same metadata features that the CCOGIF reader creates. If the writer receives a feature with a feature type of **CCOGIF_METADATA**, it will look at the **ccogif_rec-ord_code** attribute to see which of the metadata features, described in "Metadata Features" on page 243, the feature represents. It extracts from this feature whatever **ccogif_XXX** attributes apply to that particular type of metadata feature and eventually writes the information to the metadata records in the output CCOGIF file.

The order of the incoming metadata features is significant, as they are written out in a similar order to which they are received. Any volume-specific metadata **must** come before the Data Set Header Record (DSHR). Metadata features received after the DSHR appear in the output CCOGIF file as a part of the dataset.

In addition, all metadata features must be presented to the writer before any entity features are presented. The CCOGIF writer needs information from the DSHR in order to write entity data, so it creates a default DSHR if none has been given. Any metadata that comes after the default DSHR has been generated may contradict the default values placed into the DSHR, resulting in an invalid output CCOGIF file.

Several ways to generate the metadata features for the CCOGIF writer are discussed below:

- Use the Multi-Reader as the input reader and use a template CCOGIF file as a source for **CCOGIF_METADATA** features. In other words, specific metadata features may be chosen from this reader, then redirected to the writer to provide attribute values for the corresponding metadata records in the output.
- Store the template's **CCOGIF_METADATA** features in a feature store and use the **RecordingFactory** to inject them into the feature stream. If this method is chosen, it's important to choose the playback mode of **PLAYBACK** instead of **PLAYBACK_AT_END**.
- Use the **CreationFactory** to create the metadata features with all of the required attributes.

Once the features have been generated, they have to be handed to the writer with a feature type of **CCOGIF_METADATA**. This can be accomplished by creating a false theme in any of the groups in the output file such as:

```
CCOGIF_DEF CCOGIF_METADATA
  CCOGIF_GROUP_NAME      "FEATURES"
```

With this definition in place, you can correlate the metadata features to the **CCOGIF_METADATA** feature type of the output format. This correlation must equal all **ccogif_XXX** attributes on the source and target sides for all metadata feature types being correlated. For example:

```
CCOGIF CCOGIF_METADATA \
  ccogif_adj_name          %ccogif_adj_name \
  ccogif_area_to_ln_topo   %ccogif_area_to_ln_topo \
  ccogif_attrs_in_entity   %ccogif_attrs_in_entity \
  ccogif_bytes_prev_rec    %ccogif_bytes_prev_rec \
  ccogif_colloc_exists     %ccogif_colloc_exists \
  ...
  ccogif_z_min_value       %ccogif_z_min_value

SHAPE CCOGIF_METADATA \
  ccogif_adj_name          %ccogif_adj_name \
  ccogif_area_to_ln_topo   %ccogif_area_to_ln_topo \
  ccogif_attrs_in_entity   %ccogif_attrs_in_entity \
  ccogif_bytes_prev_rec    %ccogif_bytes_prev_rec \
  ccogif_colloc_exists     %ccogif_colloc_exists \
  ...
  ccogif_z_min_value       %ccogif_z_min_value
```

The following sections describe ways in which some of the types of metadata features are treated specially by the CCOGIF writer. This special treatment simply ensures that the record exists and has some legal, if not meaningful, default values in place for the **ccogif_XXX** attributes.

Volume Descriptor Record Contents

The Volume Descriptor Record (VDR) must be present in every CCOGIF file. If it is not given to the CCOGIF writer, it will be created with the default attribute values listed in the following table. If a VDR metadata feature is given to the CCOGIF writer and it defines only some of the **ccogif_XXX** attributes that appear in the table below, the default value will be “taken” for those not specified.

Attribute Name	Contents – Default Values
ccogif_log_vol_id	FME-generated CCOGIF dataset
ccogif_phys_vol_num	0
ccogif_vol_cre_date	Current date
ccogif_vol_data_desc	Empty string (“ ”)
ccogif_vol_gen_cntry	Empty string (“ ”)
ccogif_vol_gen_agncy	Empty string (“ ”)
ccogif_vol_gen_fclty	Empty string (“ ”)
ccogif_fmt_ctrl_doc_id	Empty string (“ ”)
ccogif_sw_release_id	Empty string (“ ”)
ccogif_feat_code_rev	Empty string (“ ”)
ccogif_num_ufl_recs	Computed from the amount of user data specified in the volume’s UFLR metadata features
ccogif_bytes_prev_rec	0

The value for the attribute **ccogif_num_ufl_recs** is always filled in automatically by the CCOGIF writer. Its value will be based on the length of the user data passed to the writer via the User Fixed-Length Records’ (UFLR) metadata features.

User Fixed-Length Record Contents

Each **VDR** and **DSHR** record in a CCOGIF file may be immediately followed by zero or more **UFLRs**. When given a sequence of **UFLR** metadata features, the CCOGIF writer will read the **ccogif_user_def_data** attribute of each feature and concatenate their values into one large character string. When it comes time to write out the metadata records, the writer creates as many **UFLRs** as are required to hold the accumulated data. Each **UFLR** can contain up to 2044 bytes of user data.

The placement of the **UFLRs** in the output CCOGIF file depends on where they occur in the sequence of metadata features. If the **UFLR** appears before the **DSHR** feature, or the first entity feature, if no **DSHR** feature is explicitly given, they will be written out immediately following the volume’s **VDR**. In this case, the **VDR**’s **ccogif_num_ufl_recs** are set to specify how many **UFLRs** follow.

If the **UFLR** metadata features appear after the **DSHR** metadata feature, then the **UFLRs** will be written immediately following the **DSHR** record and the **DSHR**’s **ccogif_num_ufl_recs** will be set to specify how many **UFLRs** follow.

Data Set Header Record Contents

Every CCOGIF file may contain one or more datasets.¹ The first record of each dataset in a CCOGIF volume is called the Data Set Header Record (DSHR). It provides information particular to the dataset not only for the human user—it also directs computer applications on how to process the data.

The portions of the **DSHR** that relate to the processing of the data are of specific interest to the CCOGIF writer. When it writes out data within a dataset, it must remain consistent with the dataset characteristics set out in the **DSHR**. The following information is particularly interesting to the writing process.

- **Coordinate data type:** The **DSHR** specifies the data type, **INT**, **REAL** or **DMS**, for each of the x, y, and z coordinate values within the dataset. The CCOGIF writer uses the values of the **DSHR** metadata feature's **ccogif_x_data_type**, **ccogif_y_data_type**, and **ccogif_z_data_type** attributes to format the numerical coordinates correctly.
- **Data set content indicator:** The Data Set Content Indicator (DSCI) is a subrecord of the **DSHR** that tells whether data is 3D, whether there is a known point in each area, and whether various topology information such as, point to line topology, line to point topology, line collocation, line to area topology, and area to line topology, is present in the entity attributes. FME's CCOGIF writer currently does not generate any topology information, however it passes along any that has been added to geometric entity features which have been given to it.
- **Coordinate system and map projection:** The correct coordinate system for the output CCOGIF dataset must be specified in a **DSHR** metadata feature, as this information is not yet tied in to FME's coordinate system manager. If no coordinate system information is provided to the CCOGIF writer, it will arbitrarily choose a Universal Transverse Mercator (UTM) zone 18 projection, which is most likely not what is wanted.

The following table lists the default values for all attributes in the **DSHR**. These default values will be written out for any attributes not mentioned in "any" **DSHR** metadata feature given to the writer.

Attribute Name	Contents – Default Value
ccogif_data_set_name	FME-generated CCOGIF dataset
ccogif_ds_cre_date	Current date
ccogif_ds_loc_text	Empty string (" ")
ccogif_related_ds	Empty string (" ")
ccogif_data_three_dim	Depends on whether first entity feature written is two- or three-dimensional.
ccogif_pt_to_ln_topo	F for False
ccogif_ln_to_pt_topo	F for False
ccogif_colloc_exists	F for False
ccogif_ln_to_area_topo	F for False
ccogif_area_to_ln_topo	T for True
ccogif_known_pt_in_area	T for True
ccogif_attrs_in_entity	T for True
ccogif_feat_classes	Empty string (" ")

¹The FME's CCOGIF writer currently supports only a single dataset.

Attribute Name	Contents – Default Value
ccogif_num_data_grp	Number of data groups written to this CCOGIF dataset.
ccogif_num_ufl_recs	Computed from the amount of user data specified on the UFLR metadata records.
ccogif_num_emd_recs	Number of EMDRS to write to the dataset.
ccogif_x_data_type, ccogif_y_data_type, ccogif_z_data_type	REAL
ccogif_x_data_units, ccogif_y_data_units, ccogif_z_data_units	METRES for x and y, METRES ASL for z
ccogif_z_min_value, ccogif_z_max_value	If the entity data is 3D, these are the actual minimum and maximum elevations. Otherwise, they are left blank.
ccogif_proj_id	Defaults to UTM zone 18, so the projection ID will be Transverse Mercator (0200)
ccogif_geod_datum	Name of geodetic datum
ccogif_adj_name	Name of adjustment
ccogif_vert_datum	Name of vertical datum

The **ccogif_num_ufl_recs** attribute on the **DSHR** is completely dependent on the amount of user data passed to the CCOGIF writer in **UFLR** metadata features and is always overwritten by the writer.

The FME defaults to a map projection of UTM zone 18. The following table presents the default values for the projection-related attributes for the **DSHR**.

Attribute Name	Content – Default Value
ccogif_proj_id	Constant: 0200
ccogif_proj_name	Constant: TRANSVERSE MERCATOR
ccogif_proj_cent_merid	Constant: 75
ccogif_proj_zone_width	Constant: 6
ccogif_proj_sphd_name	GRS 80
ccogif_proj_semi_major	Constant: 6.378137E+06
ccogif_proj_semi_minor	Constant: 6.35675231E+06
ccogif_proj_eccent	Constant: 6.694380070E-03
ccogif_proj_scl_fact	Constant: 0.9996

Attribute Name	Content – Default Value
ccogif_proj_false_east, ccogif_proj_fals_north	Constant: 500000 east, 0 north
ccogif_proj_zone	Constant: 18
ccogif_proj_orig_east, ccogif_proj_orig_north	Constant: (0,0)
ccogif_proj_num_bnd_crd	Constant: 0
ccogif_proj_bnd_crd{n}.x, ccogif_proj_bnd_crd{n}.y	Empty string (" ")

It is important to note that the CCOGIF writer requires the **DSHR** information (especially the x,y, and z data types) before it starts to write entity data to the CCOGIF file.

Entity Metadata Record Contents

Each dataset requires at least one Entity Metadata Record (EMDR). The entity records may reference two or three **EMDRs**.

The CCOGIF writer requires **EMDR** metadata features to define meaningful contents for the **EMDRs**. If no **EMDR** metadata features are given to the writer, a single **EMDR** will be written to the output CCOGIF file with all attributes given default values from the table below. The default attribute values are also used to fill in any values of attributes not present in the **EMDR** metadata features passed in.

Attribute Name	Description
ccogif_meta_data_id	Constant: 0
ccogif_data_gen_agncy	Empty string (" ")
ccogif_capture_method	Empty string (" ")
ccogif_col_instrmt	Empty string (" ")
ccogif_src_mat_type	Empty string (" ")
ccogif_src_mat_scale	Empty string (" ")
ccogif_src_mat_date	Current date
ccogif_fld_comp_date	Current date
ccogif_data_captr_date	Current date
ccogif_src_mat_spec	Empty string (" ")
ccogif_feat_code_spec	Empty string (" ")
ccogif_data_struc_spec	Empty string (" ")
ccogif_qual_ctrl_spec	Empty string (" ")
ccogif_trans_gen_spec	Empty string (" ")
ccogif_field_cpltn_spec	Empty string (" ")

Attribute Name	Description
ccogif_acc_det_proc_spec	Empty string (" ")
ccogif_data_resolution	Empty string (" ")
ccogif_x_accuracy, ccogif_y_accuracy, ccogif_z_accuracy	Constant: 0

The `ccogif_data_coll_md_ptr` and `ccogif_data_rev_md_ptr` attributes on all output entity features must be given an explicit value in the mapping file to ensure they are meaningful.

Entity Record Contents

The entity records written out by the CCOGIF mirror those obtained from the reader. The `ccogif_XXX` attributes must be defined to be meaningful before the feature is written to the output file. If a feature is passed into the writer with an attribute named `"ccogif_record_code"`, the value of that attribute is inspected to see if it contains one of the values: `ccogif_point`, `ccogif_line`, or `ccogif_area`. If this attribute does not exist, the geometry type of the feature is used to determine which type of entity is to represent the feature.

No topology is normally created by the CCOGIF writer when writing entities to the CCOGIF file, except when polygon data is written to a area themes. In this case, a coverage is computed by intersecting the boundaries and holes of all polygon and donut features written to the group. The coordinates for the lines delineating the resulting areas are written out as LFLR records to the group's "generic" linear data theme, and the AFLR records are written to whatever theme the original polygons were directed at.

If some other topology is required, it can be defined through other means (that is, elsewhere in the mapping file or by an external tool) and presented to the writer as `ccogif_XXX` attributes on the entity features. This would require, however, that every aspect of the CCOGIF entity, including the entity ID and any other internal references, be correctly defined on the feature before it makes its way to the CCOGIF writer.

Generated Mapping Files

In CCOGIF files, geometric entities are grouped by geographic area, then further grouped according to attributes of the data itself such as, data themes with common geometric entity types and sets of attributes. This is very different from the conceptual divisions between data entities that typically must rely on the content of the primary feature code to provide notion similar to FME's feature types. The interpretation of a feature type requires knowledge of the conventions by which the data was encoded in the CCOGIF file.

Tip: Geomatics Canada's document titled "Conversion of NTDB Data into CCOGIF Format" provides an example of such a set of convention.

Without knowledge of the underlying conventions, it is very difficult to automatically generate a single mapping file that works with more than one input file. The definitions of the themes within the groups just isn't consistent enough.

To overcome this obstacle, FME can generate two different kinds of mapping files:

- The first is a *generic* mapping file that extracts all of the information it can from the features, then groups them into FME feature types based on the data theme and data group. When run, this mapping file provides a very simple representation of the data in the output format without regard to any specific set of conventions.
- The second type of mapping file which may be generated takes into account the representation of the National Topographic Data Base (NTDB) data in the CCOGIF file and is referred to as a *profile-specific* mapping file.

The term *profile* is used to refer to a set of file, feature, and attribute naming conventions used to store NTDB in another, that is non-CCOGIF, format. Geomatics Canada has three such profiles, each designed to embody NTDB data within the characteristics and limitations of a particular file format. The three profiles are for ASCII Ungenerate (ARCGEN), MIF/MID, and DXF. The FME provides a way to generate mapping files to write CCOGIF data in another format, generally following the conventions of any of these profiles.

Note: The target format does not have to be the same as the format for which the profile was defined.

The resulting files do not exactly conform to the profile, due to differences in data format and to the way in which mapping file generation works within FME. However, the resulting data files are generally much closer to what you would want than those that a generic mapping file would yield. Manual editing of the mapping files can, of course, bring it much closer.

The advantage of the profile-specific mapping files is that the knowledge of the conventions for storing the NTDB in the source and destination formats is stored in the mapping file. Therefore a single mapping file may be used for a whole series of mapsheets, whereas a generic mapping file would only be applicable to a single CCOGIF file.

The disadvantage to the profile-specific mapping file is that the actual generation process needs a few parameters about the input mapsheets. This requires some knowledge of the data in order to generate the mapping file. In addition, the generated mapping file must be used with NTDB data that is consistent with the parameters with which the mapping file was generated.

The following sections describe the process and application of the two kinds of mapping files in greater detail.

Generic Mapping Files

The generic mapping files are useful for a “one-off” translation of a CCOGIF file to another format. It will translate all of the geometric entities in the file, along with their attributes, and perform simple polygon construction with the area entities. This sort of translation is useful to quickly determine what kind of data was stored in the CCOGIF file or to create a starting point for a hand-coded mapping file.

When a generic mapping file is used, an FME feature is generated for each geometric entity. The features generated have feature types of `<groupName>_<themeIndex>`, where `<groupName>` is the name of the data group that contains the entity and `<themeIndex>` is the position of the entity’s data theme within all of the group’s themes. Because the nature of CCOGIF makes it unlikely that two CCOGIF files could have the same group and theme structure, a mapping file generated from the contents of a given CCOGIF file should only be used to translate that file.

Profile-Specific Mapping Files

Geomatics Canada has defined conventions for storing NTDB data in four different formats:

- CCOGIF
- ASCII Ungenerate, also known as ArcInfo Generate or ARCGEN
- MID/MIF
- DXF

We refer to each of these as a profile.

The published profiles define conventions for attribute naming, file naming, file composition—for example, organized by NTDB theme versus entity name—and rules for defining the specific attributes’ values. The FME has facilities for generating mapping files that translate CCOGIF into any FME-supported format, closely adhering to one of these three profiles:

- ARCGEN
- MID/MIF
- DXF

These profiles are addressed in greater detail following this discussion on profile-specific mapping files.

Aside from choice of profile, the generated mapping file depends on the following three parameters:

- **Choice of Language (English or French):** NTDB data encoded into CCOGIF contains both French and English group names and attribute names. A mapping file is configured to choose which language is used on the output file to name output feature types and attribute names.
- **Choice of NTDB Revision (2 or 3):** NTDB data in CCOGIF follows either the revision 2 or 3 standard. Since this information is not made available to the mapping file generation process, the user must specify it at the time of mapping file generation. A mapping file generated to process mapsheets from one revision cannot be used to process mapsheets from another revision.

- **Choice of Scale (50k or 250k):** NTDB data can contain information for a 50k or a 250k mapsheet. The user must select which scale of data a mapping file is to work with at the time of mapping file generation. A mapping file generated to process one scale of data may not be used to process a mapsheet from another scale because the set of groups and attributes differ slightly.

These parameters are supplied to FME mapping file generation process using the macros **NTDB_Language**, **NTDB_Version**, and **NTDB_Scale**. When generating a mapping file from the command line, the parameters would be specified something similar to the following command. As no CCOGIF input file is required for a profile-specific mapping file, the word **unused** is given.

```
fme generate ntdbcg shape unused mymapping.fme --NTDB_Language French --NTDB_Version 3 --NTDB_Scale 50k
```

Once a profile-specific mapping file has been generated, it may be stored (for example, in the FME gallery) to be used later. It is not necessary to generate a profile-specific mapping file each time a translation is performed.

The following sections describe the specifics of each of the three profiles in more detail.

ASCII Ungenerate (ARCGEN) Profile

The ASCII Ungenerate profile is specified in the Geomatics Canada document titled "*Conversion of NTDB Edition 3 Data into ASCII Ungenerate Format*". This profile has the following properties:

- A separate output file is generated for each entity and geometric representation such as, point, line, area.
- File names have a maximum of eight characters. The first seven are the seven-character identifier for the theme—for example, **BATIMEN**, **BUILDIN**, **CHEMINE**, **CHIMNEY**—followed by a single character for the entity type—**P**, **L** or **A**.
- Point data is stored in a file with the extension **.pts**, lines in a file with the extension **.lin**, and areas in a pair of files—a **.lin** file for the boundary and a **.pts** file for the centroid.
- The National Topographic System (NTS) mapsheet number; for example, **031h01** is used to name a directory that contains the subdirectories **points**, **lines**, and **areas**.
- Attributes are stored in a comma-separated value (CSV) file in the same directory as the corresponding geometry data.
- Each attribute file contains a minimum set of attributes: **identifier**, **entity_name**, **code_gener**, **code_expli**, **ATG**, **ATZ**, **ATE**, **accuracy** (**precision** in French), and **angle** is used for point entities only.

Some of these conventions are difficult to follow with an automatically generated mapping file, especially considering the variety of output formats available. Even for ARCGEN output, however, FME cannot completely adhere to these rules without involving manual editing of the generated mapping file.

The FME's approximation to the above conventions are as follows:

- Target dataset is specified by the user at run-time to be the NTS map number. For many formats, this is a directory that contains a separate file for each feature type, or entity file name. Other formats are written to a single file, with different layers or levels, or whatever the target format's terminology is, for the entity files.
- Feature type names are the same eight-character name mentioned in the specification. The seven-digit entity name is determined by looking up the generic code in a predefined tables. Some formats tack on a suffix, such as **_arc** or **_point** to the entity name. The way mapping file generation works in FME, this is unavoidable however, it can be removed by hand once the mapping file has been generated.
- No subdirectories are created in the target directory for **points**, **lines**, and **areas**.
- If the target format were ARCGEN, the file names will all have **.gen** extensions, instead of **.pts** and **.lin**, and no CSV files will be created.
- When possible with the choice of output formats, the attributes are defined as described above. Additional attributes take either the English or French name of the corresponding CCOGIF attributes, depending on the setting of **NTDB_Language**.

To generate a mapping file for the ARCGEN profile, a source format specification of **ntdbcg**, which is an abbreviation of **NTDB CCOGIF to Generate**, is used. An example of how this is written is:

fme generate ntdbcg ...

MID/MIF Profile

The MapInfo Data Interchange Format (MID/MIF) profile is specified in the Geomatics Canada document titled *Conversion of NTDB Edition 3 Data into MID/MIF Format*. This profile has the following properties:

- NTDB entities are written to output files organized by theme. The name of the output file combines the NTS map-sheet number with the theme abbreviation. The table below summarizes the list of themes and their abbreviations.

NTS Mapsheet Themes, Abbreviations, and Numbers

Theme Name	Abbreviation	Theme number
Designated areas	AD	0
Roads	CH	1
Man-made features	CO	2
Relief and landform	FO	3
General	GE	4
Hydrography	HD	5
Hypsography	HP	6
Power network	RE	7
Rail Network	RF	8
Road network	RR	9
Water saturated soils	SS	10
Toponymy	TO	11
Vegetation	VE	12

- Each dataset in a physical volume occupies a directory identified by the NTS number.
- The output coordinate system for NTDB is a UTM system with a NAD83 datum, coordinates in metres, and a scale factor of 0.9996.
- The output file contains at a minimum the following attributes:
 - CODE (explicit code)
 - ATTF
 - ELEVATION
 - ORIENTATION
 - ATV1_ACCURACY
- Other attributes are named **ATFn_<attribute_name>** and **ATVn_<attribute_name>**.
- A MapInfo symbol table is used to represent explicit codes.

Some of these conventions are difficult to follow with an automatically generated mapping file, especially considering the variety of output formats available. Even for MID/MIF output, however, FME cannot completely adhere to all conventions without involving manual editing of the generated mapping file.

The FME's approximation to the above conventions are as follows:

- The profile specification calls for output files that correspond to FME feature types to have names including the NTS map number. Unfortunately, the FME mapping file generation process cannot use a variable name for the output feature types, therefore it generates all mapping files with output feature types of **NTSNUM_<theme-Abbrev>**, where **<themeAbbrev>** is a theme abbreviation from *NTS Mapsheet Themes, Abbreviations, and Numbers*. It is necessary to modify the generated mapping file to include the map number as a part of the output feature type names.
- Where possible with the choice of output formats, the attributes are defined as described above. Additional attributes take either the English or French name of the corresponding CCOGIF attributes, depending on the setting of **NTDB_Language**.

To generate a mapping file for the MID/MIF profile, a source format specification of **ntdbcm**, which is an abbreviation of **NTDB CCOGIF to MID/MIF**, is used. An example of how this is written is:

```
fme generate ntdbcm ...
```

DXF Profile

The DXF profile is specified in the Geomatics Canada document titled "*Conversion of NTDB Edition 3 Data into DXF Format*". This profile has the following properties:

- The data is written to a DXF file for each theme. The themes are the same thirteen themes as those used for the MID/MIF profile.
- The file names for the theme files are **<nts><abbrev>.dxf** where **<nts>** is the NTS map number and **<abbrev>** is the lower-case equivalent of the theme abbreviation listed in the above-mentioned table.
- Entities are stored in layers named **<entityName>_<explicitCode>** where **<entityName>** is the first 11 or fewer characters of the NTDB entity name and **<explicitCode>** is the explicit code of the entity.
- Fixed attributes—**ATFn_<attrName>**—are not stored with the features. Their values are implied by the explicit code.
- Except for toponymy, variable attributes are not transferred to the DXF features.
- There is no area under DXF.

Some of these conventions are difficult to follow with an automatically generated mapping file, especially considering the variety of output formats available. Even for DXF output, however, FME cannot completely adhere to these rules without involving manual editing of the generated mapping file.

The FME's approximation to the above conventions are given here:

- FME considers the target dataset of a DXF file to be the file itself, so it is not possible to generate a number of themes' output files from a single run of FME. To accomplish this, you would have to run the same CCOGIF file through the mapping file for each desired themes. Refer to the discussion under the heading *Theme Selection* for more details.
- The profile spec calls for output files that correspond to FME feature types to have names including the NTS map number. Unfortunately, FME mapping file generation process cannot use a variable name for the output feature types, therefore it generates all mapping files with output feature types of **NTSNUM_<themeAbbrev>**, where **<themeAbbrev>** is a theme abbreviation from *NTS Mapsheet Themes, Abbreviations, and Numbers*. It will be necessary to modify the generated mapping file to include the map number as a part of the output feature type names.
- When possible with the choice of output formats, the attributes are defined as described above. Additional attributes will take either the English or French name of the corresponding CCOGIF attributes, depending on the setting of **NTDB_Language**.

To generate a mapping file for the DXF profile, a source format specification of **ntdbcd**, an abbreviation of **NTDB CCOGIF to DXF**, is used. An example of how this is written is:

```
fme generate ntdbcd ...
```

Run-Time Options

Language Selection (Generic)

Normally, a generic mapping file names groups and attributes by some combination of their French and English names. The generated generic mapping file contains a few lines that may be uncommented to specify that the output files should contain only the French or English data group and attribute names.

Generating Metadata Report (Revision 2)

When a profile-specific mapping file is generated for an NTDB revision 2 CCOGIF file, it generates a report of the metadata in the file. When running the mapping file, the macro **MetadataReportFilename** must be defined. It contains the name of a file where the report is written. If a file already exists with this name, it will be overwritten with the report.

Tip: The advanced user may be interested to know that the actual generation of the report is performed by including the file `$(FME_HOME)/metafile/ntdbv2Report.fmi` into the mapping file.

Profile-Specific Theme Selection

By default, the profile-specific mapping files export all entity data from the input CCOGIF file to the output file. Often you only want to extract a single theme or a set of themes.

This may be performed by specifying a value for the **NTDB_SelectedThemes** macro when running the mapping file. This macro contains a comma-separated list of themes to be exported. If the macro is empty as it is by default, all themes are exported. The themes are specified either by the number or abbreviation in **NTS Mapsheet Themes, Abbreviations, and Numbers**.

Known Mapping File Issues

When generating mapping files to write to some formats, FME automatically appends geometry type names to the output feature types. Generally, this is a necessary practice for mapping file generation and cannot be overridden. The only ways around this is one of these approaches:

- to modify the mapping file after it has been generated
- to rename the files after the translation has completed

The first approach is more prudent for profile-specific mapping files, as they are likely to be used several times. The modified mapping files can be stored in the FME gallery for future use.

CITS Data Transfer Format (QLF) Reader/Writer

Format Notes: This format is not supported by FME Base Edition.

The Centre for Topographic Information Sherbrooke (CITS) Data Transfer Format (QLF) Reader/Writer allows FME to read and write QLF import and export files. The QLF is a published ASCII format used by CITS for import and export.

Overview

QLF files store both feature geometry and attribution. A QLF file has the following file name extension:

File Name Extension	Contents
.qlf	Vector geometric data
.qlf.gz	Same as above but in compressed gzip format.

The extension is added to the basename of the QLF file. Optionally adding .gz to the extension will output a compressed gzip QLF format file; conversely, the reader can directly read files with the extension .qlf.gz.

The QLF reader and writer supports the storage of point, line, and polygon geometric data in .qlf files. The QLF format also stores features with no geometry. Features having no geometry are referred to as having a geometry of *none*.

QLF Quick Facts

Format Type Identifier	QLF
Reader/Writer	Both
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	qlf_record type
Typical File Extensions	.qlf, .qlf.gz
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	No
Transaction Support	No
Geometry Type	qlf_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	yes		z values	yes
none	no			

Reader Overview

The QLF reader extracts features from a file one at a time, and passes them on to the rest of the FME for further processing. The reader finishes when it reaches the end of the file.

Reader Directives

The directives processed by the QLF reader are listed below. The suffixes shown are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the QLF reader is **QLF**.

DATASET

The value for this keyword is the file path of the QLF file to be read.

Required/Optional

Required

Mapping File Syntax

```
QLF_DATASET /usr/data/qlf/qlffile.qlf
```

* Workbench Parameter

Source QLF File(s)

DEF

Each QLF file may optionally be defined before it is read. The definition specifies the base name of the file, and the names and the types of all attributes. The syntax of a QLF **DEF** line is:

```
<ReaderKeyword>_DEF <baseName> \
[<attrName> <attrType>]+
```

The basename specified on the QLF **DEF** lines is constructed by using either the file name without the extension specified by the **DATASET** keyword or **qlf_record** (used only when QLF is the source).

QLF files require at least one attribute to be defined. The attribute definition given must match the definition of the file being read. If it does not, translation is halted and the true definition of the QLF file's attributes gets logged to the log file.

The following table shows the attribute types supported.

Field Type	Description
------------	-------------

char(<width>)	Character fields store fixed-length strings. The width parameter controls the maximum number of characters that can be stored by the field. No padding is required for strings shorter than this width.
date	Date fields store dates as character strings with the format YYYYMMDD.
number(<width>,<decimals>)	Number fields store single and double precision floating point values. The width parameter is the total number of characters allocated to the field, including the decimal point. The decimals parameter controls the precision of the data and is the number of digits to the right of the decimal.
logical	Logical fields store TRUE/FALSE data. Data read or written from and to such fields must always have a value of either true or false.

The following mapping file fragment defines a QLF file def line when QLF is the source file format.

```

QLF_DEF qlf_record \
  F1 char(20) \
  F2 char(20) \
  F3 char(20) \
  F4 char(20) \
  F5 char(20) \
  F6 char(20) \
  F7 char(20) \
  F8 char(20) \
  F9 char(20)

```

Required/Optional

Required

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the `mitab.dll` in the FME home directory to `mapinfo.dll`.

The syntax of the `MAPINFO_SEARCH_ENVELOPE` directive is:

```
MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

```
MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The QLF writer creates and writes feature data to a QLF file specified by the **DATASET** keyword. As with the reader, the directory must exist before the translation occurs. Any existing QLF files in the directory are overwritten with the new feature data. Only one QLF file can be written during a single FME session. Optionally a **.prj** file will also be written if the the coordinate system's (projection) information is available. Output **.prj** files comply with ESRI's shape format projection file specification.

Writer Directives

The directives that are processed by the QLF writer are listed below. The suffixes shown are prefixed by the current **<writerkeyword>_** in a mapping file. By default, the **<writerkeyword>** for the QLF writer is **QLF**.

Note:

By default, the QLF writer will write the coordinates with 15 digits of precision. If this is not desirable or it is causing problems, then the precision can be easily changed by editing the value of the QLF_PRECISION macro in the qlf_write.fmi file in the directory [FME_HOME]\pipeline.

For example, if your FME installation is in C:\Program Files\FME the file qlf_write.fmi can be found in C:\Program Files\FME\pipeline directory.

DATASET, DEF

These directives are processed as described in the Reader Directives section.

PRECISION

Define the precision of output coordinates. To be precise, the value of this keyword will determine the number of significant digits after the decimal for the output coordinates.

Required/Optional

Optional

Values

1 to 15 (default)

*** Workbench Parameter**

Output Precision

NUMFIELD

Defines the number of user defined fields. This should be set during mapping file generation.

Required/Optional

Optional

Values

1 to 512

Default: 9

*** Workbench Parameter**

Number of Fields

Feature Representation

QLF features consist of geometry and attributes. The attribute names are defined in the **DEF** line and there is a value for each attribute in each QLF feature.

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Attribute Name	Contents
qlf_type	The QLF geometric type of this entity. Range: qlf_point qlf_polygon qlf_line qlf_none Default: No default
F1- Fn	Represents a feature attribute where 'n' is the number set via the keyword NUMFIELDS during mapping file generation. If NUMFIELDS is set to 5 then there will be 5 attributes F1, F2, F3, F4 and F5. Range: Maximum of 20 characters Default: Blank

Points

qlf_type: qlf_point

QLF point features specify a single x and y coordinate in addition to any associated user-defined attributes. There are no special FME attributes for the QLF line type.

Lines

qlf_type: qlf_line

QLF line features specify linear features defined by a sequence of x and y coordinates. There are no special FME attributes for the QLF lines type.

Polygon

qlf_type: qlf_polygon

QLF polygon features specify area (polygonal) features. The areas that make up a single feature may or may not be disjoint, and may contain polygons that have holes. There are no special FME attributes for the QLF region type.

CityGML Reader/Writer

Format Note: This format is not supported by FME Base Edition.

The CityGML module enables FME to read and write files in the CityGML format.

This chapter assumes familiarity with GML and the CityGML format.

Overview

CityGML is an XML-based format for the storage and exchange of 3D urban models. It extends Geography Markup Language 3 (GML3) through an application schema.

This schema specification can be found at the CityGML website <http://www.citygml.org/>.

CityGML Quick Facts

Format Type Identifier	CITYGML
Reader/Writer	Both
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	CityGML Thematic and Appearance Models
Typical File Extensions	.gml, .xml
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	No
Schema Required	No
Transaction Support	No
Geometry Type	xml_type

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	yes	point	yes
circles	no	polygon	yes
circular arc	no	raster	no
donut polygon	yes	solid	yes
elliptical arc	no	surface	yes
ellipses	no	text	no
line	yes	z values	yes
none	yes		

Reader Overview

This reader supports CityGML datasets conforming to the CityGML 1.0, 0.4.0, and 0.3.1 application schemas.

The CityGML reader reads all parts of the Thematic and Appearance CityGML models with the exception of the Digital Terrain Models (DTMs) and Addresses.

Reader Directives

The suffixes listed are prefixed by the current *<ReaderKeyword>* in a mapping file. By default, the *<ReaderKeyword>* for the CityGML reader is *CITYGML*.

DATASET

This directive specifies the location for the input CityGML instance document.

Required/Optional

Required

Mapping File Example

```
CITYGML_DATASET C:\CityGML_Data\GenericObjects.xml
```

* Workbench Parameter

Source CityGML File(s)

APPEARANCES

This directive determines whether appearance information specified in the CityGML instance document (including texture files) should be read into the FME Appearances library.

Required/Optional

Optional

Values

No | Yes

Mapping File Syntax

```
CITYGML_APPEARANCES Yes
```

* Workbench Parameter

Read Textures and materials

MATERIAL_FEATURES

Required/Optional: *Optional*

This directive specifies whether features types for X3DMaterial and ParameterizedTexture elements should be created. Valid values are *No* and *Yes*.

In general, this keyword should be left with the value of "No" as the CityGML X3DMaterial and ParameterizedTexture are automatically incorporated into the FME geometry's appearances.

Example:

```
CITYGML_MATERIAL_FEATURES No
```

Workbench Parameter: *Include X3DMaterials and ParameterizedTextures as Feature types*

SRS_AXIS_ORDER

This directive overrides the axis order when reading coordinate tuples in a CityGML <pos> or <posList> element.

Required/Optional

Optional

Values

1,2,3 | 2,1,3

Mapping File Syntax

```
CITYGML_SRS_AXIS_ORDER 2,1,3
```

*** Workbench Parameter**

GML SRS Axis Order

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

*** Workbench Parameter**

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

This writer currently supports writing of the CityGML 1.0 and 0.4 spec.

The Noise Application Domain Extension is also supported for writing (for 0.4 and 1.0).

The writer can be populated with all possible CityGML feature types by importing feature type definitions from files found in the

```
xml/CityGML/writer_feature_types/
```

subdirectory of the FME installation.

Importing feature type definitions from

```
CityGML_feature_types.xml
```

will create feature type definitions for all supported types in the CityGML 1.0 and 0.4 spec, and

`CityGML_NoiseADE_feature_types.xml`

will additionally create definitions for the Noise Application Domain Extension feature types.

Writer Directives

The suffixes listed are prefixed by the current `<WriterKeyword>` in a mapping file. By default, the `<WriterKeyword>` for the CityGML writer is `CITYGML`.

DATASET

This directive specifies the location for the output CityGML instance document.

By default, the `<WriterKeyword>` for the CityGML writer is `CITYGML`.

Required/Optional

Required

Mapping File Syntax

`CITYGML_DATASET C:\CityGML_Data\GenericObjects.gml`

* Workbench Parameter

Destination CityGML Document

DOCUMENT_ENCODING

By default, the CityGML writer produces UTF-8 encoded documents. If this parameter is set to another encoding, the writer will transcode the data to the specified encoding.

Required/Optional

Optional

Values

No | Yes

Mapping File Syntax

`CITYGML_DOCUMENT_ENCODING UTF-16BE`

* Workbench Parameter

CityGML Document Encoding

SRS_NAME

The CityGML writer will attempt to write `srsName` attributes on the geometry based on EPSG numbers that match the coordinate system of the features that it is writing.

In some cases, there are no existing well-known names. If this is the case, FME will not write an `srsName`. However, if you want to provide an `srsName` attribute, you can set the directive `SRS_NAME` and its value will be inserted into the `srsName` attribute of the geometry written.

Note that this is **not** the same as setting the coordinate system keyword/parameter for the writer. No reprojection will be done on the basis of the `SRS_NAME` directive. This directive is strictly for those cases where the user can provide a **name** (and possibly an axis order) for an `srsName` that FME is not aware of.

Required/Optional

Optional

Mapping File Syntax

`SRS_NAME EPSG:4326`

* Workbench Parameter

GML srsName

SRS_AXIS_ORDER

This parameter is used only when the user is providing an srsName via the `SRS_NAME` parameter.

It determines the coordinate order when writing geometries.

Required/Optional

Optional

Mapping File Syntax

`SRS_AXIS_ORDER 2,1,3`

* Workbench Parameter

GML srs Axis Order

ADE

This directive specifies the name of the Application Domain Extension (ADE), if any, to be used.

Required/Optional

Optional

Values

None | Noise ADE

Mapping File Syntax

`CITYGML_ADE NoiseADE`

* Workbench Parameter

Application Domain Extension

TEXTURE_DIRECTORY

This directive specifies the name of the subdirectory to which texture files will be written out, if applicable.

The subdirectory name by default is (filename)_appearance, where (filename) is the name of the dataset.

Required/Optional

Optional

Mapping File Syntax

`CITYGML_TEXTURE_DIRECTORY SampleData_Appearance`

* Workbench Parameter

Texture subdirectory

THEME_NAME

This directive specifies the name of the theme under which FME Appearances are written in the CityGML instance document.

Required/Optional

Optional

Mapping File Syntax

```
CITYGML_THEME_NAME FMETHeme
```

* Workbench Parameter

Theme name

VERSION

This directive specifies the version of CityGML to be written.

The output documents are quite different, as CityGML changed significantly between these two versions (especially regarding the namespaces in which objects are located).

Required/Optional

Optional

Values

0.4 | 1.0

Mapping File Syntax

```
CITYGML_VERSION 1.0
```

* Workbench Parameter

CityGML Version for Writing

Feature Representation

This section describes how multiple geometries are handled in the CityGML Reader and writer, how levels of detail are portrayed, and explains the feature hierarchy that is created when the CityGML Reader interprets a CityGML dataset.

CityGML features read from the CityGML Reader are named the same in FME as they are in the application schema. For example, a CityGML Building will create a feature type named *Building* in FME. The only exception to this is *GenericCityObjects*. As generic objects, the name of the feature-type will match the name of its `gml_name` attribute whenever possible.

Multiple Geometries

In a CityGML dataset, the same feature may be represented in multiple levels of detail simultaneously. Since the FME does not support multiple geometries on a feature, the CityGML reader will create a single aggregate of geometries for a feature-type, one geometry for each level of detail.

Should only one level of detail be available for a feature-type, then a non-aggregate geometry representing the feature will be created.

As multiple geometries defined in CityGML may map to a single FME geometry, we keep the original CityGML geometry in a trait called `gml_geometry`.

This flexibility of geometry poses a problem for users wishing to write CityGML. Given a feature with some geometry, how will the writer interpret the role of the geometry. For example, a Building supports over a dozen different geometry elements. How will the writer determine the appropriate **role** for the geometry? In general, the user must mark each geometry component with a geometry trait that defines the geometry role that the geometry plays. For example if the user is writing a Building and has both a multi-surface geometry and a solid geometry, the user can use a `GeometryTraitSetter` to set the attribute `citygml_lod_name` to establish the roles, marking the multisurface with, for example, `lod3MultiSurface` and the solid with `Lod2Solid`. In some cases, it may be necessary to use a `Deaggregator` transformer to split the geometries up, mark each component, and then aggregate them together again.

The value of the `citygml_lod_name` attribute must also be compatible with the geometry type. For example, a geometry tagged with `lod3MultiSurface` must be a `MultiSurface`, or the geometry will not be written out. In some cases, a geometry that is not valid for a given role will be converted to a geometry that is valid. For example, a `Surface` tagged with `lod3MultiSurface` will be wrapped in a `MultiSurface`, and a `BRepSolid` tagged with `lod2MultiCurve` will be converted to a `MultiCurve`.

In order to aid in automatic translations, geometries without a `citygml_lod_name` trait will be assumed to be `lod4Geometry` elements if their feature-type is `GenericCityObject`.

Level of Detail

In order to keep track of the particular level of detail a feature with a geometry has, the CityGML Reader will create a list attribute called `citygml_level_of_detail`. The list element values will be integers between zero and four, inclusive. If the feature-type is an aggregate of features, then the list attribute will contain as many elements as there are geometries in that aggregate. Geometries in an aggregate maintain their order, and the list attribute keeps track of the levels of detail of each geometry in sequence.

Should a non-aggregate geometry be created, the `citygml_level_of_detail` list attribute will only contain a single element referring to the level of detail the geometry created.

In addition to the `citygml_level_of_detail` list attribute, two geometry traits will be put on the geometry itself in order to identify its level of detail: `citygml_level_of_detail` and `citygml_lod_name`. An example of this follows:

```
+++++
Feature Type: CityFurniture'
Attribute(string): citygml_class' has value 1000'
Attribute(string): citygml_function' has value 1080'
Attribute(string): citygml_level_of_detail{0}' has value 2'
Attribute(string): fme_feature_type' has value CityFurniture'
Attribute(string): fme_geometry' has value fme_point'
Attribute(string): fme_type' has value fme_point'
Attribute(string): gml_id' has value gml-id01'
Attribute(string): xml_type' has value xml_point'
Attribute: citygml_level_of_detail{0}' is sequenced
Coordinate System: '
Geometry Type: IFMEPoint
Number of Geometry Traits: 1
GeometryTrait(string): citygml_level_of_detail' has value 2'
GeometryTrait(string): citygml_lod_name' has value lod2Geometry'
Coordinate Dimension: 3
(0,0,0)
=====
```

Feature Hierarchy

The design principle for CityGML is to model real-world entities as features, such as buildings and walls, and to maintain 'part-of' relationships between features. For example, a window and a door may be on the same semantic level, thus they can both be 'part-of' the same wall.

The CityGML Reader mimics this hierarchy with regards to the CityGML Thematic Model by creating a feature for each of the CityGML features, and maintains the 'part-of' relationships through `gml_id` and `gml_parent_id` attributes. In the example above, both the window and the door would specify a `gml_parent_id` equivalent to the `gml_id` of the wall that they would be a part of.

The reading of texture data may also be suppressed through the `APPEARANCES` parameter in order to speed up translations. If texture reading is suppressed, `X3DMaterial` and `ParameterizedTexture` features will always be created.

Data features with no geometries but with a reference point (for example, in the case of implicit geometries) will have a point geometry created corresponding to this reference point.

The CityGML Writer similarly uses the `gml_id` and `gml_parent_id` attributes to determine feature hierarchy.

The following is deprecated and only applies if the `MATERIAL_FEATURES` directive is set to Yes: Previously, the CityGML Reader didn't support the FME geometry appearance mode. Thus, the CityGML material XML elements, the `X3DMaterial(s)`, and `ParameterizedTexture(s)` elements were read as separate feature types. The CityGML appearance information is now automatically applied to the specified target surfaces, the creation of feature types from `X3DMaterial` and `ParameterizedTexture` elements are thus unnecessary and suppressed by default.

Attributes

In addition to the generic FME feature attributes that FME Workbench adds to all features (see [About Feature Attributes](#)), this format adds the format-specific attributes and attribute types described in this section.

CityGML Attribute Types

CityGML provides the usual assortment of attribute types, usually prefixed with either `citygml_` or `xml_`. One type that is particularly interesting to users of the CityGML Writer is the `xml_xml` type. A string attribute that is set to this type will not be encoded when written into the XML document. One place that this is very useful is when writing elements of type `xalAddress` or other places where you wish to provide a back door into the document and insert an `xml` fragment into the document directly.

XML Type

The geometry of the CityGML feature may be identified by its `xml_type` attribute. The valid values for this attribute are:

<code>xml_type</code>	Description
<code>xml_no_geom</code>	FME Feature with no geometry.
<code>xml_point</code>	Point feature.
<code>xml_line</code>	Linear feature.
<code>xml_surface</code>	Surface feature, may contain gaps.
<code>xml_solid</code>	Solid feature, may contain voids.
<code>xml_aggregate</code>	A possibly heterogeneous collection of geometries.

No Geometry

xml_type: `xml_no_geom`

Features having their `xml_type` set to `xml_no_geom` do not contain any geometry data.

Point

xml_type: `xml_point`

Features having their `xml_type` set to `xml_point` are single coordinate features or aggregates of single coordinate features.

Line

xml_type: `xml_line`

Features having their `xml_type` set to `xml_line` are polyline features and have at least two coordinates or aggregates of polyline features.

Surface

xml_type: `xml_surface`

Features having their `xml_type` set to `xml_surface` are surfaces. They may be simple, topologically contiguous surfaces or aggregates of surface features.

Collection

xml_type: `xml_aggregate`

Features having their `xml_type` set to `xml_aggregate` are complex geometries. Each component of an aggregate may be any of the types listed in this section, including `xml_aggregate` (i.e. and aggregate may contain aggregate components).

CityGML-Specific Attributes

Other attributes depend on the feature type. Common and feature-specific attributes are as follows.

Common Attributes:

Attribute Name	Contents
<code>gml_id</code>	The unique identifier for each feature. This attribute must be unique in the dataset scope.
<code>gml_parent_id</code>	The unique identifier representing the feature's parent in the hierarchy. This must reference another feature in the dataset.
<code>gml_name</code>	The given name of the feature.
<code>gml_description</code>	A description of the feature.
<code>citygml_class</code>	A four-digit identifier for the class. Range: 1000...9999
<code>citygml_function</code>	A four-digit identifier for the function. Range: 1000...9999
<code>citygml_usage</code>	A four-digit identifier for the use. Range: 1000...9999
<code>citygml_level_of_detail{}</code>	The level of detail of the geometry/geometries. Range: 0...4
<code>citygml_lod_name</code>	Provides the specific name of the level of detail element that this feature is enclosed. The Range is (in pseudo-regular expression syntax) <code>lod[0-4](-Network (Multi)?(Surface Solid))</code> . Examples: <code>lod0Network</code> , <code>lod3Solid</code> , etc. Not all combinations are legal,

Attribute Name	Contents
	and not all CityGML feature types support all options.
citygml_library_object	The target URI for the implicit geometry.
citygml_mime_type	The mime type of the library object.
citygml_transformation_matrix	A 4x4 matrix describing the translation of the implicit geometry. Elements are space-delimited.

Building and BuildingPart Attributes:

Attribute Name	Contents
citygml_year_of_construction	The construction year.
citygml_year_of_demolition	The demolition year.
citygml_roof_type	A four-digit identifier for the roof type. Range: 1000...9999
citygml_measured_height	A real number describing the measured height.
citygml_measured_height_units	The units of measure for the height.
citygml_storeys_above_ground	A positive integer describing the number of storeys above ground.
citygml_storeys_below_ground	A positive integer describing the number of storeys below ground.
citygml_storey_heights_above_ground	A list of real numbers describing the heights of the storeys above ground. Elements are space-delimited, and the first element corresponds to the storey closest to ground level.
citygml_storey_heights_above_ground_units	The units of measure for the heights.
citygml_storey_heights_below_ground	A list of real numbers describing the heights of the storeys below ground. Elements are space-delimited, and the first element corresponds to the storey closest to ground level.
citygml_storey_heights_below_ground_units	The units of measure for the heights.

Address Attributes:

Attribute Name	Contents
citygml_address	An XML fragment describing the address in the OASIS Extensible Address Language (xAL).

WaterSurface Attributes:

Attribute Name	Contents
citygml_water_level	A four-digit identifier for the type of water level. Range: 1000...9999

TrafficArea and AuxiliaryTrafficArea Attributes:

Attribute Name	Contents
citygml_surface_material	A four-digit identifier for the type of surface material. Range: 1000...9999

SolitaryVegetationObject Attributes:

Attribute Name	Contents
citygml_species	A four-digit identifier for the type of vegetation. Range: 1000...9999
citygml_height	A real number describing the height.
citygml_height_units	The units of measure for the height.
citygml_trunk_diameter	A real number describing the trunk diameter.
citygml_trunk_diameter_units	The units of measure for the diameter.
citygml_crown_diameter	A real number describing the crown diameter.
citygml_crown_diameter_units	The units of measure for the diameter.

PlantCover Attributes:

Attribute Name	Contents
citygml_species	A four-digit identifier for the type of vegetation. Range: 1000...9999
citygml_average_height	A real number describing the average height.
citygml_average_height_units	The units of measure for the average height.

Appearance Attributes:

Attribute Name	Contents
citygml_theme	The name of the themed appearance.

X3DMaterial Attributes:

Attribute Name	Contents
citygml_is_front	A boolean describing whether the material should be applied to the front or the back of the target surface. Range: true false 1 0

Attribute Name	Contents
citygml_ambient_intensity	A real number describing the ambient intensity. Range: 0...1
citygml_diffuse_color	A list of three real numbers describing diffuse color. Elements are space-delimited and must conform to the following range. Range: 0...1
citygml_emissive_color	A list of three real numbers describing emissive color. Elements are space-delimited and must conform to the following range. Range: 0...1
citygml_specular_color	A list of three real numbers describing specular color. Elements are space-delimited and must conform to the following range. Range: 0...1
citygml_shininess	A real number describing the shininess. Range: 0...1
citygml_transparency	A real number describing the transparency. Range: 0...1
citygml_is_smooth	A boolean denoting if the surface is smooth or not. Range: true false 1 0
citygml_target	Targets to apply the material to.
GeoreferencedTexture Attributes:	
Attribute Name	Contents
citygml_is_front	A boolean describing whether the material should be applied to the front or the back of the target surface. Range: true false 1 0
citygml_image_uri	The target uri that the texture is located at.
citygml_mime_type	The mime type of the texture.
citygml_texture_type	The texture type. Range: specific typical unknown
citygml_wrap_mode	The type of wrapping to apply. Range: none wrap mirror clamp border

Attribute Name	Contents
citygml_border_color	A list of three to four real numbers describing the border color and opacity. Elements are space-delimited and must conform to the following range. Range: 0...1
citygml_prefer_world_file	A boolean denoting if a world file should be sought and used when possible instead of the included geo-referenced texture data. Range: true false 1 0
citygml_orientation_matrix	A 2x2 matrix describing the rotation and scaling of the texture. Elements are space-delimited.
citygml_target	Targets to apply the texture to.

ParameterizedTexture Attributes:

Attribute Name	Contents
citygml_is_front	A boolean describing whether the material should be applied to the front or the back of the target surface. Range: true false 1 0
citygml_image_uri	The target uri that the texture is located at.
citygml_mime_type	The mime type of the texture.
citygml_texture_type	The texture type. Range: specific typical unknown
citygml_wrap_mode	The type of wrapping to apply. Range: none wrap mirror clamp border
citygml_border_color	A list of three to four real numbers describing the border color and opacity. Elements are space-delimited and must conform to the following range. Range: 0...1
citygml_target{}.uri	The location of the surface that the texture applies to.
citygml_target{}.coordinate_list{}.ring	The named ring of the surface that the texture applies to.
citygml_target{}.coordinate_list{}- .coordinates	A list of real numbers describing the coordinates on the ring that the texture applies to. Elements are space-delimited.
citygml_target{}.coordinate_list{}.world_to_texture_matrix	A 3x4 matrix describing the linear translation and spatial location of the texture to be mapped. Elements are space-delimited.

Noise ADE

The Noise Application Data Extension (ADE) extends the CityGML model by adding new feature-types and attributes that allow noise data to be transported with a CityGML city model. To do this, the Noise ADE adds the NoiseRoadSegment, NoiseRailwaySegment, Train, and NoiseCityFurnitureSegment feature-types, as well as adding noise-specific attributes to the Building feature-type.

To specify that the CityGML Reader should read or write Noise ADE data, please ensure that the ADE directive is set to *NoiseADE*.

Feature Hierarchy

The Noise ADE adds a NoiseRoadSegment as part of a Road, a NoiseRailwaySegment as part of a Railway, as well as a NoiseCityFurnitureSegment, which is part of a CityFurniture. As with standard CityGML, these will have `gml_parent_id` attributes which contain the `gml_id` attributes of their respective parents. Similarly, a new Train feature-type is considered part of a NoiseRailwaySegment, and will also have a `gml_parent_id` which reflects this.

Attributes

In addition to the format-specific attributes shown above for standard CityGML, the following attributes are also added for the Noise ADE. Noise ADE attributes are prefixed with `ade_noise` to clearly distinguish them. In the case that a Noise ADE attribute is a measure type, there will be an associated FME attribute with the same name, but suffixed with `'_units'` to provide the Units Of Measure.

Building Attributes:

Attribute Name	Contents
<code>ade_noise_building_reflection</code>	The reflection property of the building.
<code>ade_noise_building_reflection_correction</code>	A real number describing noise emission in dB.
<code>ade_noise_building_Lmax_day</code>	A real number describing noise emission L_{\max} for the day in dB.
<code>ade_noise_building_Lmin_day</code>	A real number describing noise emission L_{\min} for the day in dB.
<code>ade_noise_building_Lmax_night</code>	A real number describing noise emission L_{\max} for the night in dB.
<code>ade_noise_building_Lmin_night</code>	A real number describing noise emission L_{\min} for the night in dB.
<code>ade_noise_building_Leq_day</code>	A real number describing noise emission L_{eq} for the day in dB.
<code>ade_noise_building_Leq_night</code>	A real number describing noise emission L_{eq} for the night in dB.
<code>ade_noise_building_inhabitants</code>	A positive integer describing the number of building inhabitants.
<code>ade_noise_building_apartments</code>	A positive integer describing the number of building apartments.

Attribute Name	Contents
ade_noise_buliding_emission_points	A list of integers for emission points. Elements are space-delimited.
ade_noise_building_remark	Any additional remarks.

NoiseRoadSegment Attributes:

Attribute Name	Contents
ade_noise_average_hourly_traffic_day	A real number describing the hourly traffic flow for the day in vehicles per hour.
ade_noise_average_hourly_traffic_evening	A real number describing the hourly traffic flow for the evening in vehicles per hour.
ade_noise_average_hourly_traffic_night	A real number describing the hourly traffic flow for the night in vehicles per hour.
ade_noise_average_hourly_traffic_day_16	A real number describing the hourly traffic flow for the 16-hour day in vehicles per hour.
ade_noise_heavy_vehicle_percentage_day	A real number describing the percentage of heavy vehicles for the day.
ade_noise_heavy_vehicle_percentage_evening	A real number describing the percentage of heavy vehicles for the evening.
ade_noise_heavy_vehicle_percentage_night	A real number describing the percentage of heavy vehicles for the night.
ade_noise_heavy_vehicle_percentage_day_16	A real number describing the percentage of heavy vehicles for the 16-hour day.
ade_noise_average_daily_traffic	A real number describing the average daily traffic flow in vehicles per 24 hours.
ade_noise_passenger_car_speed_limit_day	A real number describing the speed limit for passenger cars for the day in kilometers per hour.
ade_noise_passenger_car_speed_limit_evening	A real number describing the speed limit for passenger cars for the evening in kilometers per hour.
ade_noise_passenger_car_speed_limit_night	A real number describing the speed limit for passenger cars for the night in kilometers per hour.
ade_noise_heavy_vehicle_speed_limit_day	A real number describing the speed limit for heavy vehicles for the day in kilometers per hour.
ade_noise_heavy_vehicle_speed_limit_evening	A real number describing the speed limit for heavy vehicles for the evening in kilometers per hour.

Attribute Name	Contents
ade_noise_heavy_vehicle_speed_limit_night	A real number describing the speed limit for heavy vehicles for the night in kilometers per hour.
ade_noise_road_surface_material	The surface material of the road.
ade_noise_road_surface_correction	A real number describing noise emission in dB.
ade_noise_carriageway_width	A real number describing the width of the carriageway cross-section in meters.
ade_noise_outer_lane_to_center_width	A real number describing the distance from the outer road lane to the road center line in meters.
ade_noise_bridge	A boolean denoting if this is a bridge or not. Range: true false 1 0
ade_noise_tunnel	A boolean denoting if this is a tunnel or not. Range: true false 1 0
ade_noise_road_gradient	A real number describing the road gradient as a percentage.
ade_noise_lineage	A remark regarding the data source.

NoiseRailwaySegment Attributes:

Attribute Name	Contents
ade_noise_railway_surface_material	The surface material of the railway.
ade_noise_railway_surface_correction	A real number describing noise emission in dB.
ade_noise_bridge	A boolean denoting if this is a bridge or not. Range: true false 1 0
ade_noise_crossing	A boolean denoting if this is a crossing or not. Range: true false 1 0
ade_noise_curve_radius	A real number describing the curve radius in meters.
ade_noise_additional_correction_segment	A real number describing an addition noise emission in dB.

Train Attributes:

Attribute Name	Contents
ade_noise_train_type	The type of train.
ade_noise_train_type_correction	A real number describing noise emission in dB.
ade_noise_brake_portion_day	A real number describing the percentage of wagons with wheel disc brakes during the day.

Attribute Name	Contents
ade_noise_brake_portion_evening	A real number describing the percentage of wagons with wheel disc brakes during the evening.
ade_noise_brake_portion_night	A real number describing the percentage of wagons with wheel disc brakes during the night.
ade_noise_train_length_day	A real number describing the average train length during the day in meters.
ade_noise_train_length_evening	A real number describing the average train length during the evening in meters.
ade_noise_train_length_night	A real number describing the average train length during the night in meters.
ade_noise_speed_limit_day	A real number describing the speed limit during the day in kilometers per hour.
ade_noise_speed_limit_evening	A real number describing the speed limit during the evening in kilometers per hour.
ade_noise_speed_limit_night	A real number describing the speed limit during the night in kilometers per hour.
ade_noise_additional_correction_train	A real number describing an additional noise emission in dB.

NoiseCityFurnitureSegment Attributes:

Attribute Name	Contents
ade_noise_type	The type of noise barrier.
ade_noise_reflection	The reflection property of the noise barrier.
ade_noise_reflection_correction	A real number describing noise emission in dB.
ade_noise_height	A real number describing the height of the noise barrier in meters.
ade_noise_distance_to_road_center	A real number describing the distance between the noise barrier and the road center line in meters.

Writing CityGML from FME

The FME CityGML writer maps any feature type definitions that are not defined in CityGML into GenericCityObject(s). This document explains how the FME CityGML writer can be used to output datasets with predefined thematic CityGML feature types, such as Building, CityFurniture, WaterBody, Road, Railway, etc...

CityGML feature types in FME

The simplest way to get CityGML writer feature type definitions for the supported CityGML features is to import them from an existing CityGML dataset.

FME ships with two sample CityGML datasets that can be used to populate Workbench with the required CityGML writer feature types. These files are located under the "xml/CityGML/writer_feature_types" directory of the FME installation. The "CityGML_feature_types.xml" and "CityGML_NoiseADE_feature_types.xml" sample datasets have the feature type definitions for CityGML (0.4 and 1.0) and CityGML-NoiseADE (0.4 and 1.0), respectively. Note that these CityGML sample files are not complete datasets in the GML sense; they do not contain any meaningful data for reading, and are only meant to be used for importing feature type definitions into a writer.

CityGML levels of detail

CityGML specifies five different Levels of Detail (LODs), ranging from 0 (general topology) to 4 (detailed architectural features and furniture). Most features may contain geometry models for different LODs. LOD0 generally models terrain features. LOD1 models simple prismatic buildings and general landscape features. LOD2 incorporates some architectural features in building models, greater detail in transportation, vegetation, and outdoor furniture features. LOD3 models buildings and outdoor objects as they would actually appear. And LOD4 models detailed interior structures.

In order for various applications to correctly interpret the multiple geometries of a feature, GML geometries are enclosed in an element that indicates the geometry role. As an example, a Building feature with a solid geometry at LOD2 is shown below:

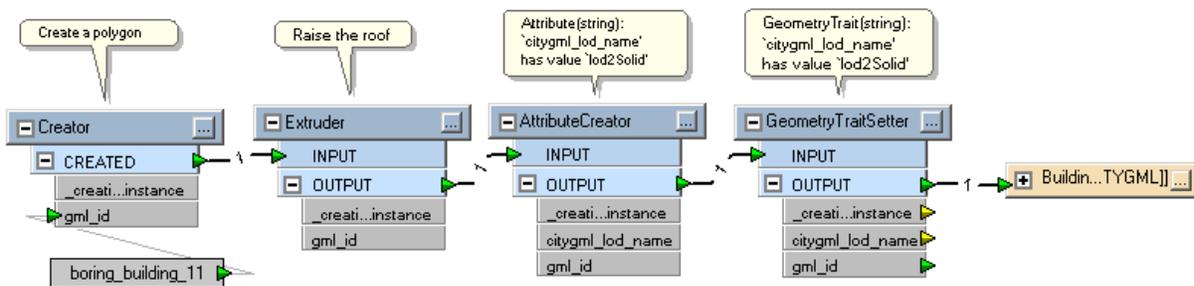
```
<cityObjectMember>
<bldg:Building gml:id="building890384">
<bldg:lod2Solid>
<gml:Solid gml:id="aebd7312">
...

```

FME Geometries in CityGML features types must be tagged with their intended geometry role in order to be written out correctly. The geometry role is specified through the geometry trait "citygml_lod_name". The geometry roles that are valid for each feature type are given in Table 1, and the valid FME geometry types for each geometry role are shown in Table 2.

Setting the geometry role in Workbench

The geometry role can be set using the AttributeCreator and the GeometryTraitSetter transformers in tandem.



The reason we are using an AttributeCreator here is to set citygml_lod_name as an attribute, and then transforming it into a trait using the GeometryTraitSetter.

Since we've passed in a valid FME geometry for the geometry role (an extrusion as a "lod2Solid"), we end up with a Building feature containing this geometry in our output:

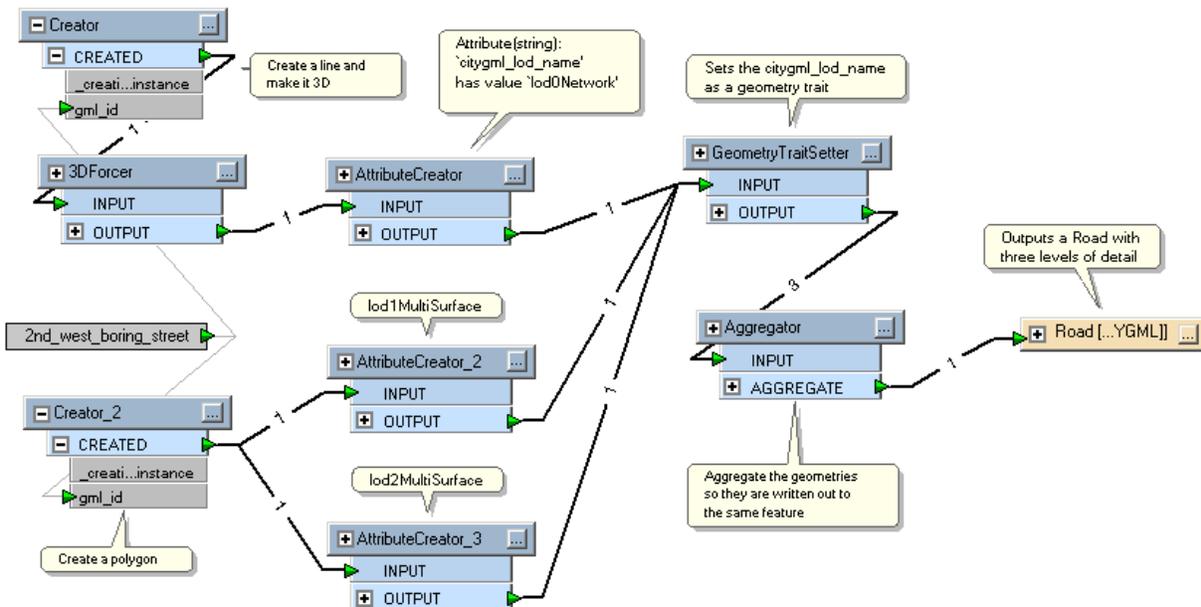
```
<cityObjectMember>
<bldg:Building gml:id="boring_building_11">
<bldg:lod2Solid>
<gml:Solid srsDimension="3">
...

```

All of the other CityGML-specific attributes can also be easily set on an FME feature through the AttributeCreator, though many of these attributes have a restricted set of values that are defined in the CityGML external code lists. See Tables 1-4 for more details.

Writing multiple geometry roles to one feature

Creating additional geometry roles for a feature is not much more difficult. Simply label a copy of the same, or any other geometry with the appropriate role, and aggregate all of the geometries for the feature into a single geometry:



Here is the CityGML Road feature:

```
<tran:Road gml:id="2nd_west_boring_street">
  <tran:lod0Network>
    <gml:LineString srsDimension="3">
      <gml:posList>12 0 0 12 12 0 0 12 0</gml:posList>
    </gml:LineString>
  </tran:lod0Network>
  <tran:lod1MultiSurface>
    <gml:MultiSurface srsDimension="3">
      <gml:surfaceMember>
        <gml:Polygon>
          <gml:exterior>
            <gml:LinearRing>
              <gml:posList>13 0 0 11 0 0 11 11 0 0 11 0 0 13 0 13 13 0 13 0 0
              </gml:posList>
            </gml:LinearRing>
          </gml:exterior>
        </gml:Polygon>
      </gml:surfaceMember>
    </gml:MultiSurface>
  </tran:Road>
```

```
</tran:lod1MultiSurface>
<tran:lod2MultiSurface>
  <!--The same gml:MultiSurface as for the lod1MultiSurface-->
</tran:lod2MultiSurface>
</tran:Road>
```

Generally, the geometry at LOD2 should have more detail than the geometry at LOD1, but using the same geometry works for this example. In the case of this Road, it may have extra curves representing the separate lanes of the road.

Geometry validation in Workbench

Note that in the above example, we tagged a polygon as a lodXMultiSurface, though it only represents a single area. The CityGML writer will check that the FME geometry can actually be written out as the specified geometry role. If it is unable to, and it is easy to convert the geometry to a type that is valid, the writer will make the conversion. For example, our polygon was converted to a surface and then wrapped in a multi-surface. If the FME geometry is not valid for the specified geometry role and cannot be converted to a valid geometry (Eg. a point tagged with "lod3MultiCurve"), it will not be written out.

Setting CityGML attributes and properties in FME

The non-geometric attributes and properties of the CityGML features are specified through certain attributes of the FME feature types. Most of these attributes are prefixed with "citygml_". For example, the value of the "citygml_roof_type" attribute of an FME Building feature will be written as the value of the "roofType" element of a CityGML Building feature.

The full list of valid attributes for a CityGML feature type will appear on feature types defined by importing feature types from a CityGML document that contains those features.

Most of the CityGML attributes and properties that take a string value have an enumerated list of valid values that are listed in the CityGML external code lists. For example, for the "roofType" element of a Building feature, "1070" indicates a pavilion roof. The corresponding attributes of CityGML features in FME should be set to one of these valid values, though FME currently does no validation of the values.

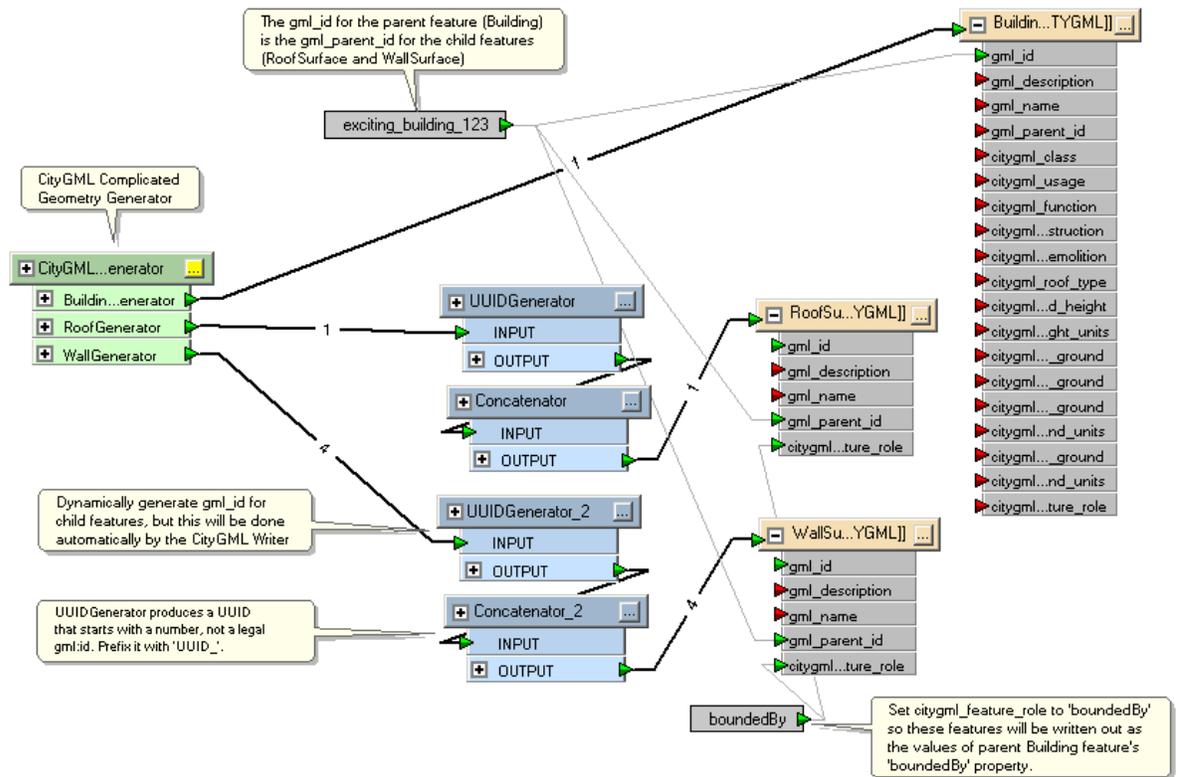
Writing CityGML Appearances from FME

FME 2010 supports reading and writing of CityGML appearances. In FME, Appearance objects are stored in a common library, and surfaces contain a reference to the Appearance that is applied to them. The FME Appearance contains information about both constant (material) and non-constant (texture) surface properties. Any FME Appearances that are referenced by a surface passed into the CityGML Writer will be written out as a ParameterizedTexture or an X3DMaterial, depending on whether or not the Appearance contains textural information. The writer will try to re-use any existing elements in the case that ParameterizedTexture or X3DMaterial features are passed in (Eg. during a CityGML to CityGML translation). In this case, the elements will be written out as members of the original CityGML Appearance and under their original theme. If new elements must be created, they will be members of a new Appearance under the CityModel. The default theme name is "FMTheme", but it can be changed through the advanced workbench parameter "Theme name". ParameterizedTexture and X3DMaterial elements can belong to the same theme even under different Appearance features.

If any of the referenced FME Appearances contain texture information, the image files will be written out to a sub-directory of the destination directory. By CityGML convention, this directory is named "appearance" by default. The directory name may be specified through the advanced workbench parameter "Texture subdirectory".

CityGML Feature Hierarchy

Many CityGML features are aggregations of other features. In FME, this relationship can be specified through the "gml_id" and "gml_parent_id" attributes. The value of the "gml_parent_id" of a child element should be equal to the value of the "gml_id" of its parent element. The gml:id attribute must be unique for all features in a CityGML document. Using the UUIDGenerator transformer will ensure that these values are unique, but they must be prefixed with a non-numeric character. Note that the CityGML Writer will dynamically generate gml:id attribute values for all features it writes (essentially using the same method shown below), so it is not necessary to use these transformers unless the value of the gml_id attribute must be accessed.



This workspace produces the following output:

```

<bldg:Building gml:id="exciting_building_123">
  <bldg:boundedBy>
    <bldg:RoofSurface gml:id="UUID_47be0fd6-cc6c-45ab-96f7-0868ff9208d0">
      <bldg:lod3MultiSurface>
        ...
      </bldg:lod3MultiSurface>
    </bldg:RoofSurface>
  </bldg:boundedBy>
  <bldg:boundedBy>
    <bldg:WallSurface gml:id="UUID_dafb5306-41aa-4e07-8f58-80874167a2a7">
      <bldg:lod3MultiSurface>
        ...
      </bldg:lod3MultiSurface>
    </bldg:WallSurface>
  </bldg:boundedBy>
  <!--Three more wallSurfaces-->
</bldg:Building>

```

Valid Geometry Role Lists

Note: Only one of the listed values for "citygml_lod_name Value" is needed.

Table 1: Valid LOD types for CityGML Feature Types

CityGML Feature Type	citygml_lod_name Value
GenericCityObject	lod[0-4]Geometry lod[0-4]T-terrainIntersection
Address	multiPoint
Building BuildingPart	lod[1-4]Solid lod[1-4]MultiSurface lod[2-4]MultiCurve lod[1-4]T-terrainIntersection
BuildingInstallation	lod[2-4]Geometry
RoofSurface WallSurface GroundSurface ClosureSurface FloorSurface InteriorWallSurface CeilingSurface	lod[2-4]MultiSurface
Door Window	lod[3-4]MultiSurface
Room	lod4Solid lod4MultiSurface
BuildingFurniture IntBuildingInstallation	lod4Geometry
WaterBody	lod[0-1]MultiCurve lod[0-1]MultiSurface lod[1-4]Solid
WaterSurface WaterGroundSurface	lod[2-4]Surface
TransportationComplex Track Road Railway Square	lod0Network lod[1-4]MultiSurface
TrafficArea	lod[2-4]MultiSurface

AuxiliaryTrafficArea	
SolitaryVegetationObject	lod[1-4]Geometry
PlantCover	lod[1-4]MultiSurface lod[1-3]MultiSolid
CityFurniture	lod[1-4]Geometry lod[1-4]T- errainIntersection
LandUse	lod[0-4]MultiSurface
GeoreferencedTexture	referencePoint
TINRelief	extent
MassPointRelief	reliefPoints
BreaklineRelief	ridgeOrValleyLines breaklines
NoiseRoadSegment NoiseRailwaySegment NoiseCityFurnitureSegment	lod0BaseLine

Table 2: Valid geometries for the citygml_lod_name attribute values

LOD name	Valid FME Geometries	GML Geometry
lod0Geometry lod1Geometry lod2Geometry lod3Geometry lod4Geometry relativeGMLGeometry geometry	all except null, raster, and empty aggregates	gml:GeometryType
lod1MultiSolid lod2MultiSolid lod3MultiSolid	all solids all surfaces homogeneous aggregates of surfaces or areas	gml:MultiSolidType
lod0MultiSurface lod1MultiSurface lod2MultiSurface lod3MultiSurface lod4MultiSurface	all surfaces all areas homogeneous aggregates of surfaces or areas	gml:MultiSurfaceType
lod0MultiCurve lod1MultiCurve lod2MultiCurve lod3MultiCurve lod4MultiCurve	line arc all surfaces all solids homogeneous aggregates of lines, arcs, surfaces, or solids	gml:MultiCurveType
lod1Solid lod2Solid lod3Solid lod4Solid	all solids all surfaces aggregates containing a single solid or surface	gml:SolidType
lod2Surface lod3Surface lod4Surface	all surfaces all areas aggregates containing a single surface or area	gml:SurfaceType

lod0TerrainIntersection	line	gml:MultiCurveType
lod1TerrainIntersection	arc	
lod2TerrainIntersection	all surfaces	
lod3TerrainIntersection	all solids	
lod4TerrainIntersection	homogeneous aggregates of lines, arcs, surfaces, or solids	
lod0Network	all points line arc all areas all surfaces all solids aggregates containing a single point, line, arc, area, surface, or solid	gml:GeometricComplexType
noise:lod0BaseLine(move me)	line arc aggregates containing a single line or arc	
referencePoint	point aggregates containing a single point	gml:PointType
reliefPoints	point homogenous aggregates of points	
extent	face rectangular face all areas aggregates containing a single face, rectangular face, or area	gml:PolygonType
ridgeOrValleyLines	line arc all surfaces all solids homogeneous aggregates of lines, arcs, surfaces, or solids	

breaklines	line arc all surfaces all solids homogeneous aggregates of lines, arcs, surfaces, or solids	<code>gml:MultiCurveType</code>
multiPoint	point aggregate containing a single point	<code>gml:MultiPointType</code>

Table 3: CityGML Feature Roles

CityGML Feature Type	citygml_feature_role Value
Appearance	appearance * appearanceMember *
GeoreferencedTexture ParameterizedTexture X3DMaterial	surfaceDataMember
TINRelief MassPointRelief BreaklineRelief	reliefComponent
BuildingInstallation	outerBuildingInstallation
IntBuildingInstallation	interiorBuildingInstallation roomInstallation
RoofSurface WallSurface GroundSurface ClosureSurface CeilingSurface InteriorWallSurface FloorSurface	Building/boundedBy **
Room	interiorRoom
BuildingPart	consistsOfBuildingPart
Address	address
Door Window	opening
BuildingFurniture	interiorFurniture
WaterSurface WaterGroundSurface	WaterBody/boundedBy **
TrafficArea	trafficArea
AuxiliaryTrafficArea	auxiliaryTrafficArea
noise:NoiseRailwaySegment	noise:noiseRailwaySegmentProperty
noise:NoiseRoadSegment	noise:noiseRoadSegmentProperty
noise:NoiseCityFurnitureSegment	noise:noiseCityFurnitureSegmentProperty
noise:Train	noise:usedBy

Table 4: CityGML Feature Properties

CityGML Parent Feature Type	Valid properties
all feature types except Address X3DMaterial ParameterizedTexture GeoreferencedTexture noise:Train	appearance * appearanceMember *
Appearance	surfaceDataMember
ReliefFeature	reliefComponent
Building BuildingPart	outerBuildingInstallation interiorBuildingInstallation Building/boundedBy ** interiorRoom consistsOfBuildingPart address
RoofSurface WallSurface GroundSurface ClosureSurface FloorSurface InteriorWallSurface CeilingSurface	opening
Door	address
Room	Building/boundedBy ** interiorFurniture roomInstallation
WaterBody	WaterBody/boundedBy **
TransportationComplex Track Square	trafficArea auxiliaryTrafficArea
Road	trafficArea auxiliaryTrafficArea noise:noiseRoadSegmentProperty
Railway	trafficArea auxiliaryTrafficArea

	noise:noiseRailwaySegmentProperty
CityFurniture	noise:noiseCityFurnitureSegmentProperty
noise:NoiseRailwaySegment	noise:usedBy

- * Appearance features that are properties of other features in CityGML v0.4 are enclosed in an "appearanceMember" element. In v1.0, Appearances that are properties of features are enclosed in an "app:appearance" element, but those that are properties of the CityModel are enclosed in an "app:appearanceMember" element.
- ** "boundedBy" is a property of both the AbstractBuildingType and of the WaterBodyType, though the allowed values differ by module. The recognized "citygml_feature_role" value for both of these properties is "boundedBy".

ComGraphix Data Exchange Format (CGDEF) Reader/Writer

The ComGraphix Data Exchange Format (CGDEF) Reader/Writer allows FME to read and write MapGrafix import and export files. The CGDEF is a published ASCII format that can be used by the MapGrafix™ product for input and output.

ComGraphix Data Exchange Format Files are often called CGDEF files.

Overview

MapGrafix is a two-dimensional (2D) system with no provision for storing user-defined attributes with the geometric data. The CGDEF reader and writer support symbols (point), lines (vector), polylines, arcs (arc, ovalarc, and polyarc), ellipses (oval, circle), polygons, and text geometric data.

Some geometric entities may have display properties such as pen and brush width (lineweight), pattern, and color.

CGDEF files are ASCII format, and use a system of keywords and values to define map parameters, overlay (layer) structure, element definitions and graphic attributes. All CGDEF map and vector geometric data is contained in a single file with the **.cgdef** extension.

FME does **not** support the import and export of TIFF files with the CGDEF. TIFF files operate on the image layer, which is something that FME does not support.

CGDEF Quick Facts

Format Type Identifier	CGDEF
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	File
Feature Type	Overlay base name
Typical File Extensions	.cgdef
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	Yes
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	cgdef_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	yes		polygon	yes
circular arc	yes		raster	no
donut polygon	yes		solid	no
elliptical arc	yes		surface	no
ellipses	yes		text	yes
line	yes		z values	yes
none	no			

Reader Overview

The CGDEF reader first opens CGDEF file defined in the mapping file. The CGDEF reader then extracts map parameters, followed by overlay definitions, and all the features from the file. Options are provided for returning symbols as single points, or exploded into their component pieces.

Reader Directives

The suffixes shown are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the CGDEF Reader is **CGDEF**.

DATASET

Required/Optional: *Required*

The value for this directive is the file name of the CGDEF file to be read.

Example:

```
CGDEF_DATASET /usr/data/cgdef/myfile.cgdef
```

Workbench Parameter: *Source ComGraphix CGDEF File(s)*

EXPLODE_SYMBOLS

Required/Optional: *Optional*

Default Value: *Yes*

The value for this directive will determine the reader's action when it comes across a symbol in the file. If the value is YES, then the reader, rather than outputting a symbol feature, will look at the symbol definition for that symbol and output it as a series of individual features that make up the symbol. If the value is NO, then a symbol in the source file is sent as a **cgdef_symbol** type rather than attempting to send its actual features individually. The default value is YES, since most formats cannot properly interpret a symbol type as anything more than a point feature – thus, by exploding the symbol, it can be seen.

Example:

```
EXPLODE_SYMBOLS yes
```

Workbench Parameter: *Explode Symbols*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

✳ Workbench Parameter

Additional Attributes to Expose

Writer Overview

The CGDEF writer creates and writes feature data to the CGDEF file specified by the **DATASET** keyword. The directory where the file is created must exist before the translation occurs, and if there is an old CGDEF file with the same filename in the directory, it will be overwritten with the new feature data. Before actually writing out features, the writer first scans the prototype/template file defined in the mapping file (see *Writer Keywords*) to extract all the header information required by CGDEF.

Writer Directives

The suffixes shown are prefixed by the current **<WriterKeyword>** in a mapping file. By default, the **<WriterKeyword>** for the CGDEF writer is **CGDEF**.

DATASET

Required/Optional: *Required*

The value for this directive is the file name into which data is to be written.

Example:

```
CGDEF_DATASET /usr/data/cgdef/myfile.cgdef
```

Workbench Parameter: *Destination ComGraphix Data Exchange Format (CGDEF) File*

PROTOTYPE_FILE

Required/Optional: *Optional*

The value following this directive is the file name of the CGDEF file that is used as a template file. This file should include all setup and header information along with RGB color definitions (required), symbol definitions (required) and overlay definitions (optional). The writer will collect this information and use it as it processes features to output.

Example:

```
CGDEF_PROTOTYPE_FILE /usr/data/cgdef/template.cgdef
```

Workbench Parameter: *Prototype File*

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

CGDEF features consist of geometry but no user-defined attributes, although there are special attributes to hold the type of the geometric entity and its display parameters.

The FME considers the CGDEF overlay name to be the FME feature type of a CGDEF feature. When writing, the CGDEF writer will create a new overlay for each unique feature type that is passed to the writer. All CGDEF features contain a `cgdef_type` attribute, which identifies the geometric type. Each geometric/element type can also have an id, up to 31 characters long, associated with it. Every element type except symbols will have associated colors attached to it. Both `cgdef_symbol_name` and `cgdef_symbol_sequence_number` are fields that are only filled if the element is part of a symbol instance. Depending on the geometric type, the feature contains additional attributes specific to the geometric type. These are described in subsequent sections.

Attribute Name	Contents
<code>cgdef_type</code>	<p>The CGDEF geometric type of this entity.</p> <p>Range: <code>cgdef_symbol cgdef_polyline cgdef_polygon cgdef_text cgdef_ellipse cgdef_arc</code></p> <p>Default: No default</p>
<code>cgdef_element_id</code>	<p>The CGDEF ID for this entity, this is an optional attribute</p> <p>Range: String</p> <p>Default: No default</p>
<code>cgdef_color.red</code>	<p>The element's red color intensity, as determined by looking up the element's color index in the color table.</p> <p>Range: 0..65535</p> <p>Default: 27000 (when writing only)</p>
<code>cgdef_color.green</code>	<p>The element's green color intensity, as determined by looking up the element's color index in the color table.</p> <p>Range: 0..65535</p> <p>Default: 30000 (when writing only)</p>
<code>cgdef_color.blue</code>	<p>The element's blue color intensity, as determined by looking up the element's color index in the color table.</p> <p>Range: 0..65535</p> <p>Default: 38000 (when writing only)</p>
<code>fme_color</code>	<p>This is a string that represents the color intensities of the element. It is formatted as red, green, blue intensities which range between 0..1 This 0..1 value is arrived at by taking the color intensity and dividing it by the total intensity range, in this case, 65535</p> <p>Range: String. (0..1, 0..1, 0..1)</p> <p>Default: 27000/65535, 30000/65535, 38000/65535(when writing only)</p>
<code>cgdef_symbol_name</code>	<p>If the element is part of a symbol and the symbol has</p>

Attribute Name	Contents
	<p>been exploded into its individual elements, then this field contains the symbol name</p> <p>Range: String Default: None</p>
cgdef_symbol_sequence_number	<p>If the element is part of a symbol and the symbol has been exploded into its individual elements, then this field contains the a unique number which identifies itself and the other elements in the symbol</p> <p>Range: String Default: None</p>

Symbols

cgdef_type: cgdef_symbol

CGDEF symbol features specify a single x and y coordinate. This coordinate defines the center of the symbol. The symbol is defined by a symbol number, and a scale attribute. If no scale is defined, then the symbol will be placed at the current default symbol scale setting.

The table below lists the special FME attribute names used to control the CGDEF symbol settings.

Attribute Name	Contents
cgdef_symbol_number	<p>This number references a resource in the map. If the symbol number does not have a resource in the map, the default symbol rectangle is placed at the specified location.</p> <p>Range: Any integer number > 0 Default: No default</p>
cgdef_symbol_scale	<p>The scale at which the symbol is to be placed.</p> <p>Range: 1..20 Default: 1</p>

Symbol and Group Definitions

Symbol Definitions

Symbols are defined as a set of feature/element types. The collection of feature types becomes the symbol. As an example, a symbol can be defined as two circles and an arc, which together form a happy face. Thus, a **cgdef_symbol** type actually references its definition through the **cgdef_symbol_number** and places that symbol with appropriate scaling at the coordinates specified in the **cgdef_symbol** type.

Symbol Definitions

Symbols are defined as a set of feature/element types. The collection of feature types becomes the symbol. As an example, a symbol can be defined as two circles and an arc, which together form a happy face. Thus, a **cgdef_symbol** type actually references its definition through the **cgdef_symbol_number** and places that symbol with appropriate scaling at the coordinates specified in the **cgdef_symbol** type.

Group Definitions

Along the same lines is a group definition. A group is another feature which is made up of a set of other element types. However, a group does not have a group number or group name to identify it (although it may still have an ID

which any feature may possess).

FME **does not** recognize groups; instead, it outputs the elements of the group as independent features.

Text

cgdef_type: cgdef_text

CGDEF text is used for text annotation in CGDEF. The coordinates specify the lower left coordinates of the text when it is placed. In addition, the size and angle that the text is output can be specified.

The table below lists the special FME attribute names used to control the CGDEF text:

Attribute Name	Contents
cgdef_text_size	The size of the text specified in ground units of the map. Range: float > 0 Default: 0
cgdef_text_angle	The text angle is given in degrees and measured from the horizontal. Range: -360..360 Default: 0
cgdef_text_font	The type of font. Range: String Default: No default
cgdef_text_style	The display style for the text. Range: String Default: No default
cgdef_text_string	The text to be displayed Range: String Default: No default

Polylines

cgdef_type: cgdef_polyline

CGDEF polyline features specify linear features defined by a sequence of x and y coordinates. Polylines encapsulate the concept of a line since a line is just a sequence of two points. Furthermore, the polyline type will be used with the **cgdef_arc** type to handle poly arcs used in MapGraphix. Poly arcs will be represented by a sequence of polylines and arcs.

Each polyline has a pen style associated with it specifying the color, line weight, and line type used when the line is drawn. If no pen style is defined for a polyline entity, the previous style is used.

The table below lists the special FME attribute names used to control the CGDEF polyline settings.

Attribute Name	Contents
cgdef_pen_lineweight	Defines the lineweight used to draw the polyline. This is measured in screen pixels. Range: 1..127 Default: 1

Attribute Name	Contents
cgdef_pen_linetype	The linetype used to draw the line. Range: 1..19 Default: 1

Polygons

cgdef_type: cgdef_polygon

CGDEF polygon features specify area (polygonal) features. The areas that make up a single feature may or may not be disjoint, and may contain polygons that have holes. Each polygon has a pen style associated with it to control the color, line weight, line type, and brush pattern used when it's drawn. If no pen style is defined for a polygon entity, the previous style is used.

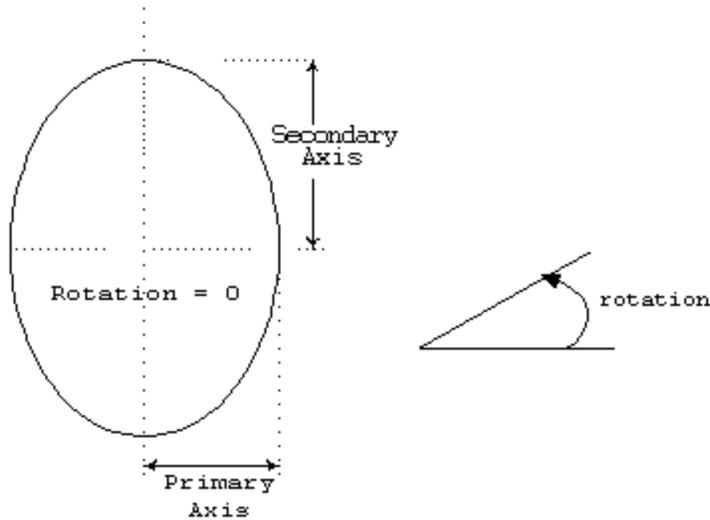
The following table lists the special FME attribute names used to control the CGDEF polygon settings.

Attribute Name	Contents
cgdef_pen_lineweight	Defines the lineweight used to draw the polyline. This is measured in screen pixels Range: 1...127 Default: 1
cgdef_pen_linetype	The linetype used to draw the line. Range: 1...19 Default: 1
cgdef_brush_pattern	The pattern used to draw the line. Range: 1...41 Default: 1

Ellipse

cgdef_type: cgdef_ellipse

The **cgdef_ellipse** corresponds to ovals in MapGraphix. Ellipse features are point features, and have only a single coordinate. This point serves as the center of the ellipse. Additional attributes specify the primary axis (X) and secondary axis (Y) of the ellipse. CGDEF ellipses also support rotation.



*Tip: The primary ellipse axis is **not** necessarily the longest axis, but rather the one on the x axis.*

CGDEF ellipses can also arrive at circles, since circles are just ellipses with equal primary axis and the secondary axis.

In addition to the attributes below, ellipses also make use of the brush and pen attributes as defined by [cgdef_polygon](#).

Attribute Name	Contents
cgdef_primary_axis	The length of the semi-major axis in ground units. (x-axis) Range: Any real number > 0 Default: No default
cgdef_secondary_axis	The length of the semi-minor axis in ground units. (y-axis) Range: Any real number > 0 Default: No default
cgdef_rotation	The rotation of the major axis. The rotation is measured in degrees counterclockwise up from horizontal. Range: -360.0..360.0 Default: 0

Arc

cgdef_type: cgdef_arc

The arc definition here handles arcs, oval arcs and parts of poly arcs used in MapGraphix. Poly arcs use the [cgdef_arc](#) type as well as [cgdef_polyline](#) types to form the original poly arc defined in the CGDEF file.

CGDEF arc features are linear features used to specify elliptical arcs. As such, the feature definition for [cgdef_arc](#) is similar to the ellipse definition, with two additional angles to control the portion of the ellipse boundary drawn. CGDEF arcs also support rotation.

Tip: The function @Arc() can be used to convert an arc to a linestring. This is useful for storing Arcs in systems that don't support them directly.

In addition to the attributes below, arcs also make use of the pen attributes as defined on [cgdef_polyline](#).

Attribute Name	Contents
cgdef_primary_axis	The length of the semi-major axis in ground units. (x-axis) Range: Any real number > 0 Default: No default
cgdef_secondary_axis	The length of the semi-minor axis in ground units. (y-axis) Range: Any real number > 0 Default: No default
cgdef_start_angle	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of start_angle. Range: 0.0..360.0 Default: 0
cgdef_sweep_angle	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of sweep_angle. Range: 0.0..360.0 Default: No default
cgdef_rotation	The rotation of the major axis. The rotation is measured in degrees counterclockwise up from horizontal. Range: -360.0..360.0 Default: 0

Comma-Separated Value (CSV) Reader/Writer

The Comma-Separated Value (CSV) Reader/Writer allows FME to read and write files in the CSV format. This reader/writer provides a somewhat simpler interface to the FME's Relational Table Reader/Writer CSV module.

Overview

CSV files are ASCII database files, where each column in a row is separated by some separator character. The FME feature attributes for each line of the file are the columns values of that row. There is no geometry or dimension to the features created from the CSV files, but they may have attributes that can be turned into geometry via FME facilities such as @XValue, @YValue, and ConnectionFactory. Therefore, none of the features read from CSV are directly viewable.

By convention, these files use the .csv filename extension, but the CSV reader and writer can use any extension. CSV reader can also read from a gzipped file with that extension ".csv.gz" and the writer can write a gzipped file if the extension of destination file ends with ".gz"

Tip

When using RELATE statements that use a CSV file (where the CSV file is not part of the source or destination dataset), the Relational Table reader/writer keywords for CSV should be used instead of the CSV reader/writer keywords (see the chapter on Relational Table Reader/Writer). For example:

```
Relate TABLE_DEF roads CSV \  
  CSV_OUTPUT_FIELDNAMES yes \  
  MSLINK number(10,0) \  
  LENGTH number(10,2) \  
  PAVEMENT char(20)
```

CSV Quick Facts

Format Type Identifier	CSV
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	Directory or File
Feature Type	File base name
Typical File Extensions	.CSV, .CSV.GZ
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	csv_type
Encoding Support	Yes

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	no
circles	no		polygon	no
circular arc	no		raster	no
donut polygon	no		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	no		z values	n/a
none	yes			

Reader Overview

The CSV reader module produces an FME feature for each line in each the CSV files residing in the given directory. The CSV reader first scans the directory for all CSV files that are defined in the mapping file. If IDs lines are specified, the CSV reader processes only the specified files; otherwise, it reads all files in the directory. Optionally a single CSV file can be given in the mapping file. In this case, only that CSV file is read.

Reader Directives

The suffixes shown are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the CSV reader is `CSV`.

DATASET

Required/Optional: *Required*

This is the name of a directory containing one or more CSV files, or the name of a single CSV file. The default extension for CSV files is `.csv`.

Example:

```
CSV_DATASET /usr/data/csv/input
```

Workbench Parameter: *Source Comma Separated Value (CSV) File(s)*

DEF

Required/Optional: *Required*

Each CSV file must be defined before it can be read. The definition contains the file's base name without any of the extensions, followed by the names and types of the attributes. There may be many DEF lines, one for each file to be read. If this is not specified, then all defined CSV files in the directory are read.

The syntax of a CSV DEF line is:

```
<ReaderKeyword>_DEF <baseName> \
  [<attrName> <attrType>]+
```

The following table shows the attribute types supported.

Field Type	Description
<code>char(<width>)</code>	Character fields store fixed length strings. The width parameter controls the maximum number

Field Type	Description
	of characters that can be stored by the field. No padding is required for strings shorter than this width.
date	Date fields store date as character string with the format YYYYMMDD.
number(<width>,<decimals>)	Number fields store single and double precision floating point values. The width parameter is the total number of characters allocated to the field, including the decimal point. The decimals parameter controls the precision of the data and is the number of digits to the right of the decimal.
float	Float fields store floating point values. There is no ability to specify the precision and width of the field.
integer	Integer fields store 32-bit signed integers.
smallint	Small integer fields store 16-bit signed integers and therefore have a range of -32767 to +32767.
logical	Logical fields store TRUE/FALSE data. Data read or written from and to such fields must always have a value of either true or false.

The following table shows all the DEF line directives that are supported by the CSV reader. Each of these directives has the same meaning as the global CSV reader keyword with the same suffix. Any value specified on a DEF line will override values defined for equivalent global directives, as they apply to the table being defined.

DEF Line Directives	Value	Required/Optional
CSV_FIELD_NAMES <yes no>	See FIELD_NAMES for details.	Optional
CSV_FIELD_NAMES_AFTER_HEADER <yes no>	See FIELD_NAMES_AFTER_HEADER for details.	Optional
CSV_SEPARATOR (<separator>)	See SEPARATOR for details.	Optional
CSV_SKIP_LINES <number>	See SKIP_LINES for details.	Optional

DEF Line Directives	Value	Required/Optional
CSV_STRIP_QUOTES <yes no>	See STRIP_QUOTES for details.	Optional
CSV_DUPLICATE_DELIMS <yes no>	See DUPLICATE_DELIMS for details.	Optional
CSV_EXTENSION <extension>	See EXTENSION for details.	Optional
CSV_ENCODING <encoding>	See ENCODING for details.	Optional

The following mapping file fragment defines a CSV file called **roads**. Here we define the '?' as the separator character for columns in the file and we choose not to output the field names to the output file.

```
CSV_DEF roads \
    CSV_SEPARATOR (?) \
    CSV_FIELD_NAMES no \
    id_num number(11,0) \
    type char(20)
```

IDs

Required/Optional: *Optional*

This specification limits the available and defined CSV files read. If no **IDs** are specified, then all defined and available CSV files in the directory are read.

The syntax of the **IDs** keyword is:

```
<ReaderKeyword>_IDs <baseName> \
    <baseName1> ... \
    <baseNameN>
```

The basenames must match those used in **DEF** lines.

The example below selects only the **roads** CSV file for input during a translation:

```
CSV_IDS roads
```

Workbench Parameter: *Feature Types to Read*

FIELD_NAMES

Required/Optional: *Optional*

If the field or column names of the CSV table are specified in the file, then set this value to yes and the names will be extracted from the file. Otherwise, the columns of the CSV table are given default names (i.e. col0, col1, ... , colN) with the setting no. The default is no.

Note: If FIELD_NAMES is set to yes, skip_lines should also be set to skip at least one row, or the first row will be also be processed as a feature. You can also set FIELD_NAMES_AFTER_HEADER to yes. See FIELD_NAMES_AFTER_HEADER below for details.

Values: <yes | no>

FIELD_NAMES_AFTER_HEADER

Required/Optional: *Optional*

If the column/field names is AFTER the header information instead of BEFORE, then you can set **FIELD_NAMES_AFTER_HEADER** to **yes**. Otherwise, by default, the first line of the file will be used as the column/field names.

Notes:

This parameter is ignored if FIELD_NAMES is not set, or it is set to no.

If FIELD_NAMES_AFTER_HEADER is set to yes, SKIP_LINES should also be set to skip at least one row, or the first row will be also be processed as a feature.

Values: <yes | no>

SEPARATOR

Required/Optional: *Optional*

A special field is listed to identify the separator used to divide the fields in the file. By default, a comma is used; however, different one-character separators can also be specified. Tab character separators are indicated by a backslash (\) followed by a "t"; for example:

CSV_SEPARATOR (\t)

Note: There must be a space between **CSV_SEPARATOR** and (<separator>). The begin and end parentheses are optional.

Values: (<separator>)

SKIP_LINES

Required/Optional: *Optional*

This field can be listed to indicate the number of lines to skip at the top of the file. By default, no lines are skipped. Each line skipped is logged to the log file. This is useful if the CSV file contains a header line of field names or other descriptive material that should be skipped.

Values: <number>

Workbench Parameter: *Number of Lines to Skip*

SKIP_FOOTER

Required/Optional: *Optional*

This field can be listed to indicate the number of footer lines to skip at the bottom of the file. By default, no footer lines are skipped. Each footer line skipped is logged to the log file. This is useful if the CSV file contains a footer line of descriptive material that should be skipped.

Values: <number>

Workbench Parameter: *Number of Footer Lines to Skip*

STRIP_QUOTES

Required/Optional: *Optional*

Some CSV files place quotation marks around all values they contain. By setting this special field to **yes**, then these quotes can be stripped from each attribute. The default is no.

Values: <yes|no>

Workbench Parameter: *Strip Quotes from Fields*

DUPLICATE_DELIMS

Required/Optional: *Optional*

This field can be listed to indicate if duplicate delimiters are to be treated as a single delimiter. If set to yes then multiple contiguous delimiters are treated as a single delimiter; otherwise, each delimiter is treated as if it delimits a different field.

Values: <yes|no>

Workbench Parameter: *Skip Duplicate Delimiters*

EXTENSION

Required/Optional: *Optional*

This specifies the file extension to be read or written in. The default is .csv.

Values: <.extension> (Include the period (.) in front of the extension name.)

Default: 0

ENCODING

Required/Optional: *Optional*

This specifies the file encoding to use when reading.

Values: <encoding>

Workbench Parameter: *Character Encoding*

Encodings
UTF-8
UTF-16LE
UTF-16BE
ANSI
BIG5
SJIS
CP437
CP708
CP720
CP737
CP775
CP850
CP852
CP855
CP857
CP860
CP861
CP862
CP863
CP864

Encodings

CP865

CP866

CP869

CP932

CP936

CP950

CP1250

CP1251

CP1252

CP1253

CP1254

CP1255

CP1256

CP1257

CP1258

ISO8859-1

ISO8859-2

ISO8859-3

ISO8859-4

ISO8859-5

ISO8859-6

ISO8859-7

ISO8859-8

ISO8859-9

ISO8859-13

ISO8859-15

Writer Overview

The CSV Writer writes all attributes of a feature to an CSV file. Features of different types are written to different CSV files.

Writer Directives

The suffixes shown are prefixed by the current **<WriterKeyword>** in a mapping file. By default, the **<WriterKeyword>** for the CSV reader is **CSV**.

DATASET

Required/Optional: *Required*

This is the name of a directory containing one or more CSV files. The default extension for CSV files is **.CSV**. To write gzipped files, use **.CSV.gz** as the destination file extension.

An example of the **DATASET** keyword in use is:

```
CSV_DATASET /usr/data/csv/output
```

Workbench Parameter: *Destination Comma Separated Value (CSV) Directory*

DEF

Required/Optional: *Required*

Defines a CSV file. The definition contains the file's base name without any of the extensions, followed by the definitions of the attributes. There may be many DEF lines, one for each file to be written.

The syntax of a CSV DEF line is:

```
<writerKeyword>_DEF <baseName> \
[<attrName> <attrType>]+
```

The attribute types supported by the CSV writer are the same as those listed in the Reader Keywords DEF section. The following DEF line directives are supported by the CSV writer:

DEF Line Directives	Value	Required/Optional
CSV_FIELD_NAMES <yes no>	See FIELD_NAMES below for details.	Optional
CSV_SEPARATOR (<separator>)	See SEPARATOR below for details.	Optional
CSV_EXTENSION <extension>	See EXTENSION below for details.	Optional
CSV_ENCODING <encoding>	See ENCODING for details.	Optional
CSV_END_OF_LINE <encoding>	See END_OF_LINE for details.	Optional

Each of these directives has the same meaning as the global CSV writer keyword with the same suffix. Any value specified on a DEF line will override values defined for equivalent global directives, as they apply to the table being defined.

FIELD_NAMES

Required/Optional: *Optional*

If the field or column names of the CSV table are specified as the first row of the file, set this value to yes and the names will be written to the file. Otherwise, none of the column names will be written to file.

Values: <yes|no>

Default: *no*

Workbench Parameter: *Output field names on first line*

SEPARATOR

Required/Optional: *Optional*

A special field is listed to identify the separator used to divide the fields in the file. By default, a comma is used; however, different one-character separators can also be specified. Tab character separators are indicated by a backslash (\) followed by a "t"; for example:

CSV_SEPARATOR (\t)

Note: There must be a space between `CSV_SEPARATOR` and `(<separator>)`. The begin and end parentheses are optional.

Values: `(<separator>)`

Workbench Parameter: *Separator Character*

EXTENSION

Required/Optional: *Optional*

This specifies the file extension to be written. The default is `.CSV`.

Note: Include the period in front of the extension name. `.CSV.gz` extension will output gzipped files.

Values: `<extension>`

Workbench Parameter: *Extension*

QUOTE_OUTPUT

Required/Optional: *Optional*

This specifies whether the fields written to the CSV file are quoted. If set to `yes`, then every field, including field names, will be quoted. If set to `no`, no fields will be quoted. If set to `if_needed`, fields will be quoted only if they contain a delimiter character.

Note: The meaning of a `yes` value differs slightly between the CSV format writer and the CSV mode of the Relational Table writer.

Values: `yes` | `no` | `if_needed`

Workbench Parameter: *Quote Output Values*

QUOTE_FIELD_NAMES

Required/Optional: *Optional*

This specifies whether the field names written on the first row of the CSV file are quoted. If set to `yes`, then field names will be quoted. If set to `no`, field names will not be quoted.

Values: `yes` | `no`

Default: `no`

Workbench Parameter: *Quote Field Names*

APPEND

Required/Optional: *Optional*

This specifies whether rows will be appended to existing files if a matching CSV file was found in the destination directory.

Values: `yes` | `no`

Default: `no`

Workbench Parameter: *Append to File*

ENCODING

Required/Optional: *Optional*

This specifies the file encoding to use when writing.

Values: <encoding>

Workbench Parameter: *Character Encoding*

Encodings
UTF-8
UTF-16LE
UTF-16BE
ANSI
BIG5
SJIS
CP437
CP708
CP720
CP737
CP775
CP850
CP852
CP855
CP857
CP860
CP861
CP862
CP863
CP864
CP865
CP866
CP869
CP932
CP936
CP950
CP1250
CP1251
CP1252
CP1253
CP1254
CP1255

Encodings
CP1256
CP1257
CP1258
ISO8859-1
ISO8859-2
ISO8859-3
ISO8859-4
ISO8859-5
ISO8859-6
ISO8859-7
ISO8859-8
ISO8859-9
ISO8859-13
ISO8859-15

END_OF_LINE

Required/Optional: *Optional*

This specifies the end of line character to use when writing.

Values: *Macintosh | Windows | Unix | System*

Default: *System*

Workbench Parameter: *Line Termination*

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see [About Feature Attributes](#)), this format adds the format-specific attributes described in this section.

The CSV feature attributes consists of the columns that were in the CSV table. All **CSV** features contain a **csv_type** attribute, which is always set to **csv_none** as there is no geometry to CSV features. This represents that the feature was generated from a CSV file.

Attribute Name	Contents
csv_type	The CSV geometric type of this entity. Range: csv_none Default: csv_none

Danish DSFL Reader

The DSFL Reader module enables FME to read the DSFL basic format which is a genuine subset of the full Danish National Format (DSFL). This chapter assumes familiarity with that format.

Overview

DSFL is an ASCII format, widely used in Denmark for exchanging Geographic Information System (GIS) data between different systems. The information within the DSFL file is contained in these four sections:

- **header section** – contains global information common to all data
- **origin section** – where the accuracy and the origin of the data are specified
- **data section** – where data is referenced by features and where the spatial and non-spatial data for the features are contained
- **stop code** – this is how the end of the DSFL data set is signalled

The following file extensions are commonly used:

Filename Extension	Content
.dsf, .asc, .txt and others	Feature geometry and attribution data

DSFL Quick Facts

Format Type Identifier	DSFL
Reader/Writer	Reader
Licensing Level	Base
Dependencies	None
Dataset Type	File
Feature Type	Geometry type
Typical File Extensions	.dsf, .fla
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Never
Schema Required	Not applicable
Transaction Support	No
Geometry Type	dsfl_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	yes
line	yes		z values	yes
none	no			

Reader Overview

The DSFL reader reads the header information from the DSFL file being processed, and extracts the parameters required to determine the coordinate system and sequence use. The dimension of the input file is also known after the coordinate sequence had been determined. The reader then returns each read-in feature with its attributes to the FME for processing. The DSFL reader doesn't have any requirements for definition statements.

Each feature returned by the DSFL reader has its feature type set to one of the following: `dsfl_point`, `dsfl_line`, `dsfl_polygon`, `dsfl_aggregate`, `dsfl_none`, or `dsfl_header`.

Reader Directives

The suffixes shown are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the DSFL reader is `DSFL`.

DATASET

Required/Optional: *Required*

The file name of the input DSFL file.

Example:

```
D MDF_DATASET /usr/data/dmdf/input.dd1
```

Workbench Parameter: *Source DSFL File(s)*

OUTPUT_ORIGINS

Required/Optional: *Optional*

Determines whether or not the origin data is output as a separate features. If the value is YES, then origin data is output as `dsfl_origin` feature type with each feature having its unique index number in `dsfl_record_index_number` attribute. If the value is NO, then the origin data is merged with other data features.

Range: YES | NO

Default: NO

Example:

```
D MDF_RASTER_POINT_FEATURE_CODE HA3500000
```

Workbench Parameter: *Output Origin Data*

KEEP_Z_NULL

Required/Optional: *Optional*

This directive determines whether or not to preserve the NULL value placeholder for z coordinates defined by %H9 header tag. If the value is YES, the z coordinate of the geometry will be unchanged; if the value is NO, it will be set to 0.

For example, if the %H9 tag is set to -99.00 and, while reading a feature, it encounters a z coordinate of -99.00, then based on the value of this keyword, either the z coordinate will remain as 99.00 (if value is YES) or changed to 0.0 (if the value is NO).

Range: *YES | NO*

Default: *YES*

Workbench Parameter: *Preserve Original Null Value for Z*

SPLINE_EDGE_TOLERANCE

Required/Optional: *Optional*

After the DSFL reader has converted splines to straight lines, this directive can be used to remove extraneous points. Real values from 0 and up are acceptable. If a negative number is input, the DSFL reader will ignore it, and not generalize the line. This will only be used if **SPLINE_TO_POINTS** was set to at least 1.

In Workbench, this functions like a LineGeneralizer transformer using the Douglas algorithm.

A recommended use is to set **SPLINE_TO_POINTS** to a moderately high number, such as 100, and then generalize to an acceptable precision. This will keep the overall number of points generated down, but will ensure precision is available where it is needed to keep the error down.

Example:

The following example sets the edge tolerance for generalizing lines to 2.5:

```
DSFL_SPLINE_EDGE_TOLERANCE 2.5
```

Workbench Parameter: *Edge tolerance for generalizing splines*

SPLINE_TO_POINTS

Required/Optional: *Optional*

The DSFL reader converts all spline curves into straight lines by inserting intermediate points. This directive specifies the number of intermediate points to be inserted. Integer values from 0 to 10 are acceptable. If an illegal value is entered, the DSFL reader will automatically use the default value of 3. The recommended range is 0 to 10.

Example:

The following example sets the number of intermediate points to be calculated to four:

```
DSFL_SPLINE_TO_POINTS 4
```

Range: *>=0*

Default: *3*

Workbench Parameter: *Points Per Segment*

COMMA_IS_A_DELIMITER

Required/Optional: *Optional*

When set to "Yes", this directive will tell the DSFL reader to also use the comma (,) as a delimiter when separating DSFL tokens as well as when parsing attributes. This will remove commas from attributes (for example, "Vancouver, Canada" becomes "Vancouver Canada"). Therefore, by setting this value to "No," it will keep the commas in the attributes.

It should be noted that if the DSFL file's tokens or coordinate points were separated with commas, then the reader may not work properly. However, since FME only supports the basic version of DSFL (and commas are not allowed to separate tokens and coordinates in the basic version), this should never be a problem.

The default value of this directive is "No". However, if the directive is missing, then it will implicitly set the value to "Yes" so that workspaces created prior to the addition of this directive will continue to exhibit the same behavior as before.

Example:

```
DSFL_COMMA_IS_A_DELIMITER NO
```

Range: Yes | No

Default: No

Workbench Parameter: *Use Comma as a Delimiter*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

✳ Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

✳ Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]

* Workbench Parameter

Clip To Envelope

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

DSFL features consist of geometry and attributes. When reading-in a feature, the DSFL reader holds a set of currently active DSFL data fields. These active data fields are controlled by definitions of the %D token, found in the DSFL input file. The active data fields are given as attributes to the feature being read. The active set of data fields may be empty. In this case, the feature has no DSFL data fields attributes. The name for these attributes are of the form Dx, where x is a positive integer.

If the feature references any of the accuracy and origin definitions processed at the beginning of the file, these definitions will also become attributes for the referencing feature.

All DSFL features contain a dsfl_type attribute that identifies the geometric type. Depending on the geometric type, the features may contain additional attributes that are specific to them.

Attribute Name	Contents
dsfl_type	The DSFL geometric type of this feature. Range: dsfl_point dsfl_line dsfl_polygon dsfl_text dsfl_aggregate dsfl_none dsfl_header dsfl_origin Default: No default

General Attributes

Attributes specific to each dsfl_type are described in the next sections. All DSFL features, except those features having dsfl_text and dsfl_header as the value of their dsfl_type, may contain the following attributes:

Attribute Name	Contents
dsfl_class	This is the DSFL feature code class.
dsfl_subclass	This is the DSFL feature code subclass.
dsfl_origin_ND1	Acronyms for production, also known as data generation,

Attribute Name	Contents
	method Range: DU DF DL SK SL UU FF LL Default: No default
dsfl_origin_ND11	Standard deviation for plane coordinates, in metres with decimals. Default: No default
dsfl_origin_ND12	Standard deviation for height coordinate, in metres with decimals Default: No default
dsfl_origin_ND21	Date of base map generation Range: yymmdd Default: No default
dsfl_origin_ND22	Date of land surveying Range: yymmdd Default: No default
dsfl_origin_ND23	Date for photo flight Range: yymmdd Default: No default
dsfl_origin_ND32	Scale of photogrammetric photos Default: No default
dsfl_origin_ND41	Producer of digital data Range: String, maximum 40 characters Default: No default
dsfl_origin_ND51 to dsfl_origin_ND59	Descriptive text Range: String, maximum 40 characters Default: No default

Points

dsfl_type: dsfl_point

DSFL point features specify a single x and y coordinate for two-dimensional (2D) data or a single x, y, and z coordinate for three-dimensional (3D) data. Point features may have the following additional special attributes associated with them.

Attribute Name	Contents
dsfl_point_rotation	DSFL angles are defined as grades. The DSFL reader automatically converts these into degrees. The degrees are measured counterclockwise from horizontal. Range: 0.0 .. 360.0 Default: 0.0

Attribute Name	Contents
Z	When the DSFL data is 3D, the <code>dsfl_point</code> will contain this attribute having as its value the third, or z, coordinate of the point.

Lines

dsfl_type: dsfl_line

DSFL line features specify linear features by a sequence of x and y coordinates for 2D data or by a sequence of x, y, and z coordinates for 3D data.

Polygons

dsfl_type: dsfl_polygon

DSFL polygon features specify polygon features by a sequence of x and y coordinates for 2D data or by a sequence of x, y, and z coordinates for 3D data. The first and last coordinates of the polygon are equal.

Text

dsfl_type: dsfl_text

DSFL text features are used to specify annotation information. Each text feature has a single x and y coordinate for 2D data or a single x, y, and z coordinate for 3D data. The following table lists the special FME attribute names for the DSFL text feature.

Attribute Name	Contents
dsfl_text_code	The DSFL data field code.
dsfl_text_value	The value for the DSFL data field.
dsfl_rotation	DSFL angles are defined as grades. The DSFL reader automatically converts these into degrees measured counterclockwise from horizontal. Range: 0.0 .. 360.0 Default: 0.0
dsfl_text_justification	Indicates the position of the text coordinate in relation to the text. Range: dsfl_top_left dsfl_top_center dsfl_top_right dsfl_middle_left dsfl_middle_center dsfl_middle_right dsfl_bottom_left dsfl_bottom_center dsfl_bottom_right Default: No default

Aggregate

dsfl_type: dsfl_aggregate

DSFL aggregates are a collection of **dsfl_line** or **dsfl_polygon** features. The geometry of the **dsfl_aggregate** feature is homogeneous. The **dsfl_type** attribute of this feature will be set to **dsfl_polygon** if all composing features are polygons. If all composing features are lines, it will be set to **dsfl_line**.

None

dsfl_type: dsfl_none

This is a DSFL feature with no geographic representation.

Header

dsfl_type: dsfl_header

This DSFL feature contains the metadata stored in the header section for the input DSFL file. The feature contains no geometry. The following table lists the attributes that this feature contains. Basically, all header tokens with their values become attributes for this feature.

Attribute Name	Contents
H0	Character string specifying the three special Danish characters in upper- and lowercase.
H1	Plane coordinate system—this value is used to set the coordinate system on the features Range: S34J S34S S45B U32 U33 U32W U33W LOK
H2	Acronym for the height coordinate system Range: DNNGI
H3	Coordinate sequence Range: XY XYZ YX YXZ NE NEH
H9	Specific value for “no available height” Range: Real
H11	Supplier’s company name Range: String
H12	Supplier’s address Range: String
H13	Supplier’s postal code Range: String
H14	Supplier’s postal district Range: String
H15	Supplier’s phone number Range: String
H16	Supplier’s fax number Range: String

Attribute Name	Contents
H41	Date and time of generation for the data set Range: <i>yymmdd hhmm</i>
H58	Data content Range: <i>Basis-udgave 970901</i> for the DSFL basic format
H59	Version date of DSFL format Range: <i>yymmdd</i>

Danish UFO Reader/Writer

Format Notes:

This format is not supported by FME Base Edition.

The Danish UFO Reader/Writer allows FME to read and write UFO files. The UFO is a published ASCII format used by the National Survey and Cadastre of Denmark.

Overview

The UFO format provides facilities for carrying a variety of metadata together with the actual feature data. Metadata can even be associated with individual vertices. The UFO reader and writer provide a complete set of facilities for reading and writing this metadata, however, custom mapping files are required to populate the metadata fields. A custom mapping file is provided with FME to do UFO to UFO translation in a lossless manner.

UFO files store both feature geometry and attribution. UFO feature coordinates are always measured in centimetres.

A UFO file has the following file name extension:

File Name Extension	Contents
.ufo	UFO format file

The extension is added to the basename of the UFO file.

UFO Quick Facts

Format Type Identifier	UFO
Reader/Writer	Both
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	Feature role
Typical File Extensions	.ufo
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	ufo_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	yes
line	yes		z values	yes
none	yes			

Reader Overview

The UFO reader module produces FME features for all data held in the UFO file. The UFO reader extracts data from the file one row at a time, producing FME features from the file before passing them on to the rest of the FME for further processing. The features are produced in the order they are read from the source file. The various metadata features are emitted first, followed by the feature data. When the file is exhausted, the UFO reader terminates.

Reader Directives

The directives listed below are processed by the UFO reader. The suffixes listed are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the UFO reader is **UFO**.

DATASET

Required/Optional: *Required*

This is the name of an UFO file. The extension for UFO files is **.ufo**.

An example of the **DATASET** keyword in use is:

```
UFO_DATASET /usr/data/ufo/input.ufo
```

Workbench Parameter: *Source Danish UFO File(s)*

Writer Overview

The UFO writer creates and writes feature data to an UFO file specified by the **DATASET** keyword. Existing UFO files with the same name as the specified file are overwritten with the new feature data.

The UFO writer process two kinds of features: regular features and meta features. Regular features are features such as points and lines, which are commonly seen in FME formats. Meta features are features specifically created for the UFO writer so that it can receive information for the origin for objects, label types, and the other metadata types stored in Sections 2 through 5 of the format. Note that the order in which the meta features are received by the writer is extremely important. They must be received in the order listed in the meta feature section under the Feature Representation section.

Writer Directives

The directives listed below are processed by the UFO writer. The suffixes shown are prefixed by the current **<WriterKeyword>** in a mapping file. By default, the **<WriterKeyword>** for the UFO writer is **UFO**.

DATASET

Required/Optional: *Required*

The **DATASET** directive operates in the same manner as it does for the UFO reader.

Workbench Parameter: *Destination Danish UFO File*

DEF

Required/Optional: *Required*

SOURCE_IS_UFO

The translation from UFO to UFO is handled differently from a translation from other formats to UFO. In such a situation, the input data stream will contain additional information that the UFO writer can make use of. This directive is used in the "ufo2ufo.fme" mapping file provided in the FME installation directory.

An example of the SOURCE_IS_UFO keyword in use is:

```
UFO_SOURCE_IS_UFO yes
```

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Each UFO element, regardless of its geometry type, shares a number of other parameters, as described in the following tables. Subsequent sections will describe parameters specific to each of the supported element types.

Attribute Name	Contents
ufo_type	The UFO geometric type of this entity. Range: ufo_point ufo_line ufo_polygon ufo_text ufo_none Default: No default

The main object line:

Attribute Name	Contents
ufo_obj_code	An integer that expresses the object code. Range: 32 bit integer Default: 0
ufo_db	A text string which indicates the database the object connects to. Range: Max. of 80 characters Default: <i>no_database</i>
ufo_key	The unique identification in a database expressed as a whole number. 0 means that the object has not yet been assigned a unique identification. Range: 32 bit integer Default: 0
ufo_obj_origin	A whole number which indicates the object origin group the object belongs to.

Attribute Name	Contents
	Range: 32 bit integer Default: 0

The object reference:

Attribute Name	Contents
ufo_obj_ref{<number>}.group	A integer that refers to an object reference type group. Note: The <number> refers to the group of object reference description. Directives that belong to the same description should have the same number. The <number> is a positive integer which starts at 0. Range: 32 bit integer Default: 0
ufo_obj_ref {<number>}.database	A text string which indicates the database being referred to. Range: max. 80 characters Default: no_database
ufo_obj_ref {<number>}.key	A whole number which indicates the object being referred to in the database. Range: 32 bit integer Default: 0

The label description:

Attribute Name	Contents
ufo_label {<number>}.ufo_text_type	A character which indicates if the label has a full description or a free text description. The allowable range for the string are "L" and "F" corresponding to "full" and "free". Note: The <number> refers to the group of label description. Directives that belong to the same description should have the same number. The <number> is a positive integer which starts at 0. Range: L or F Default: No default.
ufo_label{<number>}.ufo_group	A whole number that refers to a label type group. Range: 32 bit integer Default: 0
ufo_label{<number>}.ufo_northing	An integer which refers to the north-coordinate of the label.

Attribute Name	Contents
	Range: 32 bit integer Default: 0
ufo_label{<number>} .ufo_easting	An integer which refers to the east-coordinate of the label. Range: 32 bit integer Default: 0
ufo_label{<number>} .ufo_height	An integer which refers to the height-component of the label. Range: 32 bit integer Default: 0
ufo_label{<number>} .ufo_pt_origins	A whole number which refers to the point origin group. Range: 32 bit integer Default: 0
ufo_label{<number>} .ufo_font	A whole number code for the label text font. Range: 32 bit integer Default: 0
ufo_label{<number>} .ufo_caps	Specifies the capitalization of the label text. Range: 32 bit integer 0 = Mix of lower and uppercase letters, 1 = Only upper case letters. Default: 0
ufo_label{<number>} .ufo_text_size	A whole number that refers to the height of the label. Range: 32 bit integer Default: 0
ufo_label{<number>} .ufo_color	A whole number that refers to the color of the label. Range: 32 bit integer Default: 0
ufo_label{<number>} .ufo_justification	A code in DSFL that indicates the justification of the label. Range: 32 bit integer 1 = TL (top left), 2 = TM (top middle), 3 = TR (top right), 4 = ML (middle left), 5 = MM (middle middle), 6 = MR (middle right), 7 = BL (bottom left), 8 = BM (bottom middle), 9 = BR (bottom right) Default: 3

Attribute Name	Contents
ufo_label{<number>} .ufo_orientation	A whole number that indicates the orientation of the label. Range: 32 bit integer Default: 0
ufo_label{<number>} .ufo_spacing	A whole number that indicates how many extra spaces are needed. Range: 32 bit integer Default: 0
ufo_label{<number>} .ufo_number	A whole number used in SNOSOR to indicate the place/appellative-flag. Range: 32 bit integer 0 = appellative, 1 = place name. Default: 0
ufo_label{<number>} .ufo_text_string	The actual text for the label. Range: max. 80 characters Default: no_label

Points

ufo_type: ufo_point

A multi-point feature in UFO turns into an aggregate point feature in FME. In this case, the feature will have the attribute **ufo_pt_origins{<number>}** for each point with the **<number>** being the coordinate index. The value of this attribute is a comma separated value list of point origin numbers with the first one being the point origin number for the first point and the second one being the point origin number of the second point and so on.

Attribute Name	Contents
ufo_pt_origins	This is the point origin number that refers to the meta feature ufo_meta_pt_origin's ufo_group_id . Range: 32 bit integer Default: 0

Lines

ufo_type: ufo_line

A multi-line feature in UFO turns into an aggregate line feature in FME. In this case, the feature will have the attribute **ufo_pt_origins{<number>}** for each line with the **<number>** being the line index. The value of this attribute is a comma separated value of point origin numbers with the first one being the point origin number for the first point and the second one being the point origin number of the second point. Each comma-separated value list contains the point origin number for a line.

Attribute Name	Contents
ufo_pt_origins	This a comma-separated string that holds the list of ufo_pt_origins as describe for Points. If all points of the line has the same value for ufo_pt_

Attribute Name	Contents
	<p>origins then this string will only hold that single value.</p> <p>Range: 32 bit integer</p> <p>Default: 0</p>

Polygons

ufo_type: ufo_polygon

A polygon with more than one line is turned into a donut feature in FME. There is also a multi-polygon feature in UFO that turns into an aggregate polygon feature. In both cases, the feature will have the attribute `ufo_pt_origins{<number>}` for each polygon, with **<number>** being the polygon index. The order of the polygons are important.

The value of `ufo_pt_origins{<number>}` is a comma separated value of point origin numbers with the first one being the point origin number for the first point and the second one being the point origin number of the second point. Each comma-separated value list contains the point origin number for a polygon.

Attribute Name	Contents
ufo_pt_origins	<p>This a comma-separated string that holds the list of <code>ufo_pt_origins</code> as describe for Points. If all points of the polygon has the same value for <code>ufo_pt_origins</code> then this string will only hold that single value.</p> <p>Range: 32 bit integer</p> <p>Default: 0</p>

Text

ufo_type: ufo_text

Text features are not native UFO features. They are created during translation from UFO to other formats, in order to convert the text information which can be part of the UFO feature label descriptions (Here, we call this UFO feature the original feature). The text feature is a clone of the original feature with the addition of the attributes **`ufo_text_size`**, **`ufo_text_string`** and **`ufo_orientation`** which are listed in the label description part of the Feature Representation Section, but do no contain the `ufo_label{<number>}` prefix. The coordinates for the text features are the first set of coordinates of the original feature if it is not available through the original features label description.

Meta Features

ufo_type: ufo_none

Note that the meta features must be created in the mapping file in the exact order as it is listed here. Otherwise, an error is output. In addition, all regular feature must be created before the meta features in order for the UFO Writer to function correctly.

Header

This is the header feature of the ufo file. The feature type of this feature is **`ufo_meta_header`**. The supported attributes are as followed:

Attribute Name	Contents
ufo_coord_sys	<p>A text string which describes the coordinate system used.</p> <p>Range: Text string with max length of 25</p> <p>Default: <code>ufo_coord_sys</code></p>

Origin and Definitions of Precision for Points

This is the Origin and Definitions of Precision for Points feature of the ufo file. The feature type of this feature is **ufo_meta_pt_origin**. The supported attributes are as followed:

Attribute Name	Contents
ufo_group_id	A unique number for KMS which expresses a number for the group. Range: Integer Default: 0
ufo_attr{<number>}	Text strings which contain information belonging to the group. <number> is a positive integer which expresses the line number of the text strings. Range: Consecutive integer starting from 0 Default: 0

Label Types

This is the Label Types feature of the ufo file. The feature type of this feature is **ufo_meta_label**. The supported attributes are exactly the same as those supported by Origin And Definitions of Precision For Points feature.

Object Reference Types

This is the Object Reference Types feature of the ufo file. The feature type of this feature is **ufo_meta_ref**. The supported attributes are exactly the same as those supported by Origin And Definitions of Precision For Points feature.

Origin for objects

This is the Origin for Object feature of the ufo file. The feature type of this feature is **ufo_meta_obj_origin**. The supported attributes are exactly the same as those supported by Origin And Definitions of Precision For Points feature.

dBase (DBF) Reader/Writer

The dBase Format (DBF) Reader/Writer allows FME to read and write data in the DBF format.

All DBF files are formatted according to the dBase III specification. The DBF Reader/Writer reuses the Relational Table Reader/Writer's CSV to expand its capabilities from being usable only in the FME Universal (Quick) Translator to also being usable in FME Workbench and Universal Viewer.

Note: Any single DBF file can have a maximum file size of 2 GB, a limit imposed by the dBase III specification. Files larger than 2 GB may be readable, but not officially supported. Files larger than 2 GB are not writable, and will produce an error message.

Overview

A DBF file defines a single table within a database. The feature attribution of a FME feature are the columns and values of the DBF database table. There is no geometry or dimension to the features created from the DBF files. They are all undefined. Therefore, none of the features created are viewable because there is no graphical component to the features.

DBF files store only feature attribution. The DBF format has one physical file. The extension is added to the basename of the DBF file.

DBF Quick Facts

Format Type Identifier	DBF
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	Directory or File
Feature Type	File base name
Typical File Extensions	.dbf
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	dbf_type
Encoding Support	Yes

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	no
circles	no		polygon	no
circular arc	no		raster	no
donut polygon	no		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	no		z values	n/a
none	yes			

Reader Overview

The DBF reader module produces FME feature for each table entry held in the DBF files residing in the given directory. The DBF reader first scans the directory for all DBF files which are defined in the mapping file. It processes only the specified files if IDs lines are available. Otherwise, all files in the directory are read. The DBF reader extracts data from the file one row at a time, producing FME features before passing them on to the rest of the FME for further processing. When the file is exhausted, the DBF reader moves on to the next file in the directory. Optionally a single DBF file can be given as the dataset. In this case, only the single file is read. The reader supports dBASE III, dBASE IV and FoxPro files.

Reader Directives

The suffixes listed are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the DBF reader is `DBF`.

DATASET

Required/Optional: *Required*

This is the name of a directory containing one or more DBF files, or a single DBF file. The extension for DBF files is `.dbf`.

Example:

```
DBF_DATASET /usr/data/dbf/input
```

Workbench Parameter: *Source dBASE (DBF) File(s)*

DEF

Required/Optional: *Required*

Each DBF file may optionally be defined before it can be read. The definition specifies the base name of the file, and the names and the types of all attributes.

Example:

```
<ReaderKeyword>_DEF <baseName> \  
[<attrName> <attrType>]+
```

The following table shows the attribute types supported.

Field Type	Description
<code>char(<width>)</code>	Character fields store fixed length strings. The width parameter controls the maximum

Field Type	Description
	number of characters that can be stored by the field. No padding is required for strings shorter than this width.
date	Date fields store date as character strings with the format YYYYMMDD.
number(<width>,<decimals>)	Number fields store single and double precision floating point values. The width parameter is the total number of characters allocated to the field, including the decimal point. The decimals parameter controls the precision of the data and is the number of digits to the right of the decimal.
logical	Logical fields store TRUE/FALSE data. Data read or written from and to such fields must always have a value of either true or false.
memo	The reader can read dBASE III, IV and Fox-Pro memo fields. When writing, only dBASE III format memo fields are supported.

The example below is a DEF line for the trees DBF file that has the attributes name and id_number:

```
DBF_DEF trees \
    name char(30) \
    id_number number(11,0)
```

Workbench Parameter: <WorkbenchParameter>

IDs

Required/Optional: *Optional*

This optional specification limits the available and defined DBF files read. If no IDs are specified, then all defined and available DBF files are read.

The syntax of the IDs keyword is:

```
<ReaderKeyword>_IDs <baseName> \
    <baseName1> ... \
    <baseNameN>
```

The basenames must match those used on the DEF lines.

The example below selects only the pipeline DBF file for input during a translation:

```
DBF_IDS pipeline
```

ENCODING

Required/Optional: *Optional*

This optional specification controls which character encoding is used to interpret text attributes from the DBF file. If the value is not set, then the character encoding will be automatically detected from the source DBF file. If the value is set, it will take precedence over the automatically detected character encoding.

This directive is useful when the character encoding information stored in the DBF file is missing or incorrect.

Example:

<ReaderKeyword>_ENCODING <character encoding>

Workbench Parameter: *Character Encoding*

Parameter	Description
<character encoding>	<p>The character encoding to use when interpreting text attributes. Must be set to any of the following values:</p> <ul style="list-style-type: none"> ANSI - this means use the "current OS language" BIG5 EUC ISO OEM SJIS UTF-8 CP437 CP708 CP720 CP737 CP775 CP850 CP852 CP855 CP857 CP860 CP861 CP862 CP863 CP864 CP865 CP866 CP869 CP932 CP936 CP950 CP1250 CP1251 CP1252 CP1253 CP1254

Parameter	Description
	CP1255 CP1256 CP1257 CP1258 ISO8859-1 ISO8859-2 ISO8859-3 ISO8859-4 ISO8859-5 ISO8859-6 ISO8859-7 ISO8859-8 ISO8859-9 ISO8859-13 ISO8859-15

TRIM_PRECEDING_SPACES

Required/Optional: *Optional*

This option specifies whether the reader should trim preceding spaces of attribute values. If the option is set to YES, then preceding spaces in attribute values will be discarded. If the option is set to NO, then preceding spaces will be left intact. The default value is YES.

Workbench Parameter: *Trim Preceding Spaces*

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

✳ Workbench Parameter

Additional Attributes to Expose

Writer Overview

The DBF Writer writes all attributes of a feature to a DBF file. Features of the different feature types are written to different DBF files.

Writer Directives

The suffixes shown are prefixed by the current `<WriterKeyword>` in a mapping file. By default, the `<WriterKeyword>` for the DBF writer is `DBF`.

The DBF writer processes the `DATASET` and `DEF` keywords as described in the *Reader Keywords* section above. However, it does not make use of the `IDS` keywords.

Unlike the reader, the writer requires a `DEF` line for each file being written.

The `ENCODING` directive is used to specify which character encoding should be used when writing text attributes into DBF files. If the value of this directive is not set, the current OS language is used. The syntax of the `ENCODING` writer directive is the same as the `ENCODING` reader directive, as described in the *Reader Directives* section.

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

The DBF feature attributes consists of the column name that were in the DBF table. All DBF features contain a `dbf_type` attribute, which is always set to `dbf_none` as there is no geometry to DBF features. This shows that the feature was generated from a DBF file.

Attribute Name	Contents
dbf_type	The DBF geometric type of this entity. Range: dbf_none Default: dbf_none

Digital Line Graph (DLG) Reader

Format Notes:

This format is not supported by FME Base Edition.

The Digital Line Graph (DLG) reader enables FME to import Level 3 DLG data and export it to any of the FME output formats. DLG is a published ASCII format developed by the United States Geological Survey (USGS) Federal Agency and is intended to assist in data exchange with the National Digital Cartographic Data Base (NDCDB).

The DLG reader supports all three distinct types of DLG data:

- large-scale DLG data (1:24,000 scale)
- intermediate-scale DLG (1:100,000 scale)
- small-scale DLG data (1:2,000,000 scale)

The three scales of DLG data are physically formatted into files in one of these ways: standard, optional, and graphics formats. FME supports both the standard and the optional DLG distribution formats. However, the graphics format is not supported. Most DLG data is distributed in the optional format.

Overview

DLG data files consist of ASCII fixed field records. The records may or may not be stored with embedded carriage returns or end of line markers. The DLG reader intelligently determines the end of each record, and interprets files with or without explicit end of record markers.

The DLG file structure was designed to accommodate all categories of spatial data represented on a conventional line map. Node, line, and area data types are present within the DLG format, along with linkages and attribute codes.

Linkages are references to other features within the same DLG data set, used in a variety of contexts.

DLG files do not explicitly store attribute values but use a feature coding approach in which unique feature codes are assigned to the different types of features stored within the data set. Each geometric entity present in a DLG file may be assigned major and minor attribute codes which always appear as a pair. Together these codes often form complex relationships to assign specific attributes for each feature. The attribute coding scheme is designed to accommodate basic cartographic data categories such as hypsography, hydrography, or political and cultural features, as well as additional thematic data categories. The FME supports a maximum of 12 attribute code pairs per feature.

The FME looks for an extension of either `.dlg` or `.opt` for the input DLG files, but accepts any DLG file as input regardless of file name or extension.

Although mapping files may be created from scratch to work with the features as presented directly by the DLG reader, starting with an FME generated mapping file provides an easy way to harness the enhanced semantic interpretation of all attribute codes and linkages built into the FME distribution. This section will first outline the features and attributes produced directly by the DLG reader. These features and attributes produced by using an FME generated mapping file are presented at the end of this section.

DLG Quick Facts

Format Type Identifier	DLG
Reader/Writer	Reader
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	Feature category
Typical File Extensions	.dlg, .opt
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Never
Schema Required	No
Transaction Support	No
Geometry Type	dlg_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	yes		z values	no
none	no			

Reader Overview

The DLG reader simply opens the input file and immediately starts reading features and returning them to the rest of the FME for processing. The reader doesn't have any requirement for definition statements as there are no user-defined attributes.

Each feature returned by the DLG reader has its feature type set to one of the following: **dlg_point**, **dlg_line**, or **dlg_area**.

Reader Directives

The suffixes shown are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the DLG reader is `DLG`.

DATASET

Required/Optional: *Required*

The value for this directive is the file containing the DLG dataset to be read.

Example:

```
SHAPE_DATASET /usr/data/shape/92i080
```

Workbench Parameter: *Source Digital Line Graph (DLG) File(s)*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The `COORDINATE_SYSTEM` directive, which specifies the coordinate system associated with the data to be read, must always be set if the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` to the reader `COORDINATE_SYSTEM` prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

DLG features consist of geometry, linkages, and attribute code information. All DLG FME features contain the **dlg_type** attribute, which identifies the geometric type as well as several other standard attributes and are listed in the following table.

Attribute Name	Contents
dlg_type	The DLG geometric type of this entity. Range: dlg_point dlg_line dlg_area Default: No default
dlg_element_number	The element's internal identification number. The numbers are unique, positive, and sequential within each element type. Range: 1 - 32000
dlg_record_type	The character element type of the feature. Valid values include: N = Node Element L = Line Element A = Area Element
dlg_num_text_characters	Number of pairs of text characters attached to the feature. Although this field is present within the DLG format, it is not currently used. Range: 1 - 32000

Attribute Name	Contents
dlg_linkage{#}	<p>A list of linkages. These values refer to the features by their <code>dlg_element_number</code>. These linkages have different uses depending on their context. For example, a linkage list on an area feature refers to the line features that form the boundary of the area.</p> <p>Note: For area features, linkages with a value of zero are not included in this list.</p> <p>Range: 1 - 32000</p>
dlg_num_attribute_codes	<p>Number of attribute codes attached to the feature.</p> <p>Range: 1 - 32000</p>
dlg_attribute_code{#}.major	<p>A list of major attribute codes. This list will have a maximum of 12 entries.</p> <p>Range: 0 - 999</p>
dlg_attribute_code{#}.minor	<p>A list of minor attribute codes. This list will have a maximum of 12 entries.</p> <p>Range: 0 - 9999</p>
dlg_attribute_code{#}. padmajor	<p>This list is identical to the <code>dlg_attribute_code{#}.major</code> list except all values in this list are padded with zeros to exactly three character places.</p> <p>For example, if <code>dlg_attribute_code{0}.major</code> was 90, <code>dlg_attribute_code{0}.padmajor</code> would be 090.</p> <p>Range: 000 - 999</p>
dlg_attribute_code{#}. padminor	<p>This list is identical to the <code>dlg_attribute_code{#}.minor</code> list except all values in this list are padded with zeros to exactly four character places.</p> <p>For example, if <code>dlg_attribute_code{0}.minor</code> was 214, <code>dlg_attribute_code{0}.padminor</code> would be 0214.</p> <p>Range: 0000 - 9999</p>

Attribute Name	Contents
<code>dlg_attribute_code{#}.partminor1</code>	<p>This list contains the first character of the corresponding entry in the <code>dlg_attribute_code{#}.padminor</code> list.</p> <p>For example, if <code>dlg_attribute_code{0}.padminor</code> was 0214, <code>dlg_attribute_code{0}.partminor1</code> would be 0.</p> <p>Range: 0 - 9</p>
<code>dlg_attribute_code{#}.partminor2</code>	<p>This list contains the second character of the corresponding entry in the <code>dlg_attribute_code{#}.padminor</code> list.</p> <p>For example, if <code>dlg_attribute_code{0}.padminor</code> was 0214, <code>dlg_attribute_code{0}.partminor2</code> would be 2.</p> <p>Range: 0 - 9</p>
<code>dlg_attribute_code{#}.partminor34</code>	<p>This list contains the third and fourth characters of the corresponding entry in the <code>dlg_attribute_code{#}.padminor</code> list.</p> <p>For example, if <code>dlg_attribute_code{0}.padminor</code> was 0214, <code>dlg_attribute_code{0}.partminor34</code> would be 14.</p> <p>Range: 0 - 9</p>
<code>dlg_code_list</code>	<p>A text string containing all major and minor codes assigned to this feature, in the following format:</p> <p>Range: <null> <code list> <code list> = (<major code>-<minor code> [,<major code>-<minor code>]*)</p> <p>For example, if the feature had major and minor code pairs of 180/201, 180/605, and 180/210, the string value of <code>dlg_code_list</code> attribute would be "(180-201,180-605,180-210)"</p>

Depending on the geometric type, the feature may contain additional feature coding attributes specific to the geometric type. These are described in subsequent sections.

Points

dlg_type: `dlg_point`

DLG point features specify a single x and y coordinate. While the DLG format does allow for points to be defined as degenerate lines—lines containing two identical points—the DLG reader converts these into standard points with a single set of coordinates.

There is one attribute specific to point features.

Field Name	Description
dlg_num_linkage_records	The number of linkages associated with this feature. This number indicates the number of entries in the dlg_linkage{#} attribute list. Range: 1 - 32000

Lines

dlg_type: dlg_line

DLG line features represent two-dimensional linear features.

There are several attributes specific to line features.

Field Name	Description
dlg_num_coordinates	The number of coordinates associated with this line feature. Range: 1 - 32000
dlg_starting_node	This number refers a node feature which is located at the initial point of the line. The value refers to the feature by its dlg_element_number. Range: 1 - 32000
dlg_ending_node	This number refers a node feature which is located at the final point of the line. The value refers to the feature by its dlg_element_number. Range: 1 - 32000
dlg_left_area	This number refers an area feature which is located to the immediate left of the line. The value refers to the feature by its dlg_element_number. Range: 1 - 32000
dlg_right_area	This number refers an area feature which is located to the immediate right of the line. The value refers to the feature by its dlg_element_number. Range: 1 - 32000

Areas

dlg_type: dlg_area

DLG area features represent polygonal features in 2D. These features are actually point features with one x and one y coordinate. This coordinate location may have little utility, as the boundary of the area is specified indirectly through the use of the dlg_linkage{} list attribute. Each entry in this list refers to a dlg_line which, together, form the boundary of the area. Additional attributes assigned to this area are attached to the original dlg_area feature.

There are several attributes specific to area features.

Field Name	Description
dlg_num_islands	The number of islands or holes within this area feature. Range: 1 - 32000
dlg_num_linkage_records	The number of entries in the <code>dlg_linkage{#}</code> list attribute. This list contains references to the line features that define the border of the area. Note: linkages with a value of zero are not included in this count. Range: 1 - 32000
dlg_num_points_area_list	The number of coordinates associated with the linear features necessary to define the border of this area feature. Range: 1 - 32000

Features Created by Generated DLG Mapping Files

The attribute and geometric information within DLG data sets are encoded indirectly with major, minor, and linkage codes. The FME generates mapping files that can interpret all of these codes. The suggested method of creating custom mapping files for reading DLG data is to start with a generated mapping file. This provides an easy way to harness the enhanced semantic interpretation of all attribute codes and linkages built into the FME distribution. The following information pertains to the features and attributes produced by the mapping files generated to read DLG data.

Feature Representation

The DLG features produced by the generated mapping file consist of geometry and explicit attribute information. Each feature that has passed through all of the factories in the generated mapping file has its feature type set to one of the following: **HP**, **HY**, **SC**, **NV**, **BD**, **SM**, **RD**, **RR**, **MT**, **MS**, or **PL**. These features correspond to the category abbreviations as outlined in the DLG standards – see *DLG Categories*. The geometry of each feature is appropriate to the **dlg_type**: **dlg_point** features have a single coordinate pair, **dlg_line** features contain multiple coordinates, and **dlg_area** features define closed polygons with holes where appropriate.

DLG Categories

Name in Full	Abbreviation
Hypsography	HP
Hydrography	HY
Vegetative Surface Cover	SC
Non-Vegetative Features	NV
Boundaries	BD
Survey Control and Markers	SM
Roads and Trails	RD
Railroads	RR
Pipelines, Transmission Lines, and Miscellaneous Transportation Features	MT
Man-made Features	MS
U.S. Public Land Survey System	PL
Wetlands	WL
Unrecognized Category	UNKNOWN

All features share several attributes however, the feature will contain additional feature coding specific to the feature type. These are described in subsequent sections. All features tagged with major and minor codes of zero, indicating an outside area, are deleted.

DLG Attributes

The following table lists the different DLG attributes attached to every feature which has passed through the generated mapping file.

Field Name	Description
dlg_element_number	The element's internal identification number. The numbers are unique, positive, and sequential within each element type. Range: 1 - 32000
dlg_type	The DLG geometric type of this entity. Range: dlg_point dlg_line dlg_area Default: No default
dlg_code_list	A text string containing all Major and Minor codes assigned to this feature, in the fol-

Field Name	Description
	<p>lowing format:</p> <p>Range: <null> <code list> <code list> = (<major code>-<minor code> [,<major code>-<minor code>]*)</p> <p>For example, if the feature had major and minor code pairs of 180/201, 180/605, and 180/210, the string value of dlg_code_list attribute would be "(180-201,180-605,180-210)"</p>
category	The full length text string of the feature's category, as defined in the DLG standards. See DLG Categories.
description	<p>A text string containing all descriptive terms assigned to the feature through the Major and Minor codes. The source of these strings are the DLG standards documentation. Each description is separated by a semicolon.</p> <p>For example, if the feature had major and minor code pairs of 180/201, 180/605, and 180/210, the string value of description would be "Railroad; Underpassing; Arbitrary line extension [Code Deleted 07/95]"</p>
coincidentFeature	<p>If not null, this value indicates the other feature it is coincident with. The value refers to the coincident feature by its dlg_element_number.</p> <p>Range: 1 - 32000</p>

Hypsography

FEATURE_TYPE: HP

This category of data consists of information on topographic relief – primarily contour data – and supplementary spot elevations.

There is one attribute specific to Hypsography features.

Field Name	Description
elevation	<p>The elevation of the feature. The description attribute indicates whether the units are feet or metres.</p> <p>Range: -99999999.9 to +99999999.</p>

Hydrography

FEATURE_TYPE: HY

This category of data consists of all flowing water, standing water, and wetlands.

There are several attributes specific to Hydrography features.

Field Name	Description
elevation	The elevation of the feature. The description attribute indicates whether the units are feet or meters. Range: -99999999.9 - +99999999.9
rotationAngle	The angle of clockwise rotation of the feature.

Vegetative Surface Cover

FEATURE_TYPE: SC

This category of data consists of information about vegetative surface cover such as woods, scrub, orchards, and vineyards. Vegetative features associated with wetlands, such as marshes and swamps, are collected under Hydrography.

There are no attributes specific to Vegetative Surface Cover features.

Non-Vegetative Features

FEATURE_TYPE: NV

This category of data consists of information about the natural surface of the Earth as symbolized on the map such as lava, sand, and gravel features. This category is not all inclusive, as other non-vegetative surface features, such as glaciers, are found in the category of Hydrography.

There are no attributes specific to Non-Vegetative Features.

Boundaries

FEATURE_TYPE: BD

This category of data consists of:

- political boundaries that identify States, counties, cities, and other municipalities, and
- administrative boundaries that identify areas such as national and State forests.

Political and administrative boundaries are always collected as a single data set. There are several attributes specific to Boundaries features.

Field Name	Description
state	The full name of the American state or the state equivalent. Range: "ALABAMA" to "VIRGIN ISLANDS"
county	The full name of an American county or a county equivalent for all states. Range: "Abbeville" to "Ziebach"
township	The full name of an American civil township or a civil township equivalent for all states. Range: "Aasu" to "Zwolle"

Field Name	Description
population1990	The 1990 complete-count population of the American county or the county equivalent.
monument	The alphanumeric monument number of the feature.

Survey Control and Markers

FEATURE_TYPE: SM

This category of data consists of information about points of established horizontal position and third order or better elevations used as fixed references in positioning and correlating map features.

There are several attributes specific to Survey Control and Markers features.

Field Name	Description
elevation	The elevation of the feature. The description attribute indicates whether the units are feet or meters. Range: -999999999.9 to +999999999.9
state	The full name of the American state or the state equivalent. Range: "ALABAMA" to "VIRGIN ISLANDS"
county	The full name of an American county or a county equivalent for all states. Range: "Abbeville" to "Ziebach"

Roads and Trails

FEATURE_TYPE: RD

This category of data includes major transportation systems.

There are several attributes specific to Roads and Trails features.

Field Name	Description
numberOfLanes	The number of lanes the road or trail has.
routeNumber	The alphanumeric route number or the road or the trail.
routeType	This attribute indicates whether the route is an Interstate, U.S., State, County, Reservation, Park, or Military Route.

Railroads

FEATURE_TYPE: RR

This category of data includes major transportation systems.

There are several attributes specific to Railroads features.

Field Name	Description
numberOfTracks	The number of tracks the railroad has.
rotationAngle	The angle of clockwise rotation of the feature.

Pipelines, Transmission Lines, and Miscellaneous Transportation Features

FEATURE_TYPE: MT

This category of data includes major transportation systems.

There is one attribute specific to Pipelines, Transmission Lines, and Miscellaneous Transportation Features.

Field Name	Description
rotationAngle	The angle of clockwise rotation of the feature.

Man-made Features

FEATURE_TYPE: MS

This category of data includes cultural features not included in the other major data categories, such as buildings and other related industrial, commercial, and residential features.

There are several attributes specific to Man-made Features.

Field Name	Description
featureWidth	Width in mils of feature to scale.
rotationAngle	The angle of clockwise rotation of the feature.

Wetlands Features

FEATURE_TYPE: WL

This category of data is not found in DLG files produced by USGS. However, some agencies create DLG data of this type.

There are no attributes specific to Wetlands Features.

U.S. Public Land Survey System

FEATURE_TYPE: PL

This category of data describes the rectangular system of land surveys which is administered by the U.S. Bureau of Land Management. Public Land Survey System (PLSS) data exist only for areas falling solely or in part within the States which were formed from the public domain. The PLSS subdivides the public domain and represents property boundaries or references to property boundaries. These DLG data are not intended to be official or authoritative. They are presented as cartographic reference information. The only legal basis for determining land boundaries remains the original survey.

There are several attributes specific to U.S. Public Land Survey System features.

Field Name	Description
section	The alphanumeric Section Identifier number.

Field Name	Description
township	Township Identifier numbers north and south of baseline, including fractions. Examples: "101 South", "23 1/2 North"
range	Range Identifier numbers east and west of principal meridian including fractions, duplicate, and triplicate notification. Examples: "5 East", "79 1/2 West", "47 West, duplicate to north or east of the original township"
origin	Full text string identifying the origin of the survey, including township, state, and date. Examples: "Boise - PM ID 1867", "Ohio River - OH OH,IN 1785"
nonsectionID	Full text string of the Non-Section Identifier. Examples: "51", "W", "San Ignacio de la Canoa grant in Arizona", "Pueblo of Santa Ana grant in New Mexico"
monument	Land grant corner, location, or mineral monument number. Range: 0000 - 9999

Unknown Features

FEATURE_TYPE: UNKNOWN

This category is used to catch any DLG features that do not belong to one of the previous categories. This can happen if the original data was not produced by USGS. The major/minor codes associated with the feature are saved with it, and should be used in consultation with the producing agency to interpret the feature.

There are no attributes specific to this type of feature.

Dutch Top10 GML Reader/Writer

Format Notes: This format is not supported by FME Base Edition.

Overview

Top10 GML (also known as Top10NL) is a GML-based format from the Dutch National Mapping Agency *Kadaster*. The Top10 specification is still under development (although nearing completion), so FME's Reader/Writer is subject to change as the specification changes.

Of note is that this format supports heterogeneous aggregate geometries. This has some repercussions for manipulating Top10NL data within FME. See the Reader and Writer sections for more details.

Top10 Quick Facts

Format Type Identifier	TOP10
Reader/Writer	BOTH
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	Geometry
Typical File Extensions	.xml
Automated Translation Support	Yes
User-Defined Attributes	No. Attributes that are not part of the schema are currently ignored
Coordinate System Support	No
Generic Color Support	n/a
Spatial Index	n/a
Schema Required	No
Transaction Support	No
Geometry Type	xml_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	yes		z values	yes
none	yes			

Reader Overview

The Top10NL reader supports reading both geometries and attributes.

Reader Directives

The directive processed by the Top10NL reader are listed below. The suffix shown is prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the Top10NL reader is `TOP10`.

DATASET

Required/Optional: *Required*

The value for this keyword is the file name of the Top10NL file to be read. The normal extension for the files is `.xml`.

An example of the `DATASET` keyword in use is:

```
TOP10_DATASET /user/data/top10/Roads.xml
```

Workbench Parameter: *Source Dutch TOP10 File(s)*

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

Writer Directives

The directive processed by the Top10NL writer is listed below. The suffix shown is prefixed by the current `<WriterKeyword>` in a mapping file. By default, the `<WriterKeyword>` for the Top10NL writer is `TOP10`.

DATASET

Required/Optional: *Required*

The value for this keyword is the file name of the Top10NL file to be read. The normal extension for the files is `.xml`.

An example of the `DATASET` keyword in use is:

```
TOP10_DATASET /user/data/top10/MainRoads.xml
```

Workbench Parameter: *Destination Dutch TOP10 GML File*

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), all features also have the attribute `gml_geometry_property`. This attribute specifies what role the feature had in a heterogeneous geometric feature. See the section *Known Issues* for further information.

Attribute Name	Attribute Description
gml_geometry_type	This holds the value of the features geometry within the Top10NL format. The values thus range over the types of geometry values that feature allows. Example values are "geometrieVlak", "geometriePunt", although other values are possible as given in the Top10NL specification.

Known Issues

Top10NL is a GML format. For further information, see the section on the GML Reader/writer. There are some Top10NL-specific factors to be aware of:

Top10NL supports heterogeneous geometries, but not all other formats do. Therefore, when reading Top10NL data, FME will split up any features with heterogeneous geometries into features that agree in their values for all attributes, but which have non-heterogeneous geometries. This will result in some multiple features that differ only in their geometries, as the original features are "spread out" among a set of simpler features.

The Top10NL Writer will then combine those features that agree in their **gml_id** values back into a single feature with heterogeneous geometry. This technique of split-and-recombine is currently necessary, but does violate a common invariant that GML features each have a unique **gml_id** value. Special care is required when doing transformations on Top10NL data to maintain the relationship between features who agree in their **gml_id** values. If the FME features do not have a value for **gml_id**, a unique value will be assigned.

Since Top10NL allows several geometries per feature, it is not possible to determine automatically what the role of a particular geometry will be. It is thus necessary to specify what the role of a feature's geometry will be by setting the attribute *gml_geometry_attribute* to the appropriate value. For example, when writing a *Wegdeel* feature, you might set *gml_geometry_property* to 'geometrieVlak' or 'geometriePunt' depending on it's role. Much of this can be automated by the user with the use of a *GeometryFilter* transformer and *AttributeCreator* transformer in *WorkBench* (or relevant factories in a mapping file). For convenience, if reading from Top10NL, FME will set this property for you. Thus in most Top10NL-to-Top10NL transformations, no special handling will be required.

The Top10NL Writer requires that the features that it is given are valid Top10NL. For example, the Top10 GML specification requires that features of type *Gebouw* have an attribute called *hoogteklasse*. If the Top10NL Writer is given a *Gebouw* feature that does not have such an attribute, the resulting data will not be valid Top10NL (according to their schema document). In cases such as these, the Top10NL Writer will attempt to fail to write the data rather than allow non-valid Top10NL GML to be written. The onus is then on the user to ensure that the data is both syntactically and semantically valid Top10NL.

Users should consult the Top10NL Specification directly from the Dutch National Mapping Agency *Kadaster* website.

Encapsulated PostScript (EPS) Writer

Format Notes:

This format is not supported by FME Base Edition.

The Encapsulated PostScript® (EPS) Writer allows FME to write Encapsulated PostScript export files.

EPS is typically used for high-quality plots in desktop publishing software.

EPS Quick Facts

Format Type Identifier	EPS
Reader/Writer	Writer
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	Not used
Typical File Extensions	.eps
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	Yes
Spatial Index	Not applicable
Schema Required	Yes
Transaction Support	No
Geometry Type	eps_type
Encoding Support	No

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	point	yes
circles	yes	polygon	yes
circular arc	yes	raster	no
donut polygon	yes	solid	no
elliptical arc	yes	surface	no
ellipses	yes	text	yes
line	yes	z values	no
none	no		

Overview

EPS is a two-dimensional (2D) system with no provision for storing user-defined attributes for the geometric data.

All EPS information is contained within one page, beginning with a version header as well as a bounding box definition. EPS is based upon the PostScript format which provides methods for graphical drawing, simple programming control structures and the ability to create user-defined variables and functions.

All EPS data is contained in a single file with an **.eps** extension.

Filename Extension	Contents
.eps	All vector geometric data.

The EPS writer supports export of lines, polygons, arcs, ellipses (ellipse/circle), and text geometric data.

Some geometric entities may have display properties such as pen and brush width, type, pattern, and color. Color may be specified in red/green/blue (RGB) as well as cyan/magenta/yellow/black (CMYK).

Writer Overview

The EPS writer creates and writes feature data to an EPS file specified by the **DATASET** directive. The writer searches the mapping file for the `<writerKeyword>_DATASET` directive in the mapping file. This directive is required to be in the mapping file. An old EPS file in the directory with the same file name is overwritten with the new feature data. A typical mapping file fragment specifying the output EPS file looks like:

```
EPS_DATASET /usr/data/eps/myfile.eps
```

Writer Directives

The suffixes shown are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the EPS writer is **EPS**.

DATASET

Required/Optional: *Required*

The EPS writer processes the **DATASET** directive as described in *Writer Overview*. In addition, the writer scans the mapping file for `<writerKeyword>_RESOLUTION _X` and `<writerKeyword>_RESOLUTION _Y` directives. Both of these are optional directives and they define the bounding box of the EPS output file. The bounding box extends from the lower left corner of the page (defined as 0,0) and extends out to the values entered. By default, the X value is set to 612 and the Y value is set to 792. These values map onto an 8.5- by 11-inch piece of paper.

Workbench Parameter: *Destination Encapsulated PostScript (EPS) File*

FORCE_CMYK

Required/Optional: *Optional*

This directive specifies whether or not to force all output colors to be in CMYK format and defined as such in the EPS file. By setting the value following this keyword to **YES**, then all color usage output to the EPS file is done in CMYK. By default, this value is **NO**, meaning that a mix of RGB and CMYK color schemes may be in the output EPS file. However, despite forcing CMYK color output, some EPS viewers may not support the `setcmykcolor` call in their library. In these cases, the actual output of colors is done using a function we define in PostScript which interfaces exactly like the `setcmykcolor` call but uses `setrgbcolor` underneath. This will depend on the EPS viewer you are using.

Value: YES | NO

Default Value: NO

Workbench Parameter: *Force CMYK Colors*

LINE_JOIN_TYPE

Required/Optional: *Optional*

This directive specifies the default corner types to be drawn onto paths. The values specify the default shape to be put at corners of paths painted: 0 specifies a sharp corner, 1 specifies a rounded corner, and 2 specifies a butt-end corner.

Value: *0, 1, 2*

Default Value: *0*

Workbench Parameter: *Line Join Type*

LINE_WIDTH

Required/Optional: *Optional*

This directive specifies the default line width used to draw lines. This is measured in EPS units.

Value: *float >=0*

Default Value: *0.0* (the thinnest line that can be rendered at device resolution, i.e. 1 pixel wide)

Workbench Parameter: *Line Width*

MAINTAIN_ASPECT

Required/Optional: *Optional*

This directive specifies whether or not the source map dimensions will be kept or stretched to fit to the output bounding box. A **YES** indicates that the original map aspect will be maintained to fit within the destination-defined bounding box. This means that the entire destination bounding box defined may not be used. Alternatively, the value **NO** causes the original map to be stretched onto the destination bounding box defined.

Value: *YES | NO*

Default Value: *YES*

Workbench Parameter: *Maintain Map Aspect Ratio*

MAP_BUFFER

Required/Optional: *Optional*

This directive specifies the percentage of buffer room between the border of the output EPS map within the specified bounding box. It should be followed by a percentage value in decimals (for example, 0.20 is 20%). This value is used to buffer the border of the outputted EPS map within the specified bounding box. This prevents the border of the output map from being precisely on the bounding box border. The default value is 0.05 (5%), which places a 2.5% buffer between each map border and the bounding box border. This creates a total 5% buffer in the x and y axes.

Value: *0...1*

Default Value: *0.05*

Workbench Parameter: *Buffer Ratio*

RESOLUTION_X

This directive specifies the maximum EPS units (1 unit = 1/72 inch) for the x dimension of the output map.

Value: *Integer > 0*

Default Value: *612*

Workbench Parameter: *Width (points)*

RESOLUTION_Y

This directive specifies the maximum EPS units (1 unit = 1/72 inch) for the y dimension of the output map

Value: *Integer > 0*

Default Value: 792

Workbench Parameter: *Height (points)*

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

EPS features consist of geometry but no user-defined attributes, although there are special attributes to hold the type of the geometric entity and its display parameters. The feature type of features written to EPS is ignored.

All EPS features contain a **eps_type** attribute, which identifies the geometric type. Each element type also has a color associated with it. Depending on the geometric type, the feature contains additional attributes specific to the geometric type. These are described in subsequent sections.

Attribute Name	Contents
eps_type	The EPS geometric type of this entity. Range: eps_polyline eps_area eps_text eps_ellipse eps_arc Default: No default
eps_cmyk_color	This is a string that represents the color intensities of the element. It is formatted as cyan (C), magenta (M), yellow (Y) and black (K), This color attribute has highest priority. If present, it will be used in preference over eps_color and fme_color attributes. Range: String. (0..1, 0..1, 0..1, 0...1) Default: String (0,0,0, 1)
eps_color	This is a string that represents the color intensities of the element. It is formatted as red, green, blue intensities which range between 0..1 Note that if this attribute is not found, then fme_color will be used. Range: String. (0..1, 0..1, 0..1) Default: String (0,0,0)

Arc

eps_type: eps_arc

The arc definition here handles arcs, including those with different primary and secondary axis values. EPS arc features are linear features used to specify elliptical arcs. As such, the feature definition for **eps_arc** is similar to the ellipse definition with two additional angles to control the portion of the ellipse boundary drawn. EPS arcs also support rotation.

Tip: The function @Arc() can be used to convert an arc to a linestring. This is useful for storing Arcs in systems not supporting them directly.

In addition to the attributes below, arcs also make use of the pen attributes as defined for `eps_area` since arcs can also have fills.

Attribute Name	Contents
eps_primary_axis	The length of the semi-major axis in ground units. (x-axis) Range: Any real number > 0 Default: No default
eps_secondary_axis	The length of the semi-minor axis in ground units. (y-axis) Range: Any real number > 0 Default: No default
eps_start_angle	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of start_angle. Range: 0.0..360.0 Default: 0
eps_sweep_angle	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of sweep_angle. Range: 0.0..360.0 Default: No default
eps_rotation	The rotation of the major axis. The rotation is measured in degrees counter clockwise up from horizontal. Range: 360.0..360.0 Default: 0

Areas

eps_type: eps_area

EPS polygon features specify area (polygonal) features. The areas that make up a single feature may or may not be disjoint, and may contain polygons that have holes. Each area has a pen style associated with it to control the color, line weight, line type, and brush pattern used when it's drawn. If the area contains holes then when the fill pattern is applied, the holes enclosed by the area will **not** be filled. If no pen style is defined for a polygon entity, the previous style is used.

The following table lists the special FME attribute names used to control the EPS polygon settings.

Attribute Name	Contents
eps_line_width	Defines the line width used to draw the polyline. By default, the line is drawn one pixel wide. Range: Float >= 0 Default: 0.0
eps_dash_on	The number of pixels to be used as the on part of the dashed line used to draw the feature. If <code>eps_line_width</code> is specified, then this value is multiplied by the size of the pen to determine the number of pixels. If

Attribute Name	Contents
	<p>both <code>eps_dash_on</code> and <code>eps_dash_off</code> are 0, then a solid line is used.</p> <p>Range: Integer > 0</p> <p>Default: 0</p>
<code>eps_dash_off</code>	<p>The number of pixels to be used as the <code>off</code> part of the dashed line used to draw the feature. If <code>eps_line_width</code> is specified, then this value is multiplied by the size of the pen to determine the number of pixels. If both <code>eps_dash_on</code> and <code>eps_dash_off</code> are 0, then a solid line is used.</p> <p>Range: Integer > 0</p> <p>Default: 0</p>
<code>eps_line_join_type</code>	<p>Specify the type of corner that should be drawn onto this path.</p> <p>0 = sharp corners, 1 = rounded corners, 2 = butt-end corners</p> <p>Range: 0, 1, 2</p> <p>Default: 0</p> <p>Optional: Yes</p>
<code>eps_cmyk_fill_color</code>	<p>This is a string that represents the fill color intensities of the element. It is formatted as cyan (C), magenta (M), yellow (Y) and black (K), This color attribute has highest priority. If present, it will be used in preference over <code>eps_fill_color</code> and <code>fme_fill_color</code> attributes.</p> <p>Range: String. (0..1, 0..1, 0..1, 0..1)</p> <p>Default: String (0,0,0,1)</p>
<code>eps_fill_color</code>	<p>This is a string that represents the color intensities of the element. It is formatted as red, green, blue intensities which range between 0..1. If this attribute is not found, then the writer will refer to <code>fme_fill_color</code>.</p> <p>Range: String. (0..1, 0..1, 0..1)</p> <p>Default: None</p>

Ellipse

eps_type: `eps_ellipse`

The `eps_ellipse` features are point features, and have only a single coordinate. This point serves as the centre of the ellipse. Additional attributes specify the primary axis (X) and secondary axis (Y) of the ellipse. EPS ellipses also support rotation.

Tip: The primary ellipse axis is not necessarily the longest axis, but rather the one on the x axis.

From the EPS ellipse, we also can arrive at circles (since they are just ellipses with both primary and secondary axes being equal).

In addition to the attributes below, ellipses also make use of the brush and pen attributes as defined by `eps_area`.

Attribute Name	Contents
<code>eps_primary_axis</code>	The length of the semi-major axis in ground units. (x-axis) Range: Any real number > 0 Default: No default
<code>eps_secondary_axis</code>	The length of the semi-minor axis in ground units. (y-axis) Range: Any real number > 0 Default: No default
<code>eps_rotation</code>	The rotation of the major axis. The rotation is measured in degrees counterclockwise up from horizontal. Range: -360.0..360.0 Default: 0

Polylines

eps_type: `eps_polyline`

EPS polyline features specify linear features defined by a sequence of x and y coordinates. Polyline encapsulate the concept of a line since a line is just a sequence of two points. Each polyline has a pen style associated with it that specifies the color, line weight, and line type used when the line is drawn. If no pen type is defined for a polyline entity, if line attributes aren't found, then default parameters are used.

The table below lists the special FME attribute names used to control the EPS polyline settings.

Attribute Name	Contents
<code>eps_line_width</code>	Defines the line width used to draw the polyline. By default, the line is drawn one pixel wide. Range: Float >= 0 Default: 0.0
<code>eps_dash_on</code>	The number of pixels to be used as the on part of the dashed line used to draw the feature. If <code>eps_line_width</code> is specified, then this value is multiplied by the size of the pen to determine the number of pixels. If both <code>eps_dash_on</code> and <code>eps_dash_off</code> are 0, then a solid line is used. Range: Integer > 0 Default: 0
<code>eps_dash_off</code>	The number of pixels to be used as the off part of the dashed line used to draw the feature. If <code>eps_line_width</code> is specified, then this value is multiplied by the size of the pen to determine the number of pixels. If both <code>eps_dash_on</code> and <code>eps_dash_off</code> are 0, then a solid line is used. Range: Integer > 0 Default: 0

Attribute Name	Contents
eps_line_join_type	Specify the type of corner that should be drawn onto this path. 0 = sharp corners, 1 = rounded corners, 2 = butt-end corners Range: 0, 1, 2 Default: 0 Optional: Yes

Text

eps_type: eps_text

EPS text is used for text annotation in EPS. The coordinates specify the lower left coordinates of the text when it is placed. In addition, the size and angle in which the text is output can be specified.

The table below lists the special FME attribute names used to control the EPS text:

Attribute Name	Contents
eps_size	The size of the text specified in ground units Range: float > 0 Default: 0
eps_rotation	The text rotation is given in degrees and measured counter-clockwise up from the horizontal. Range: -360..360 Default: 0
eps_font	The PostScript name of the font. The fonts supported depend on the destination of the EPS file. Some typical fonts are Times, Helvetica and Courier. Range: String Default: Times
eps_style	The style of the font. This attribute must be matched with the current font since it's the combination of font and style that EPS recognizes. Some typical fonts and styles are Times-(Roman, Italic, Bold, BoldItalic), Helvetica-(Oblique, Bold, BoldOblique) and Courier-(Oblique, Bold, BoldOblique) Range: String Default: Roman
eps_text_string	The text to be displayed. Range: String Default: No default

Point

eps_type: eps_point

EPS point is used for points in EPS.

The table below lists the special FME attribute names used to control the EPS point:

Attribute Name	Contents
eps_size	<p>The size of the point specified in ground units</p> <p>Range: float > 0</p> <p>Default: 0</p>
eps_rotation	<p>The point rotation is given in degrees and measured counterclockwise up from the horizontal.</p> <p>Range: -360..360</p> <p>Default: 0</p>
eps_font	<p>The PostScript name of the font. The fonts supported depend on the destination of the EPS file. Some typical fonts are Times, Helvetica and Courier.</p> <p>Range: String</p> <p>Default: Times</p>
eps_style	<p>The style of the font. This attribute must be matched with the current font since it's the combination of font and style that EPS recognizes. Some typical fonts and styles are Times-(Roman, Italic, Bold, BoldItalic), Helvetica-(Oblique, Bold, BoldOblique) and Courier-(Oblique, Bold, BoldOblique)</p> <p>Range: String</p> <p>Default: Roman</p>
eps_symbol_string	<p>The text to be displayed.</p> <p>Range: String</p> <p>Default: "."</p>

ESRI ArcGIS Layer Reader

Format Notes:

To use FME's ESRI ArcGIS® Layer Reader, you must also install ESRI ArcGIS 9. It is not available with ArcGIS 8. The ArcGIS Layer reader modules allow FME to read ESRI Layer files and feature classes viewable in ESRI Arc-Catalog® and ArcMap®.

ArcGIS Layer Quick Facts

Format Type Identifier	ARCGIS_LAYER
Reader/Writer	Reader
Licensing Level	Professional
Dependencies	ESRI ArcGIS 9
Dataset Type	ESRI Layer/Feature Class
Feature Type	ESRI Layer/Feature Class
Typical File Extensions	ESRI Layer: .lyr ESRI Feature Class: none. The "path" to the feature class is used.
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Optional
Schema Required	N/A
Transaction Support	N/A
Enhanced Geometry	Yes
Geometry Type	geodb_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	yes		polygon	yes
circular arc	yes		raster	no
donut polygon	yes		solid	no
elliptical arc	yes		surface	no
ellipses	yes		text	yes

Geometry Support				
Geometry	Supported?		Geometry	Supported?
line	yes		z values	yes
none	yes			

Overview

Since the ArcGIS Layer Reader is based on the same technology as FME's Geodatabase Reader, any **spatial** feature that the Geodatabase Reader supports is also supported by the ArcGIS Layer Reader. The ArcGIS Layer reader does not read (non-spatial) tables.

With FME's ArcGIS Extension or the ESRI ArcGIS Data Interoperability Extension installed, it is possible to read FME feature classes for formats that ESRI does not natively support. An FME feature class is simply an FME feature type, that, where applicable, has been split geometry type (i.e. point, line, polygon, text, null).

The ArcGIS Layer module provides the following capabilities:

- **Fully Automatic Import:** The FME's ArcGIS Layer support provides fully automated import of data through the FME's Graphical User Interface (GUI). This is ideal for quick data imports.
- **Mapping File/Workspace Customization:** The FME's ability to generate mapping files/workspaces for user customization allows greater and more precise control over ArcGIS Layer translations.
- **Enhanced Geometry Model Support:** This reader supports the enhanced geometry model. The addition of enhanced geometry model support allows lines and polygons containing arcs to be maintained, rather than stroked or the geometry split up into multiple segments.

Reader Overview

Reader Directives

The suffixes listed are prefixed by the current **<ReaderKeyword>** in a mapping file. Unless otherwise specified, the **<ReaderKeyword>** for the ArcGIS Layer reader is the same as the **<ReaderType>**.

DATASET

Required/Optional: *Required*

A single ESRI layer/feature class from which data is to be read. The value for an ESRI layer is simply the layer file. The value for a feature class is the ESRI path to the file/database containing the feature class plus the name of the feature class. The ESRI path is displayed in the Location toolbar within ESRI ArcCatalog. The FME Dataset picker has been enhanced for the ArcGIS Layer reader to use ESRI's layer/feature class picker. Using this picker will greatly simplify the process of selecting a dataset.

ESRI Layer Example:

```
ARCGIS_LAYER_DATASET "C:\data\runways.lyr"
```

Feature Class within a Personal Geodatabase Example:

```
ARCGIS_LAYER_DATASET "C:\data\airport.mdb\runways"
```

Feature Class within an Enterprise Geodatabase Example:

```
ARCGIS_LAYER_DATASET "Database Connections\esri92.sde\JOE.runways"
```

FME Feature Class within a MIF/MID file Example (requires ArcGIS to be extended by FME):

```
ARCGIS_LAYER_DATASET "C:\data\mif\usa.mif\usa Line"
```

Workspace Parameter: *ESRI ArcGIS Layer*

DEF

Required/Optional: *Optional*

Describes feature classes. Normally these lines are automatically generated within a mapping file/workspace using FME.

Example:

```
ARCGIS_LAYER_DEF Parcels
  geodb_type          geodb_polygon
  OBJECTID_1         integer
  PROPERTY_I         double
  LANDUSE_CO         char(3)
  ZONING             char(6)
  PARCEL_ID         integer
  Res               smallint
  ZONING_S           char(4)
  Shape_Length      double
  Shape_Area        double
```



SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see [About Feature Attributes](#)), this format adds the format-specific attributes described in this section.

The ArcGIS Layer modules make use of the following special attribute names.

Attribute Name	Contents
geodb_type	The type of geometric entity stored within the feature. The valid values are listed below: geodb_point geodb_multipoint geodb_polyline

Attribute Name	Contents
	<p>geodb_arc geodb_ellipse geodb_polygon geodb_annotation geodb_dimension geodb_simple_junction geodb_simple_edge geodb_complex_junction geodb_complex_edge</p> <p>For a description of the attributes belonging to each of the different geodb_type's, please see the chapter <i>ESRI Geodatabase Reader/Writer</i>.</p>
geodb_feature_is_simple	Indicates whether or not the geometry is simple.
geodb_measures Available only with classic geometry.	This is present for features that have measures. This is a comma-separated list of floating values that correspond to the vertex measures. The first value is for the first vertex, second for the second, and so on.

Features read from a dataset also have an attribute for each attribute in the layer/feature class.

ESRI ArcGIS Map (.mxd) Reader

Format Notes: To use FME's ESRI ArcGIS Map Reader, you must also install ESRI® ArcGIS® 9. It is not available with ArcGIS 8.

The ESRI ArcGIS Map reader enables FME to retrieve data from ESRI's ArcMap Document.

ArcGIS Map Quick Facts

Format Type Identifier	ARCGISMAP
Reader/Writer	Reader
Licensing Level	ESRI Edition
Dependencies	ESRI ArcGIS 9
Dataset Type	File
Feature Type	Layer name (Table name for tables)
Typical File Extensions	.mxd
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	Yes
Spatial Index	Always
Schema Required	No
Transaction Support	N/A
Enhanced Geometry	Yes
Geometry Type	geodb_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	yes		polygon	yes
circular arc	yes		raster	no
donut polygon	yes		solid	no
elliptical arc	yes		surface	no
ellipses	yes		text	yes
line	yes		z values	yes
none	yes			

Overview

Since the ArcGISMap Reader is based on the same technology as FME's Geodatabase Reader, **any** feature that the Geodatabase Reader supports is also supported by the ArcGISMap Reader.

With FME's ArcGIS Extension or the ESRI ArcGIS Data Interoperability Extension installed, it is possible to create an ArcMap document containing data from formats that are not natively supported by ESRI. This ArcMap document can then be read using the ArcGISMap Reader.

The ArcGISMap module provides the following capabilities:

- **Fully Automatic Import:** FME's Geodatabase support provides fully automated import of data through the FME's Graphical User Interface (GUI). This is ideal for quick data imports.
- **Mapping File/Workspace Customization:** FME's ability to generate mapping files/workspaces for user customization allows greater and more precise control over Geodatabase translations.
- **Enhanced Geometry Model Support:** This reader supports the enhanced geometry model. The addition of enhanced geometry model support allows lines and polygons containing arcs to be maintained, rather than stroked or the geometry split up into multiple segments.

Reader Overview

Reader Directives

The suffixes listed are prefixed by the current `<ReaderKeyword>` in a mapping file. Unless otherwise specified, the `<ReaderKeyword>` for the ArcGISMap reader is the same as the `<ReaderType>`.

DATASET

Required/Optional: *Required*

The file from which data is to be read.

Workbench Parameter: *Source ESRI ArcGIS Map File(s)*

IDS

Required/Optional: *Optional*

Specifies the layers/tables from which features are to be retrieved. This directive is used in conjunction with the `DEF` keyword. If both `DEF` and `IDS` are specified, then the intersection is taken from both of these directives. The layers that are read are subject to use of the `READ_INVISIBLE_LAYERS` keyword. If the layer name is blank for a particular layer, then the feature class name must be used instead.

Workbench Parameter: *Feature Types to Read*

DEF

Required/Optional: *Optional*

Describes layers/tables. Normally these lines are automatically generated within a mapping file using FME. This directive is used in conjunction with the `IDS` directive. If both `DEF` and `IDS` are specified, then the intersection is taken from both of these directives. The layers that are read are subject to use of the `READ_INVISIBLE_LAYERS` directive. If the layer name is blank for a particular layer, then the feature class name must be used instead.

IGNORE_MAP_EXTENTS

Required/Optional: *Optional*

Specifies whether to read only those features that are within the extents of the ArcMap document, or to ignore the extents and read all the features in the layer. (This directive does not affect the reading of features from tables.)

Value: YES | NO

Default Value: *NO*

Workbench Parameter: *Ignore Map Extents*

READ_INVISIBLE_LAYERS

Required/Optional: *Optional*

Specifies whether to read features from an invisible layer. (This directive does not affect the reading of features from tables.)

Value: *YES | NO*

Default Value: *NO*

Workbench Parameter: *Read Invisible Layers*

USE_SELECTION_SET

Required/Optional: *Optional*

Specifies whether or not to only read the selected features. It is used in conjunction with the directives **READ_INVISIBLE_LAYERS** and **IGNORE_MAP_EXTENTS** since it is possible that some of the selected features are currently invisible and/or outside the current extents of the map. If set to YES and there are no features in the selection set, all the features from the specified layers will be read. When the layer name of a layer is blank, the feature type is set to the feature class name of the layer, rather than using the blank layer name. When using the **IDS** directive, the name of the feature class should be specified when setting up to read from the layer with the blank layer name. (This directive does not affect the reading of features from tables.)

Value: *YES | NO*

Default Value: *NO*

Workbench Parameter: *Use Selection Set*

RESOLVE_DOMAINS

Required/Optional: *Optional*

This directive specifies whether to resolve attributes that have a default coded value domain (i.e., the domain was not set up through a subtype) associated with them. This means that when an attribute of a feature has a coded value domain associated with it, another attribute will also be added that represents the textual description of the coded attribute. The new attribute will be **<attribute-name>_resolved**, where **<attribute-name>** is the name of the attribute containing the code. This attribute will only be added when **<attribute-name>** contains a non-NULL value.

Value: *YES | NO*

Default Value: *NO*

Workbench Parameter: *Resolve Domains*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

The ArcGISMap modules make use of the following special attribute names.

Attribute Name	Contents
geodb_type	<p>The type of geometric entity stored within the feature. The valid values are listed below:</p> <ul style="list-style-type: none"> geodb_table geodb_point geodb_multipoint geodb_polyline geodb_arc geodb_ellipse geodb_polygon geodb_annotation geodb_dimension geodb_simple_junction geodb_simple_edge geodb_complex_junction geodb_complex_edge <p>For a description of the attributes belonging to each of the different geodb_type's, please see the chapter <i>ESRI Geodatabase Reader/Writer</i> .</p>
<attribute-name>_resolved	<p>When reading, if RESOLVE_DOMAINS is set to YES, then the description corresponding to the domain code is stored in this attribute.</p>
fme_color	<p>A normalized RGB triplet representing the fill color of the feature, with the format r,g,b. Currently, both 'unique value' and 'color ramped' symbologies are supported for determining the fill color.</p> <p>Range: 0,0,0 to 1,1,1 Default: No default</p>
geodb_feature_is_simple	<p>Indicates whether or not the geometry is simple.</p>
geodb_measures	<p>This is present for features that have measures. This is a comma-separated list of floating values that correspond to the vertex measures. The first value is for the first vertex, second for the second, and so on.</p> <p>Available only with classic geometry.</p>

Features read from an ArcMap document also have an attribute for each attribute in a layer.

ESRI ArcInfo Coverage/ ESRI ArcInfo Export (E00) Reader/Writer

Format Notes:

This format is available as a Reader only with FME Base Edition.

Coverage reading and writing is available only with FME ESRI Edition, FME Smallworld Edition and FME Oracle Edition.

The ESRI® ArcInfo® Coverage and Export (E00) Reader/Writer enables FME to read and write binary coverages, E00 files and Info tables. Full support for compressed E00 files – export option 1 or 2 – is also provided. Safe Software’s *FME ESRI Edition* also reads and writes binary ArcInfo coverages directly.

E00 Quick Facts

Format Type Identifier	E00 or ARCINFO
Reader/Writer	Both
Licensing Level	<ul style="list-style-type: none">• Reader: Base• Reader + Writer: FME ESRI Edition, FME Smallworld Edition and FME Oracle Edition
Dependencies	None
Dataset Type	<ul style="list-style-type: none">• Directory for ArcInfo Coverage reader• Directory for ArcInfo Table reader• File for E00 reader• Directory for both writers
Feature Type	<ul style="list-style-type: none">• Subdirectory base name for ArcInfo• File base name for E00
Typical File Extensions	.e00 (.e01, .e02, ...)
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	e00_type
Encoding Support	Yes

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	yes
line	yes		z values	n/a
none	yes			

Overview

A single E00 file describes a complete ArcInfo coverage. The file itself is actually an archive of several smaller files, referred to here as *subfiles*. Some of these subfiles have fixed names which do not vary from coverage to coverage, and follow a predefined data format. These are referred to as the *standard subfiles*.

The remainder of the subfiles contained within an E00 are the *info files*. These files may contain user-defined attributes, and have names which vary from coverage to coverage. The ways in which the names vary are discussed in Info Files.

Reader Overview

The E00/ArcInfo reader produces FME features for all feature data contained in a single E00 file, binary ArcInfo coverage or ArcInfo table. In order to process multiple E00 files, coverages, you must invoke the FME for each E00 file, or use the Multi-Reader, described in the *Multi-Reader* chapter.

Large E00 files are often split into smaller files, named `<filename>.e00`, `<filename>.e01`, `<filename>.e02`, etc. The E00 reader automatically detects this and reads the set of files as if they were a single E00 file.

To read ArcInfo coverages, specify the directory that contains the coverage to the E00 reader.

To read just the ArcInfo tables, specify the "info" directory that contains the info tables to the ArcInfo reader. All info tables will be processed as data with no spatial information attached to it even though they may be part of the coverage.

Reader Directives

The suffixes listed are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the E00 writer is `E00` or `ARCINFO`.

DATASET

Required/Optional: *Required*

The E00 reader processes the directive `<ReaderKeyword>_DATASET`, where `<ReaderKeyword>` is the keyword assigned to the E00 reader. By default, the reader keyword is `E00` for E00 files, or `ARCINFO` when working with binary ArcInfo coverages.

The `<ReaderKeyword>_DATASET` directive specifies the data set to be read by the E00 reader. When reading E00 files, this is the name of the E00 file, including the `.e00` suffix. If the E00 reader encounters the end of the E00 file `<filename>.e00` while it is expecting more data to process, it will attempt to use the file `<filename>.e01` as a continuation of the input. No specific mention of `<filename>.e01` is required.

When reading binary ArcInfo coverages directly, the value of `<Readerkeyword>_DATASET` is the path of the directory containing the files that make up the coverage. (Note that it is not the directory containing the individual coverage directories, but rather one of the coverage directories itself.) No additional configuration is required to tell the reader to process a binary coverage instead of an E00 file. If the supplied argument is a directory, the reader will assume the data set is a binary coverage.

When reading binary ArcInfo tables directly, the value of `<Readerkeyword>_DATASET` is the path to the "info" directory which contains info tables.

Workbench Parameter: *Source ESRI ArcInfo Export (E00) File(s)*

TEXT_CURVE

Required/Optional: *Optional*

There is an additional directive that tells the E00 reader how to deal with text elements which follow a splined curve in ArcInfo. In the past, the FME has simply drawn a straight line from the first point of the curve to the last point, and placed the text along that line. The default behavior now is to space the characters along the original curve, and generate a separate character for each (non-whitespace) character of the text. The directive `<ReaderKeyword>_TEXT_CURVE` allows one to change the FME's interpretation of curved text features. The default value for the directive is **FOLLOW**; a value of **IGNORE** will revert the FME to its traditional behavior; and a value of **FIT** will tell FME to evenly space out the characters of the text along the curve, so that the left edge of the first character is on the first point of the curve, and the right edge of the last character is on the last point.

Value: *FIT | FOLLOW | IGNORE*

Default Value: *FOLLOW*

Workbench Parameter: *Text Curves*

SINGLE_BYTE_TEXT

Required/Optional: *Optional*

If the E00 reader is placing the text characters along their curve (i.e., if the `<ReaderKeyword>_TEXT_CURVE` was not given a value of **FIT** or **FOLLOW**), it normally inspects the text content to try to detect whether any are multi-byte representations of "international" characters. If it finds a pair of bytes that it thinks define a single character, it will use those two bytes in a single feature representing that character of text. This behavior might lead to incorrect representation of some character strings, if they happen to be composed of single-byte characters that look like multi-byte characters. This automatic detection of multi-byte characters may be disabled by providing the `SINGLE_BYTE_TEXT` directive a value of **YES**.

Value: *YES | NO*

Default Value: *NO*

Workbench Parameter: *Force Single-Byte Text*

INCLUDE_BND

Required/Optional: *Optional*

ArcInfo coverages typically include an info file named BND, which defines the extents of the coverage. FME will normally ignore the contents of this file. If the `<ReaderKeyword>_INCLUDE_BND` keyword is specified with a value of **YES**, FME will create a single feature representing the coverage extent information.

The extent is defined on the resulting feature with the attributes XMIN, YMIN, XMAX, and YMAX. In FME, the feature will have a polygon geometry corresponding to these attributes' values; in FME Objects, the BND feature will have no geometry.

Value: *YES | NO*

Default Value: *NO*

Example:

`E00_INCLUDE_BND Yes`

INCLUDE_TIC

Required/Optional: *Optional*

ArcInfo coverages typically include an info file named TIC, which defines the tic points for the coverage. FME will normally ignore the contents of this file. If the <ReaderKeyword>_INCLUDE_TIC is specified with a value of YES, FME will create a feature for each TIC point.

The features resulting from reading the TIC file will have the IDTIC, XTIC, and YTIC attributes as defined in the TIC file, and will have a point geometry corresponding to (XTIC,YTIC).

Value: YES | NO

Default Value: NO

Example:

```
E00_INCLUDE_TIC Yes
```

HYPHENS_ARE_VALID

Required/Optional: *Optional*

When set to "Yes" the reader will not convert hyphens in attribute names to underscores. The one exception to this rule is that if the attribute name ends in "-ID" – in this case, it will be converted to _ID no matter what the directive is set to.

So, for example, if the attribute name is "HYPH-ENS-ID", then when the directive is set to "Yes", it will be read as "HYPH-ENS_ID" and if the directive is set to "No", it will be read as "HYPH_ENS_ID".

If this directive is missing, then it will have an implied value of "No". This is to ensure backwards compatibility, so workspaces created with a previous version of FME (one that does not yet support this directive) continue to work as they always have.

Value: YES | NO

Default Value: Yes

Example:

```
E00_HYPHENS_ARE_VALID Yes
```

GENERATE_NODE_FEATURES

Required/Optional: *Optional*

Traditionally the ArcInfo reader reads the NAT table and outputs the NODE attributes as a plain set of attributes. If the value is YES, the endpoints of the ARC features are turned into NODE features, which are then joined with the NAT table attributes to provide fully formed point features.

Value: YES | NO

Default Value: NO

Example:

```
E00_GENERATE_NODE_FEATURES NO
```

Workbench Parameter: *N/A (settings box only)*

There are no other directives processed by the E00 reader, meaning there are no DEF lines for reading E00 features. The features obtained from the specified data set take the form described in the remainder of this chapter.

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

*** Workbench Parameter**

Additional Attributes to Expose

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

*** Workbench Parameter**

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>

*** Workbench Parameter**

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

Writer Overview

The ArcInfo writer provides the ability to generate, in a single invocation of the FME, several E00 files or binary coverage directories. All E00 files or coverages created in a single run of the FME will be placed into a single directory.

Unlike the reader, the writer requires **DEF** lines to define the attributes of the output E00 files or binary coverage directories. The writer provides a mechanism to apply a single set of attributes to all geometric features placed into a file, as well as allowing each specific info file to have a unique set of attributes.

The E00 writer can create compressed or uncompressed E00 files, or, with *FME ESRI Edition*, binary ArcInfo coverages.

Writer Directives

The suffixes shown are prefixed by the current **<WriterKeyword>** in a mapping file. By default, the **<WriterKeyword>** for the E00 writer is **E00** or **ARCINFO**.

DATASET

Required/Optional: *Required*

Unlike the E00 reader, which reads a single E00 file, the writer may create several files. The DATASET keyword for the E00 writer is the directory into which the created E00 files (or coverages) will be written.

Workbench Parameter: *Destination ESRI ArcInfo Export (E00) File*

DEF

Required/Optional: *Required*

Workbench Parameter: *<WorkbenchParameter>*

An E00 file or coverage must be completely defined before it may be written. The definition specifies the base name of the file and the names and types of all attributes on all info files within the E00 file. The syntax of an E00 DEF line is:

```
<ReaderKeyword>_DEF <baseName> \
    [E00_FORCE_OUTPUT <subfileName>[,<subfileName>]+] \
    [<attrName> <attrType>[,INDEX]]+
```

The file name of the physical E00 file is constructed by appending the .e00 suffix to its baseName. Binary coverages are written as a directory named baseName. Coverage names are always truncated to contain thirteen or fewer characters, and converted to lowercase letters, when forming output directory names.

The E00_FORCE_OUTPUT keyword is used to force the named subfiles to appear in the output E00 file or binary coverage, even if no contents are specified for them. The value for subfileName can either be the name of a one of the standard subfiles LAB or TOL, or can be any info file. (The naming of info files follows the same pattern as in the

<infoFile>:<attrName> notation discussed below.) The TOL and LAB files are normally generated by the E00 writer, but this behavior can be overridden by specifying an E00_FORCE_OUTPUT keyword that does not list the TOL or LAB subfile: if a list of files are forced to be output, and no TOL file is specified in the list, then a TOL file will be generated only if specific tolerances are defined as discussed in the section titled *Tolerances*; if the LAB subfile is not specified in the list, a LAB file will be generated only if features are written out to define the contents of the LAB file.

The attribute names for an E00 file must be uppercase, and must not exceed 16 characters in length. One should be aware that there are a few peculiar character sequences used by the E00 writer, for historical purposes. A trailing underscore character on an attribute name is replaced by a hash character (“#”), and a trailing sequence of “_ID” is replaced with “-ID”. Thus, attributes named JOES_ and JOES_ID on the E00_DEF line of a mapping file would be called JOES# and JOES-ID in the resulting info file.

One can work around this limitation by inserting single quotes around the underscore in the name of the attribute on the DEF line. For example, an attribute listed as JOES'_ID on an E00_DEF line would result in an attribute named JOES_ID appearing in the info file, instead of an attribute named JOES-ID.

The following table shows the attribute types that are supported.

Attribute Type	Description
char(<width>)	Fixed-length character string. The width parameter controls the maximum number of characters that can be stored in the field. When a character field is written, it is right-padded with blanks, or truncated, to fit the width. width must be between 1 and 320, inclusive.
char	Character string with a default maximum length (currently set to 320). This type should be used only for testing purposes, and not for production mapping files; for most cases, use the char(<width>) form above.
date	Character string representing a date. Attributes of type date must have exactly eight characters, and be of the form YYYYMMDD , where YYYY is the year, MM is the month (01-12), and DD is the day of the month (01-31).
int int(<width>)	Integer field. The optional width parameter specifies the display width of the field within ArcInfo. width must be between 1 and 16, inclusive. Representable numbers are those in the range of -999,999,999,999 to 9,999,999,999,998, inclusive.
number (<width>,<dec>)	Numeric data displayed with a field width of width and dec decimal positions. The value of width must allow for any minus sign and decimal point in the number, and must be in the range of 1 to 16, inclusive. The value of dec must be between 0 and 14, inclusive.
binint binint(<size>) binint(<size>,<width>)	Integer value, to be stored in ArcInfo as a binary number instead of character data. If the optional size parameter is specified, it specifies the number of bytes of storage (2 or 4 bytes) ArcInfo will use to store the

Attribute Type	Description
	<p>value. The optional width parameter specifies the display width for ArcInfo to use. The size will default to 4bytes, and the display width will default to 5 for a 4-byte integer, or 4 for a 2-byte integer.</p> <p>size must be either 2 or four. width may be any integer between 1 and 6 when size is 2, or between 1 and 11 if size is 4.</p>
<p>float float(<width>,<prec>)</p>	<p>Floating point number, to be stored in ArcInfo as a four-byte binary number instead of as character data. The width and prec parameters define the display width and number of decimals for ArcInfo to use. A float field retains up to 9 digits of precision; only zero and numbers with a magnitude between 1.175494351e-38 and 3.402823466e+38 may be represented as float values.</p>
<p>double double(<width>,<prec>)</p>	<p>Floating point number, to be stored in ArcInfo as an eight-byte binary number instead of as character data. The width and prec parameters define the display width and number of decimals for ArcInfo to use. A double field retains up to 17 digits of precision; only zero and numbers with a magnitude between 2.225073858507201e-308 and 1.7975931348623158e+308 may be represented as double values.</p>
<p>redefined(<offset>, <length>, <fieldName>)</p>	<p>A redefined field specifies a subfield of another field within the same info file. The features written to an info file with redefined fields do not actually have attributes named for the redefined fields; the resulting E00 file defines the field in such a way that ArcInfo interprets the value of the redefined field as byte positions offset to (offset + length - 1) of the specified field-Name. Offsets are zero-relative, so an offset value of 1 actually refers to the second character of the named field.</p>

An attribute's type parameter may optionally be followed by the literal string **,INDEX**, such as in:

```
E00_DEF ROADS \
      NAME char(16) \
      .ARC:IDENT binint,INDEX
```

This indicates that particular field is an index in the ArcInfo info file.

Normally, any attributes provided on the DEF line will be applied to all info files created with the exception of the .BND and .TIC files, which have a specific format. However, the DEF line supports an attribute naming convention which allows a particular attribute to be applied to a specific info file. If the attribute name in the DEF line is of the form

<infoFile>:<attrName>, then the attribute **attrName** will be added only to the info file specified by infoFile. There are actually three ways to specify an infoFile, as outlined by the following table.

infoFile Specification	Application
<baseName>.<suffix>	Attribute belongs to the precise info file named, and no others, for example, HYDRO.TATANNO.
.<suffix>	Attribute belongs to all info files with the specified suffix, for example, .TATANNO, .PAT.
.TAT	Attribute applies to all info files with a suffix that starts with .TAT. This is a special case provided solely for text attributes. Since text attribute files have the suffix .TAT<annoLayer>, with the possibility of having several annoLayers in a single E00 file, this syntax allows the definition of attributes to be applied to the text attribute files for all annotation layers.

If an info file has specific attributes defined on it using the above <infoFile>:<attrName> syntax, then it will not have any of the attributes listed with the normal syntax, that is, <attrName> only.

Furthermore, the special info files .TIC and .BND always have the same predefined attributes, no matter what the contents of the DEF line for the E00 file are. Each of the special geometry-related attribute files (.AAT, .PAT, and .TAT) also has a default set of attributes which will always be present in the info file, but in these cases, the set of attributes will be supplemented with the appropriate attributes specified on the DEF line.

If an attribute name starts with a "-" character, then the specified attribute is removed. For example, if an attribute named ".TAT:-OFFSETX" is specified on the **DEF** line, then the specified attribute is not included in the resulting info file. This allows you to remove default attributes from standard info files.

If an attribute name starts with a "+" character, then its type will override the type of any of the standard attributes with the same name.

Controlling E00 Output describes what info files will be automatically created when geometric entities are directed at E00 files, and how to generate custom info files.

TOLERANCES

Required/Optional: *Optional*

An E00 file or coverage can contain a subfile which defines tolerances used within ArcInfo. There are exactly ten tolerances defined in this file. Each tolerance has a value and a state. The FME refers to these tolerances by name or by number, as described in the **Tolerance Values**.

The TOLERANCES keyword tells the FME to create a TOL subfile with a specific value or state for a particular tolerance value. The syntax of a tolerance specification is:

```
<writerKeyword>_TOLERANCES <id>=<val>[,<state>][:<id>=<val>[,<state>]]+
```

where id is a valid tolerance number or name from the table in **Tolerance Values**, val is the specified tolerance's new value, and the optional state parameter is either 1 or 2, to specify whether the tolerance has been verified. If the state is not specified, a default value of 1 is used.

The E00 writer always generates a TOL file in each generated E00 file or binary coverage unless the E00_FORCE_OUTPUT option is specified in the coverage's DEF line, and TOL does not appear in the list of subfiles to force. If a TOL file is generated, any tolerances not specified by the TOLERANCES keyword will take the following default values and states:

Tolerance Identifier	Default Value	Default State
1 (FUZZY)	1.0e-20	1
2 (GENERALIZE)	0	2
3 (NODE_MATCH)	0	2
4 (DANGLE)	0	1
5 (TIC_MATCH)	0	2
6 (EDIT)	1.0e-18	2
7 (NODESNAP)	1.0e-19	2
8 (WEED)	1.0e-19	2
9 (GRAIN)	1.0e-19	2
10 (SNAP)	1.0e-19	2

PRECISION

Required/Optional: *Optional*

All subfiles in a given E00 file or coverage are written out with either single-precision numbers or double-precision numbers. The PRECISION keyword takes a value of single or double, and specifies the precision used for all subfiles of all E00 files written.

Value: *SINGLE* | *DOUBLE*

Default Value: *DOUBLE*

Workbench Parameter: *Coverage Precision*

COMPRESSION

Required/Optional: *Optional*

E00 files may be written out uncompressed or they may have one or two levels of compression applied to them. Compressed files take up far less space than uncompressed files but they are impossible to inspect manually and are not readable by many systems.

The **COMPRESSION** directive allows you to specify how much compression to apply to a file.

Note: *FME ESRI Edition* will also generate binary ArcInfo coverages directly, if the COMPRESSION keyword is given a value of BINARY.

Value: *NONE* | *PARTIAL* | *FULL*

Default Value: *NONE*

Workbench Parameter: *Compression*

MAX_OUTPUT_SIZE

Required/Optional: *Optional*

When writing out to an E00 file, the user might want to break the file into an *.e00*, *.e01*, *.e02*, etc., based on the size of the file being written. This may be done by specifying **MAX_OUTPUT_SIZE** directive. The argument is the maximum size of each file produced.

The argument is specified as a number, optionally suffixed with a lower case "b" (bytes), "k" (kilobytes), or "m" (megabytes). If there is no such suffix, "bytes" will be assumed. For example:

E00_MAX_OUTPUT_SIZE 1024k

will produce no file greater than a megabyte.

When expressed in kilobytes or megabytes, the size is measured using base-2 measurements, hence a kilobyte 1024 bytes, and a megabyte is 1024 kilobytes.

The default behaviour is not to limit the size of output E00 files.

Workbench Parameter: *Maximum Output File Size*

BYPASS_LINEAR_TOPOLOGY

Required/Optional: *Optional*

Normally, the FME will compute a line-node topology whenever it writes out an E00 file or binary coverage. This can be a very expensive operation, and is not always needed.

For instances where a fully built linear topology is not needed in the resulting coverage, one may use the BYPASS_LINEAR_TOPOLOGY keyword to disable this feature. A value of YES will disable the topology computation, and a value of NO will leave it enabled.

For example,

E00_BYPASS_LINEAR_TOPOLOGY Yes

will produce output files with no linear topology.

Note that this affects linear output only. It is not possible to create a coverage of polygons which do not have topology connecting them to their linear boundaries. (Such representation is not possible in E00 files.).

Value: YES | NO

Default Value: NO (FME will compute a linear topology)

Workbench Parameter: *Linear Topology*

PRESERVE_CASE

Required/Optional: *Optional*

When set to "Yes", the output files will have the same case as the feature types specified on the DEF line. When set to "No" it will demote the feature type to lowercase. For example, if the feature type is **POINT**, it will be output as **POINT** if the directive is set to "Yes" and as **point** if the directive is set to "No".

If this directive is missing, then it will have an implied value of "No". This is to ensure backwards compatibility, so workspaces created with a previous version of FME (one that does not yet support this directive) continue to work as they always have.

Example:

E00_PRESERVE_CASE Yes

Value: YES | NO

Default Value: Yes

Feature Representation

This section discusses the way geometry and attributes are defined on features which represent the records in the various files within an E00 file. There is quite a difference between the features that the E00 reader emits, and those formed from the generic, automatically-generated mapping file. This discussion focuses primarily on the raw output from the E00 reader. **Generated Mapping Files** provides a description of the feature that the generated mapping file creates.

There is also a difference between the features that the reader emits and those which the writer expects to be given to write out. Most notably, the reader uses the name of the subfile as the feature type of each FME feature read from

an E00 file, whereas the writer uses the FME feature type to determine the name of the E00 file to which the feature will be written. In addition, the reader defines certain attributes on features read from E00 files – such as, E00_FEAT_ROLE and E00_RECORD_NUM – which are unused by the E00 writer.

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Geometry Composition

There are essentially four types of geometry defined in E00 files:

- arcs (lines)
- points
- polygons
- text

The geometries are formed by forming relations between certain standard subfiles and certain info files. The reader itself does not form these relations, but provides the attributes on the features allowing the mapping file to form the relations. Mapping files which have been automatically generated to read from E00 files form the necessary joins between the subfiles and the info files, and are a good starting point when creating custom mapping files to read E00 data. A description of the features output from automatically generated mapping files is given in the subsection titled *Generated Mapping Files*.

The following table summarizes the geometry types and lists the additional attributes required to fully define the geometry, as is the case for text features.

Geometry Type	Description	Additional Attributes Required
e00_arc	A string of geographic points that does not join or cross with itself. Features read from the ARC subfile contain arcs as their geometry.	None
e00_poly	A solid area, with an outer boundary and zero or more holes. No features is given a polygon geometry by the reader. They must be formed by factories in the mapping file.	None
e00_point	A simple (x,y) coordinate. The E00 reader creates points for features read from CNT and LAB standard subfiles.	None
e00_text	Defines annotation text at a particular location, with a rotation measured in degrees counterclockwise from the horizontal and text height measured in ground units. The geometry portion of a text feature is the single (x,y) point that defines its position.	e00_text_string e00_rotation e00_text_height
e00_no_geom	Feature has no associated geometry.	None

Feature Types

The E00 reader and the E00 writer view feature types somewhat differently. The features emitted from the E00 reader have a feature type dependent on the subfile or the info file from which the feature originated, whereas the E00 writer uses a feature type based on the name of the E00 file.

The following table summarizes the feature types generated for each subfile. If a subfile name in the table below contains an asterisk, then it is really a pattern to match info file names. This convention is required because the names of info files vary from coverage to coverage. The + symbol is used for an alternate asterisk for files containing two wild-card expressions. Therefore, the info file defining text attributes for the ERR annotation layer of the HYDR_SUR coverage is named HYDR_SUR.TATERR, and is referred to as *.TAT+ in the following table. References in the rest of the table row expand * to HYDR_SUR and + to ERR.

Note that the Reader Feature Type listed in the table applies only when reading E00 data, not when writing it. As discussed above, the E00 interprets the feature type as the name of the target E00 file. The subsection titled *Controlling E00 Content* describes a method for redirecting features to a particular subfile.

Subfile Name	Reader Feature Type	Geometry	Additional Attributes
ARC	e00_arcdef	e00_arc	E00_FEAT_ROLE = "e00_arc_def" LPOLY = <id of left polygon> RPOLY = <id of right polygon> FNODE = <id of start node> TNODE = <id of end node> cover_id = <id of arc in coverage> cover_num = <sequence # of arc in coverage>
CNT	e00_centroid	e00_point	E00_FEAT_ROLE = e00_poly_cnt
LAB	e00_label	e00_point	E00_FEAT_ROLE = "e00_label" poly_id = <id of polygon containing label> boundingBoxMin.x = <min x of bounding box> boundingBoxMin.y = <min y of bounding box> boundingBoxMax.x = <max x of bounding box> boundingBoxMax.y = <max y of bounding box>
LOG	e00_log	e00_no_geom	text = <whole line of text from log file>
MTD	e00_mtd	e00_no_geom	FME currently skips this subfile
PAL	e00_polyarc	e00_no_geom	E00_FEAT_ROLE = "e00_poly_arc" arcnum{n} = <record number of ARC record for segment #n> arc{n}.arcnum = <record number of ARC record for segment #n> arc{n}.fnode = <start node of ARC record for segment #n> arc{n}.lpoly = <left polygon id of ARC record for segment #n> boundingBox.minX = <min x coordinate of bounding box> boundingBox.minY = <min y coordinate of bounding box> boundingBox.maxX = <max x coordinate of bounding box>

Subfile Name	Reader Feature Type	Geometry	Additional Attributes
			boundingBox.maxY = <max y coordinate of bounding box>
PRJ	e00_projection	e00_no_geom	E00_FEAT_ROLE = "e00_proj" datum = <Name of datum> projection = <Name of projection> units = <Unit type> xshift = <Shift in x coordinate> yshift = <Shift in y coordinate> zunits = <YES/NO> zone = <UTM zone number> unknown_param{n}.name = <name of non-standard parameter #n> unknown_param{n}.value = <value of non-standard parameter #n>
RPL	e00_polyarc	e00_no_geom	E00_FEAT_ROLE = "e00_region_arc" arcnum{n} = <record number of ARC record for segment #n> arc{n}.arcnum = <record number of ARC record for segment #n> arc{n}.fnode = <start node of ARC record for segment #n> arc{n}.lpoly = <left polygon id of ARC record for segment #n> boundingBox.minX = <min x coordinate of bounding box> boundingBox.minY = <min y coordinate of bounding box> boundingBox.maxX = <max x coordinate of bounding box> boundingBox.maxY = <max y coordinate of bounding box> e00_region_subclass = <name of region subclass> (See the subsection titled <i>Region Support</i> for a description of region-related records.)
RXP	e00_regionxref	e00_no_geom	E00_FEAT_ROLE = "e00_region_xref" name = <name of tolerance type> id = <numeric id of tolerance type> state = <state of tolerance> value = <value of tolerance> (See the subsection titled <i>Region Support</i> for a description of region-related records.)
TOL	e00_tolerance	e00_no_geom	E00_FEAT_ROLE = "e00_tolerance" name = <name of tolerance type>

Subfile Name	Reader Feature Type	Geometry	Additional Attributes
			<p>id = <numeric id of tolerance type> state = <state of tolerance> value = <value of tolerance></p> <p>(See the subsection titled <i>Tolerances</i> for a description of tolerance records.)</p>
TXT	e00_text	e00_text	<p>E00_FEAT_ROLE = "e00_text_def" e00_anno_name = "" e00_anno_id = <record number within TXT file> <Attributes for text geometry></p> <p>(See the <i>Text Representation</i> subsection for a discussion about text geometry.)</p>
TX6 or TX7	e00_text	e00_text	<p>E00_FEAT_ROLE = "e00_text_def" e00_anno_name = <name of anno subclass> e00_anno_id = <position within anno subclass> e00_anno_level = <level number of text feature> e00_num_coords = <number of coordinates defining text position> parameter{} = <unnamed TX6/TX7 parameters> <Attributes for text geometry></p> <p>(See the <i>Text Representation</i> subsection for a discussion about text geometry.)</p>
	e00_textarrow	e00_arc	<p>E00_FEAT_ROLE = "e00_text_arrow" e00_anno_name = <name of anno section> e00_anno_id = <position within anno section></p>
LNK	LNK	e00_point	E00_FEAT_ROLE = "LNK"
*.AAT	*_arcattr	e00_no_geom	<p>E00_FEAT_ROLE = "e00_arc_attr" FNODE_ = <Start node cover#> TNODE_ = <End node cover#> LPOLY_ = <Left polygon cover#> RPOLY_ = <Right polygon cover#> *_ID = <arc identifier> *# = <coverage # of arc> LENGTH = <length of arc> <User-defined attributes></p>

Subfile Name	Reader Feature Type	Geometry	Additional Attributes
*.BND	*_bounds	e00_no_geom	E00_FEAT_ROLE = "e00_bounds" XMIN = <min x of bounding box> YMIN = <min y of bounding box> XMAX = <max x of bounding box> YMAX = <max y of bounding box>
*.PAT	*_polyattr	e00_no_geom	E00_FEAT_ROLE = "e00_poly_attr" AREA = <area of polygon> PERIMETER = <perimeter of polygon> *_ID = <id of polygon> *# = <coverage # of polygon> <User-defined attributes>
	*_pointattr	e00_no_geom	E00_FEAT_ROLE = "e00_point_attr" AREA = 0.0 PERIMETER = 0.0 *_ID = <id of point> *# = <coverage # of point> <User-defined attributes>
*.TIC	*_tic	e00_no_geom	E00_FEAT_ROLE = "e00_tic_point" IDTIC = <TIC point identifier> XTIC = <TIC point x coordinate> YTIC = <TIC point y coordinate>
*.TAT+	*+_textattr	e00_no_geom	E00_FEAT_ROLE = "e00_text_attr" e00_anno_name = <name of annotation layer> *# = <coverage number for text>
*.XCODE	*_textattr	e00_no_geom	E00_FEAT_ROLE = "e00_text_attr" e00_anno_name = "" *# = <coverage number for text>
*.+	*.+	e00_no_geom	E00_FEAT_ROLE = "+." <User-defined attributes>

In addition to the attributes shown in this table, all features read from an E00 file have an attribute named E00_RECORD_NUM, whose value corresponds to the position within the subfile of the record defining the feature. The record numbers start at 1 for each file, and are incremented for each record. This number provides the positional information required to define the relationships between certain geometries and their attributes.

Note that the numbering of the text features is somewhat special. See [Text Representation](#) for further details. Note that the reader also assigns features of most feature types an E00_FEAT_ROLE attribute, which defines the role of the feature within the coverage. This is required to make it easier to create a generic mapping file, when different files processed by that mapping file might have different info file names. For example, the file BART.E00 might have an info file named BART.TIC where JOSIE.E00 has an info file named JOSIE.TIC. The features emitted for these two info files would have a type of BART_tic and JOSIE_tic, respectively, but the features for both info files would have the value of e00_tic_point for their E00_FEAT_ROLE attribute. The role is given to features from the standard subfiles, as well as the info files which have one of the known suffixes – .AAT, .BND, .PAT, .TIC, .TAT+.

If features from a subfile have a particular type of geometry, then they will have an attribute named e00_type, whose value is the geometry type. For example, features from the ARC subfile will have line geometry attached, and will have an e00_type attribute with the value e00_arc.

Text Representation

The main geometry for text features are defined from records in the **TX6**, **TX7**, or **TXT** subfiles of the E00 coverage. This geometry consists of a text string, a location at which to draw the text, and optionally a string of points that form a curved line along which to place the characters. Additionally, text features from the **TX6** or **TX7** subfile might have arrows associated with them¹.

When these features are read into the FME, the form changes slightly. If the keyword <ReaderKeyword>_TEXT_CURVE has been given the IGNORE value, the start and end points of the text line are used to compute the average rotation of the characters, and the first point in the line becomes the text's position. The text feature's geometry is a point which defines the position, along with the following attributes to define the rest of the feature.

Attribute Name	Description
e00_anno_name	Name of annotation layer (subclass) containing text.
e00_anno_id	Sequence number of text features within its annotation layer.
e00_rotation	Rotation of text display, measured in degrees counter-clockwise from horizontal.
e00_text_height	Height of one line of text, measured in ground units.
e00_text_width	Height of the line of text, measured in ground units. When features contain multiple lines of text, this will be the width of the longest line of the text.
e00_tbox_height	Height of entire text block, measured in ground units. If the text contains carriage return characters, and thus spans multiple lines, this number will be greater than the value for e00_text_height; otherwise the two will have the same value.
e00_text_string	String of text being displayed.
e00_text_just	(Optional) Justification of text feature relative to its baseline. This is an integer with a value between 1 and 12, inclusive. The default justification value will be "1", indicating that the bottom of the text character is aligned with the first point of its defining arc.
e00_num_coords	(Optional) When writing TX6 or TX7 features, this attribute defines how many coordinates are to be used to define the text's position. Its value is an integer between 1 and 3 (inclusive); if no e00_num_coords attribute is present, three coordinates will be used to represent the text location.

¹Note that the E00 writer does not currently support any form of text arrow associated with text features. This capability may be added in a later release. Neither does the writer support TXT-style text records; only TX6-style text may be generated.

Attribute Name	Description
e00_text_level	(Optional) Numeric value representing the level of the text feature.
e00_text_symbol	(Optional) Numeric value representing the symbology ArcInfo will use to render the text.

If the keyword <ReaderKeyword>_TEXT_CURVE has been given the FIT value, and the text feature is defined by more than two coordinates, the FME computes a position and rotation of each character of text, generating a separate feature for each character. (No features are generated for whitespace characters.) In this case, all features corresponding to a given ArcInfo text element will have identical values for the e00_anno_name attribute, and for the e00_anno_id attribute, and will also contain two additional attributes:

Attribute Name	Description
e00_whole_text_string	The original text string, from which this feature's single character was taken.
e00_pos_in_whole_text	The position of this feature's character within the original text string. The first character has a position of "1", the second has a position of "2", and so on.

The contents of a **TX6** or **TX7** subfile within an E00 coverage may contain annotation in several different annotation layers (subclasses). Each feature belongs to one subclass, and this subclass' name is contained in the feature's e00_anno_name attribute. The features within a given subclass are numbered as they are read and the e00_anno_id attribute is assigned the feature's sequence number, starting at 1, within the layer.

If there are no named annotation subclasses in the coverage – as is always the case with annotation from the **TXT** subfile – all text features will have an empty string ("") as the value for their e00_anno_name attribute.

If the text has an associated arrow, a separate line feature is generated. This feature is type e00_textarrow, and contains the same values for its e00_anno_name and e00_anno_id attributes as the associated e00_text feature.

Every text feature defined in a **TX6** or **TX7** subfile of an E00 coverage has an associated set of user-defined attributes from a particular info file. Each record of the info file is returned from the E00 reader as a feature with the attributes defined on it. The features have an E00_FEAT_ROLE attribute of e00_text_attr and a feature type of <prefix>_<anno_name>_textattr, where <prefix> is an arbitrary prefix, and <anno_name> is the name of the annotation layer containing the feature. If the annotation layer is unnamed, the features defining the user attributes simply have a feature type of <prefix>_textattr.

The text geometries are associated with their user-defined attributes according to their position within the file. In other words, there is a one-to-one relationship between the text geometries and the features defining the user attribute. This relationship is formed by joining the text feature's e00_anno_name and e00_anno_id attributes with the attribute feature's e00_anno_name and E00_RECORD_NUM attributes.

Text features from **TXT** subfiles do not have named annotation subclasses, and consequently behave like text features from **TX6** or **TX7** files whose e00_anno_name contains an empty string. Note that the user attributes for text defined in the **TXT** subfile come from a different info file than those for text from the **TX6/TX7** subfile – *.XCODE rather than the *.TAT+ info file – but the E00 reader forms the features generated from the two info files identically.

Tolerance Values

E00 coverages contain a list of ten tolerance values, which have a specific meaning within ArcInfo. Each tolerance has a numerical identifier in the range 1..10, a state, and a floating point value.

The E00 reader generates ten features from the TOL subfile. Each feature contains the following attributes:

Attribute Name	Description
id	Original numerical id given to the tolerance.
name	A standard name given to the tolerance record. This name provides a description of the tolerance in question, and is really just a textual version of the above ID. (1=>FUZZY, 2=>GENERALIZE, 3=>NODE_MATCH, 4=>-DANGLE, 5=>TIC_MATCH, 6=>EDIT, 7=>NODESNAP, 8=>WEED, 9=>GRAIN, 10=>SNAP)
state	The state of the tolerance. (1=>tolerance has been verified, 2=>tolerance has not been verified)
value	The size of the tolerance. This is a floating point number, typically smaller than 1.0.

Projections

An E00 coverage may contain a subfile named PRJ, that defines the geographic projection of the coordinates within the coverage. The E00 reader gathers all of the information in this subfile into a single feature of the type e00_projection. The attributes of this feature are listed above, in the subsection titled *Feature Types*.

The PRJ subfile contains a list of named parameters, followed by a list of apparently unnamed parameters. Any of the named parameters, whose names are recognized, are defined as standard attributes – datum, projection, units, etc. on the e00_projection feature. Named parameters, whose names are not recognized by the reader, are placed into the attributes_unknown_parameter{}.name and unknown_parameter{}.value. Unnamed parameters are placed into the attribute list param{}.value.

The E00 attempts to interpret the coordinate system information from the projection feature. If the parameters in the E00 file are from a known coordinate system or projection, the coordinate system information will be passed on the rest of the FME for normal processing. Otherwise a warning message will be logged in the log file. In either case, a feature describing the PRJ file will be passed-in to the FME feature stream.

Likewise, the E00 writer attempts to create a PRJ record from the coordinate system information attached to the features it is writing. The features must all belong to a single coordinate system for this to work. If no mapping can be found from the FME coordinate system to an E00 projection record, a warning will be written to the log file.

Region Support

The E00 reader provides complete support for reading ArcInfo regions, while the E00 writer currently has a very limited form of support for writing ArcInfo regions. An ArcInfo region is essentially a set of non-overlapping polygons with a logical connection. They are represented in the RPL and RXP subfiles, and their attributes are defined in the .PAT<subclass> info files.

The RPL file is virtually identical in structure to the PAL file. It defines the polygons which make up the various regions by listing the arcs that make up the polygons' boundary and holes. The only real difference in structure between the PAL and the RPL is that regions may be divided into subclasses, so the contents of the RPL file are likewise divided into subclasses, with all polygons belonging to regions of a single subclass appearing together within the RPL file.

The RXP file defines the actual regions by cross-referencing the polygons defined in the RPL file with region IDs. Each data line of the RXP file contains a region ID and an RPL polygon ID. The records of the RXP file are grouped by subclass.

Regions' attributes are stored in info files named <baseName>.PAT<subclass>, much as the text attributes are stored in <baseName>.PAT<subclass>. The records of the region attribute tables contain <baseName>-ID attribute that relates them to the regions defined in the RXP file.

Mapping files generated by FME to read E00 include the necessary factories to fully recreate complete regions and output them as area features.

To write regions out to an E00 file, the mapping file must define FME features that represent the RPL, RXP, and .PAT<subclass> records as they are to appear in the output file. This means that the every aspect of the regions, from the topology down to the assignment of IDs, and the ordering of the records with each subfile, must be performed in the mapping file; the E00 writer will simply write out whatever information it is given. In most cases it will be quite difficult to define this information in the mapping file. At some point in the future, the E00 writer will actually take care of computing the various regions and defining the contents of these records, but for the time being it simply provides a way to format the information computed elsewhere.

Info Files

Unlike the standard subfiles, whose names and formats are common to all E00 files, the info files' names and data structures vary from one coverage to another. Each info file starts with a header that defines its name and attributes on each record of the file.

The name of the info file is in the form <prefix>.<extension>, where <prefix> is arbitrary and <extension> defines the role of the records of the info file. Typically, all info files within a single E00 coverage have the same <prefix>. The <extension> is usually from a standard set, which includes the AAT (Arc Attribute Table), PAT (Point or Polygon Attribute Table), and BND (coverage bounding box). The E00 reader uses the extension to determine a role for the content of this info file.

Each record of the info file is interpreted by the E00 reader as an FME feature with no geometry. The <extension> of the info file's name is used to define the feature type and the value of the E00_FEAT_ROLE attribute of these features. The attributes defined on the record as specified in the info file's header are defined verbatim on the output feature.

Generated Mapping Files

Mapping files generated by the FME to read E00 files manipulate and join the features output from the E00 reader to form fully-formed, fully-attributed features with arc, point, polygon, or text geometry. The following sections explain each type of output feature and how it is put together.

Each coverage also contains a single polygon feature defining the bounding box of the coverage, and usually a set of four point features representing the TIC points. These features have polygon and point geometries, respectively, with the feature types <prefix>_bounds and <prefix>_tic.

Mapping files generated by the FME to write E00 files will only write one type of geometry – point, text, arc, or polygon – to each E00 output file. It will also calculate a bounding box of all features for each E00 file's BND subfile, and use the corners of this bounding box to define the TIC points.

Arc Features

In ArcInfo, arcs are simply polyline features with attributes to define a topology, as well as user-defined attributes. The geometry comes from the e00_arcdef features, originating from the ARC subfile and the attributes come from the e00_arc_attr features, originating from the <prefix>.AAT info file. Typically, the attributes defining the topology – left polygon, right polygon, from node, to node – are also defined in the info file, and will appear as attributes on the resulting arc features.

The arc features have a feature type of <prefix>_arc, where <prefix> is the prefix from the info file name. The attributes defined on <prefix>_arc features are summarized in the following table.

Attribute Name	Attribute Value
e00_type	e00_arc
<prefix>-ID	Numerical identifier for arc feature.
<prefix>_	Sequence number of arc feature within the E00 file.
LENGTH	Length of the line, measured in ground units.

Attribute Name	Attribute Value
FNODE_	Sequence number of starting node of the line.
TNODE_	Sequence number of ending node of the line.
LPOLY_	Sequence number of the polygon that lies to the left of the line when travelling from FNODE to TNODE .
RPOLY_	Sequence number of the polygon that lies to the right of the line when travelling from FNODE to TNODE .

In addition, any other attributes defined in the <prefix>.AAT info file are defined on the <prefix>_arc features generated with this mapping file.

Point Features

Point features are generated when the E00 coverage contains a LAB subfile, but no PAL subfile. In this case, the e00_label features originating from the LAB subfile are joined with the attributes of the e00_point_attr features originating from the <prefix>.PAT info file. The resulting point features have a type of <prefix>_point and the attributes from the following table.

Attribute Name	Attribute Value
e00_type	e00_point
<prefix_ID>	Numerical identifier for point feature.
<prefix>_	Sequence number of the point feature.
PERIMETER	0.0

In addition, any other attributes defined in the <prefix>.PAT info file are defined on the <prefix>_point features generated with this mapping file.

Polygon Features

Polygon features are the most complex of the features created by the generated mapping files. The polygon features result from combining four different types of features output from the E00 reader: e00_arcdef, e00_centroid, e00_polyarc, and e00_poly_attr. A combination of these features is performed as follows.

- The polylines defined by the e00_arcdef features in the ARC subfile form the edges of the polygons. They are combined to form each polygon and its holes, according to the contents of the arcnun{ } attributes on each e00_polyarc feature.
- The point geometry from each e00_centroid feature is attached to the corresponding polygon, providing the values for the attributes e00_centroid_x and e00_centroid_y.
- The attributes from the e00_poly_attr features originating in the <prefix>.PAT info file are added to the formed polygon features.

The resulting polygon features have a type of <prefix>_poly and the attributes from the following table.

Attribute Name	Attribute Value
e00_type	e00_poly
<prefix_ID>	Numerical identifier for poly gon feature.

Attribute Name	Attribute Value
<prefix>_	Sequence number of the polygon feature within the E00 file.
e00_centroid_x	X coordinate of polygon's centroid.
e00_centroid_y	Y coordinate of polygon's centroid.
PERIMETER	Outer perimeter of polygon.
AREA	Area of the polygon, measured in square ground units.

In addition, any other attributes defined in the <prefix>.PAT info file are defined on the <prefix>_poly features generated with this mapping file.

Text and Textarrow Features

There are two ways text features are formed in the automatically generated mapping files. The first, and most common, is by combining the text geometries from the TX6 or TX7 subfile with the attributes from the <prefix>.TAT<annoLayer> info file. In this case, the resulting text features have a feature type of <prefix>_<annoLayer>_text, or <prefix>_text if the annotation layer is unnamed. See [Text Representation](#) for an explanation of annotation layers.

Some E00 coverages have their annotation defined in a TXT subfile rather than in a TX6 or TX7 subfile. These features are combined with the attributes of the <prefix>.XCODE info file instead of a <prefix>.TAT<annoLayer> subfile, and will always be contained in an unnamed annotation layer.

In either case, text features will have a feature type of <prefix>_text or <prefix>_<annoLayer>_text, depending on whether they are contained in a named annotation layer, and will have the attributes shown in the following table.

Attribute Name	Attribute Value
e00_type	e00_text
<prefix_ID>	Numerical identifier for text feature.
<prefix>_	Sequence number of the text feature within the E00 file.
e00_anno_name	Name of annotation layer containing text feature. This will be "" if it is in an unnamed annotation layer.
e00_anno_id	Sequence number of text feature within its annotation layer. This number starts at 1 for the first feature in each annotation layer and is incremented for every other feature.
e00_rotation	Rotation at which to display text, measured in degrees counterclockwise from horizontal.
e00_text_string	Textual portion of feature.
e00_text_height	Height of text, measured in ground units.
e00_text_level	Number indicating level of text.

In addition, any other attributes defined in the <prefix>.TAT<annoLayer> or <prefix>.XCODE info file are defined on the <prefix>_text features generated with this mapping file.

If the text geometry originates in the TX6 or TX7 subfile – as opposed to the TXT subfile – it might have a separate linear portion that acts as an arrow pointing from the text to another location. These lines are written out as features with a feature type of <prefix>_<annoLayer>_textarrow or <prefix>_textarrow, and attributes e00_anno_name and e00_anno_id which take the same values as the corresponding <prefix>_<annoLayer>_text or <prefix>_text features.

Occasionally, an E00 file will have e00_text features for which there are no corresponding attributes in the info files. In this case, the feature types of the corresponding text features generated are simply text and textarrow.

The E00 writer is not capable of generating TXT features. Text output from the FME takes place with TX6 or TX7 records. See [Controlling E00 Output](#) for a description of how geometry is formed on output E00 files.

Controlling E00 Output

The E00 writer allows easy generation of E00 files, but also provides a high level of customization to the format and content of the resulting E00 files. In its simplest form, the E00 writer takes FME features defining line, point, text, and polygon features, and writes them to the appropriate subfiles of the E00 file. This mode of operation makes it very easy to write a mapping file which creates E00 files:

- Decide on the names of the E00 files.
- For each file, decide on the type of geometry to go into the file, and the names and types of the attributes.
- Create a DEF line to define the attributes.
- Create the correlation line to direct the features at the appropriate files. Features going to an E00 file must contain an attribute named e00_type, with one of the values e00_point, e00_arc, e00_text, or e00_poly. The writer uses this attribute to determine how the features' geometry is written out. Some geometry might require additional attributes to be defined on the features being written – see the subsection titled *Geometry Composition* for more information.

This will generate all of the ARC, LAB, CNT, PAL, TX6 or TX7, *.BND, *.TIC, *.AAT, *.PAT, and *.TAT+ records needed to describe the features passed-in. This is normally all one will need to create E00 files. However, the E00 writer also contains mechanisms to allow the advanced user to:

- Define specific label locations for polygons.
- Explicitly direct a particular feature to a particular subfile.
- Create multiple info files within a single E00 file, each with its own set of attributes.

These mechanisms are described in the following subsections.

Specific Label Positions

When the E00 writer generates a PAL record to define a polygon, it also generates a centroid (CNT) record and a label (LAB) record. The position of the centroid and the label are normally computed to be some point within the polygon. (The computed location will always be inside the polygon, but not inside any hole of the polygon.) A specific location will be used for the location of the CNT and LAB records, instead of the computed location, if the E00 writer is given a feature with a point-in-polygon (PIP) geometry instead of a polygon or donut geometry. In this case, the point contained in the PIP will define the location of both the label and the centroid.

Explicit Subfile Selection

The E00 writer normally looks for the e00_type attribute to decide how to write out the features. If this attribute is not present, or has the value of e00_no_geom, the writer will look for an attribute named E00_SUBFILE on the input feature. This tells the writer the subfile in which to write the record. This can be one of the standard subfiles (ARC, LAB, TOL, etc.) or any one of the info files.

This provides a powerful tool to the mapping file author. By changing a few attributes on an FME feature, that feature can be directed to almost any part of the E00 file.

There are two warnings associated with explicit subfile selection:

- The first is that the author of the mapping file must ensure that all of the attributes necessary to write the feature are present on the FME feature when it is given to the E00 writer. This is generally true in FME mapping files, but is of particular importance to this option. For the standard files, the attributes defined in the table shown in **Feature Types** must be present on the features. Features destined for info files must provide values to all attributes defined on the info file. The following section, *Info File Creation*, explains how info file attribution works.
- The second is that the mapping file **must not** write features directly to a standard subfile or info file that is also being written to with the normal geometry writing. The writer contains an internal state to keep track of which geometries have been written to each subfile, and can become easily confused if other features are manually inserted into the same subfiles.

Info File Creation

The E00 normally assigns all attributes specified on the DEF line to all info files. This is usually not a problem, since an E00 file will typically just define some geometry of a single type, and associate attributes with each piece of geometry.

However, there are instances where the mapping file author will want more control over the format of the info file. For example, an input file of tabular data can be placed into a specific info file using the explicit subfile selection described in **Explicit Subfile Selection**. The normal means of specifying info file attributes on the DEF line will place the *same* attributes on every info file in the E00 file. If there is other information to be placed into the same E00 file, such as annotation (text features) and linear geometry (arc features). The other info files have to carry attributes from the tabular data's info file and vice-versa.

To overcome this, the DEF line can contain attribute definitions specific to each info file. The forms of syntax for this are listed in the *Writer Keywords* section titled *DEF*.

Suppose the tabular data in the example above is a simple table listing street names, and the minimum and maximum street numbers for each street. Using the normal DEF line syntax, the DEF line for the E00 file might normally be something like:

```
E00_DEF STREET \
    STREET_NAME      char(32) \
    MIN_ADDR         binint \
    MAX_ADDR         binint
```

If the E00 writer were to write lines and text to the above E00 file, as well as directing tabular data to the STREETS.TAB info file using the mechanism described in **Explicit Subfile Selection**, the resulting E00 file would contain lines, text, and STREETS.TAB records, all with the attributes STREET_NAME, MIN_ADDR, and MAX_ADDR.

In contrast, the following DEF line would apply the STREET_NAME, MIN_ADDR, and MAX_ADDR attributes only to the STREETS.TAB records, leaving no attribution on the line or text records:

```
E00_DEF STREET \
    STREETS.TAB:STREET_NAME char(32) \
    STREETS.TAB:MIN_ADDR binint \
    STREETS.TAB:MAX_ADDR binint
```

This DEF line provides us with a more useful description of the data, but it does not give the line and text features any attributes which can be used to relate them to the STREETS.TAB attributes. The following DEF line will attach a STREET_NAME attribute to each line and text feature, as well as generating the same STREETS.TAB file listed above:

```
E00_DEF STREET \
    STREETS.TAB:STREET_NAME char(32) \
    STREET_NAME char(32) \
    STREETS.TAB:MIN_ADDR binint \
    STREETS.TAB:MAX_ADDR binint \
    STREET_NAME char(32)
```

If you specify any attributes for a specific info file, then none of the "general" attributes will be added (other than the # and -ID attributes).

Finally, the following DEF line would create the same E00 file, except that the text features would be left with no attribution at all.

```
E00_DEF STREET \
    STREETS.TAB:STREET_NAME char(32) \
    STREETS.TAB:MIN_ADDR binint \
    STREETS.TAB:MAX_ADDR binint \
    .AAT:STREET_NAME char(32)
```

The E00 writer uses the following heuristic to decide which attributes are defined on a given info file:

1. The .BND and .TIC info files each have a predefined set of attributes and corresponding types which are **always** used, and never supplemented.
2. If the info file is one of the geometry-related info files – .AAT, .PAT, or .TAT+ – have a predefined set of attributes which are always present, but are supplemented by any attributes from **3** or **4** below.
3. If there are any attributes that were specified using the <infoFile>:<attrName> syntax, for this particular info file, they are appended to the info file definition.
4. If there were no info file-specific attributes in **3**, then any attributes which were specified with the normal <attrName> syntax will be appended to the info file definition.

ESRI ArcInfo Generate Reader/Writer

The ESRI® ArcInfo Generate File Reader and Writer allows FME to read and write the simple ASCII format used by the ArcInfo Generate command.

There are several types of Generate files, and each has its own syntax. Currently, point, line, and text Generate files are supported. Both two-dimensional and three-dimensional data can be imported and exported to line and point Generate files. Three-dimensional (3D) Generate files are often used to input data into ArcInfo's TINning package. Text Generate Files are defined to accept only two-dimensional (2D) coordinates.

ARCGEN Quick Facts

Format Type Identifier	ARCGEN
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	Directory or File
Feature Type	.gen
Typical File Extensions	Yes
Automated Translation Support	File base name
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	ARCGEN_GEOMETRY

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	no
circular arc	no		raster	no
donut polygon	no		solid	no
elliptical arc	no		surface	no
ellipses	no		text	yes
line	yes		z values	yes
none	no			

Overview

FME considers a Generate dataset to be a collection of Generate files in a single directory. All Generate file names are required to end with a .gen extension. The type of each Generate file must be defined in the mapping file before it can be read or written.

Tip: The very simple format of Generate files makes them useful for testing purposes or for transferring data between FME and other unsupported systems.

Reader Overview

The ARCGEN Reader produces FME features for all feature data held in Generate files residing in a given directory.

The ARCGEN Reader first scans the directory it is given for all Generate files defined in the mapping file. For each Generate file it finds, it checks to see if that file is requested by looking at the list of **IDs** specified in the mapping file. If a match is made or if no **IDs** were specified in the mapping file, the Generate file is opened for read. The Generate reader extracts features from the file one at a time and passes them on to the rest of the FME for further processing. When the file is exhausted, the Generate reader starts on the next file in the directory. Optionally, a single Generate file can be provided as the dataset. In this case, only that Generate file is read.

Reader Directives

The suffixes listed are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the Generate reader is **ARCGEN**.

DATASET

Required/Optional: *Required*

Contains the directory name of the input Generate files, or a single Generate file.

Workbench Parameter: *Source ESRI ArcInfo Generate File(s)*

DEF

Required/Optional: *Required*

Each Generate file must be defined before it can be read. The definition contains the file's base name (without the .gen extension), the type of geometry it contains, and optionally a delimiter between fields. There may be many DEF lines, one for each file to be read.

The syntax of a Generate **DEF** line is:

```
<ReaderKeyword>_DEF <baseName> \  
    ARCGEN_GEOMETRY (arcgen_point|arcgen_line|    \  
                    arcgen_text) \  
    [ARCGEN_DELIMITER "<delimiter char>"]
```

The file name of the Generate file is the basename plus the .gen extension.

The mapping file fragment below defines two Generate files—one containing point features and the other containing linear features:

```
ARCGEN_DEF nodes ARCGEN_GEOMETRY arcgen_point  
ARCGEN_DEF boundaries ARCGEN_GEOMETRY arcgen_line
```

If no delimiter clause is specified, a comma (,) is used as the delimiter.

IDs

Required/Optional: *Optional*

This directive is used to limit which available and defined Generate files will be read. If no **IDs** are specified, then all defined and available Generate files will be read. The syntax of the **IDs** keyword is:

```
<ReaderKeyword>_IDS <baseName1> \  
    <baseName1> ... \  
    <baseNameN>
```

The basenames must match those used in **DEF** lines.

Example:

The example below selects only the **nodes** Generate file for input during a translation:

```
ARCGEN_IDS nodes
```

Workbench Parameter: *Feature Types to Read*

CLOSED_LINES_AS_POLYS

Required/Optional: *Optional*

This directive specifies how to determine the type of closed lines which may be indistinguishable from polygons.

Example:

```
ARCGEN_CLOSED_LINES_AS_POLYS no
```

Value: YES | NO

Workbench Parameter: *Read closed lines as polygons*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

*** Workbench Parameter**

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

`<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>`

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

`<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]`

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The ARCGEN Writer creates and writes feature data to Generate files in the directory specified by the **DATASET** directive.

As with the reader, the directory must exist before the translation occurs. Any old Generate files in the directory are overwritten with the new feature data. As features are routed to the Generate writer by FME, it determines which file to write to and outputs them according to the type of the file. Many Generate files can be written during a single FME session.

Writer Directives

The Generate writer processes the **DATASET** and **DEF** directives as described in the *Reader Directives* section. It does not make use of the **IDs** directive.

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see the chapter **About Feature Attributes**), this format adds the format-specific attributes described in this section.

All Generate features contain a numeric ID field. The numeric ID is stored in the **arcgen_id** attribute of an FME feature read from a Generate file or destined to be written to a Generate file.

Tip: Features being written to an ARCGEN file must have an arcgen_id attribute.

FME considers the basename of the Generate file to be the *FME feature type* of a Generate feature. The feature type of a Generate feature must match the basename of a Generate file defined by a Generate **DEF** line. Each feature read from a Generate file has an ARCGEN_GEOMETRY attribute added to it that indicates if the feature came from an arcgen_point, arcgen_line, or arcgen_text file. The writer can also handle homogeneous aggregate features of points, lines or text, which also have an ARCGEN_GEOMETRY attribute.

Points

Generate files containing only points consist of a series of lines that follow this syntax:

```
<idNumber>, <x>, <y> [, <z>]
```

Tip: By using the idNumber associated with each Generate feature as a key into a comma separated value file, the @Relate command can be used to attach attributes to Generate features.

The **<idNumber>** is any numeric value, and need not be unique within a file. As well, the **<z>** value is optional and, if it is not specified, the point is considered to be two-dimensional. The file is terminated by a line containing only the word **END**. A two-dimensional point Generate file example is shown below:

```
601, 3, 7
602, 53, 21
603, 19, 20
END
```

Lines

Generate files containing only linear features consist of a series of lines that follow this syntax:

```
<idNumber>
<x0>, <y0> [, <z0>]
<x1>, <y1> [, <z1>]
...
<xN>, <yN> [, <zN>]
END
```

Tip: Polygonal features may also be input and output using linear Generate files. In such cases, the first point and the last point of the line are identical.

As with point files, the **<idNumber>** is any numeric value, and need not be unique within a file. As well, the **<z>** value is optional and, if it is not specified, the linear feature is considered to be two-dimensional. The end of each linear feature is marked by a line containing only the word **END**. A linear Generate file is terminated with two consecutive lines containing only the word **END**. A three-dimensional linear Generate file example, containing two features, is shown below:

```
101
32, 52, 1
33, 56, 2
```

```

36,59,6
31,70,3
END
102
52,32,3
53,56,5
56,29,1
61,73,14
END
END

```

Text

Generate files containing only text (annotation) features, consist of a series of lines that follow this syntax:

```
<idNumber>,<x>,<y>,<angle>,<size>,<text>
```

As with point files, the `<idNumber>` is any numeric value and need not be unique within a file. A Text generate file is terminated with the word `END`. A text Generate file example, containing two features, is shown below:

```

101,32,52,0,20,Arnet Maves
102,52,32,90,30,Barnie Maves
END

```

The attributes used by the generate reader and writer are described in the following table.

Attribute Name	Value
arcgen_rotation	Specifies the rotation of the text in degrees measured counterclockwise from horizontal. Range: -360.0 . . . 360.0
arcgen_text_size	The size of the annotation in ground units. Range: Float > 0
arcgen_text_string	The text string to be placed at the annotation location. Range: Any text string

The example below shows an FME mapping file used to translate some points and linear features from the Generate format into Shape files. The mapping file defines the dataset location and gives the Generate definitions for the two files to be read. At run time, the Generate reader goes out to the directory, reads the files, and produces an FME feature for each Generate feature it finds.

Example

Sample Generate to Shape Mapping File:

```

# -----
# Define the location of the Generate files
ARCGEN_DATASET /usr/data/generate/92i080
# -----
# Define the type of each of the Generate files
ARCGEN_DEF nodes      ARCGEN_GEOMETRY arcgen_point

# -----
# This second file uses a space as the delimiter
ARCGEN_DEF boundaries ARCGEN_GEOMETRY arcgen_line \
      ARCGEN_DELIMITER " "
# -----
# Now define the location of the Shape files
# which will be created
SHAPE_DATASET /usr/data/shape/92i080

```

```

# -----
# Define each of the Shape files.

SHAPE_DEF markers SHAPE_GEOMETRY shape_point \
                MARKER_ID number(6,0)
SHAPE_DEF edges  SHAPE_GEOMETRY shape_polyline \
                EDGE_ID number(6,0)

# -----
# Now define transfer specifications
ARCGEN nodes arcgen_id %nodeNum
SHAPE markers MARK_ID %nodeNum
ARCGEN boundaries arcgen_id %boundNum
SHAPE edges  EDGE_ID %boundNum

```

Tip: Notice the Shape file definitions create a field to store the identifier associated with each generate feature.

If the /usr/data/generate/92i080 directory contained the following files:

nodes.gen	boundaries.gen
601,7,7	101
602,53,21	32 52 1
603,19,20	33 56 2
END	36 59 6
	31 70 3
	END
	102
	52 32 3
	53 56 5
	56 29 1
	61 73 14
	END
	END

...then the FME features shown below would be created by the Generate reader.

Feature Type: nodes	
Attribute Name	Value
ARCGEN_GEOMETRY	arcgen_point
arcgen_id	601
Coordinates: 37,7	

Feature Type: nodes	
Attribute Name	Value
ARCGEN_GEOMETRY	arcgen_point
arcgen_id	602

Feature Type: nodes	
Attribute Name	Value
Coordinates: 53,21	

Feature Type: nodes	
Attribute Name	Value
ARCGEN_GEOMETRY	arcgen_point
arcgen_id	603
Coordinates: 19,20	

Feature Type: boundaries	
Attribute Name	Value
ARCGEN_GEOMETRY	arcgen_line
arcgen_id	101
Coordinates: (32,52,1),(33,56,2),(36,59,6),(31,70,3)	

Feature Type: boundaries	
Attribute Name	Value
ARCGEN_GEOMETRY	arcgen_line
arcgen_id	102
Coordinates: (52,32,3),(53,56,5),(56,29,1),(61,73,14)	

Feature Type: boundaries	
Attribute Name	Value
ARCGEN_GEOMETRY	arcgen_line
arcgen_id	101
Coordinates: (32,52,1),(33,56,2),(36,59,6),(31,70,3)	

Feature Type: boundaries	
Attribute Name	Value
ARCGEN_GEOMETRY	arcgen_line
arcgen_id	102
Coordinates: (52,32,3),(53,56,5),(56,29,1),(61,73,14)	

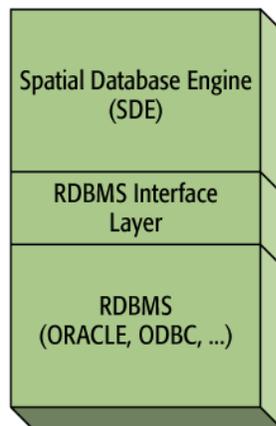
These features would then be transformed by the FME and output to their destination Shape files by the Shape writer.

ESRI ArcSDE Reader/Writer

Format Notes:

- This format is not supported by FME Base Edition.
- This chapter describes FME's ArcSDE 3.x/8.x/9.x support, which is different from SDE 2.1 support. The SDE 2.1 format is deprecated in FME 2009.
- This chapter also applies to ArcGIS 9.x.
- This chapter includes SDE30, SDEMASTER (Reader), SDEMASTERMAP (Writer), SDEMASTERCATALOG (Writer).

ESRI's Spatial Database Engine (SDE) 3.x/ArcSDE 8.x/9.x¹ brings Geographical Information Systems (GIS) into the realm of Management Information Systems (MIS) by providing a spatial interface to industry-standard Relational Database Management Systems (RDBMS).



Spatial Database Engine

SDE enables a Relational Database Management System to store both spatial and non-spatial data by providing a new "Shape" column type to the underlying RDBMS. FME is an SDE client application capable of connecting to the SDE regardless of the platform on which it is located.

Overview

The SDE provides a seamless data model into which geographic data is stored. The SDE provides a relational data model organized around tables. In the FME, an SDE feature type² is equivalent to an RDBMS table. The FME can be used to read and write any RDBMS table whether or not it has a layer (vector spatial column) and whether or not it has a raster column. Spatial geometry in a table can be found in either a layer or a raster column.

A layer (vector spatial column) has the following properties:

¹Throughout this chapter, *SDE* refers to clients using SDE 3.x or ArcSDE 8.x or 9.x, unless otherwise stated. The same reader/writer is used to access all of these clients.

²The terms "feature type" and "table" are used interchangeably throughout this chapter.

- Each layer has a spatial index that can be tuned specifically for it. The spatial index consists of between one and three two-dimensional (2D) grids. The sizes of the grid elements are ordered such that:

grid1 size < 4 X grid2 size (except if grid2 is set to zero)
grid2 size < 4 X grid3 size (except if grid3 is set to zero)

Tip: Since the SDE stores all coordinates as 32-bit (or 64-bit since ArcSDE 9.0) integer coordinates with an implied decimal position, it is possible for precision to be lost or for overflow to occur when features are stored in the SDE. Care must be taken to ensure that the SDE dataset system units preserve the data precision and the range.

- A single relational table can only have 1 layer.
- All features in a layer must be either two- or three-dimensional (2D or 3D). Mixed dimensionality is not allowed in a layer
- Each layer has its own coordinate system, false origin, and scaling factor.
- The layer in a raster catalog is referred to as the 'footprint' column, and stores the bounding box of each raster in the catalog.

A raster column has the following properties:

- A single relational table can only have 1 raster column.
- Each raster column has its own coordinate system. If the coordinate system is not specified, it will be stored as **UNKNOWN**.
- A raster column can either be a raster map storing all raster data in the table as a single raster in a single row, or as a raster catalog storing multiple rasters in multiple rows in the table.

The FME's SDE reader and writer modules take advantage of the unique capabilities of the SDE. The reader module retrieves features from the SDE by constructing queries consisting of both spatial and/or non-spatial components. An SDE database may have a very large number of features, therefore the query building capability is critical to ensure that the FME reader module is capable of satisfying highly focused data requests. The writer module takes advantage of the SDE's transaction model to ease the task of importing data into the SDE. The SDE writer is also able to operate in either an "Update" mode or an "Insert" mode, enabling the FME to be used as a key component in an SDE-based solution. The SDE reader is also capable of performing multi-table join queries, thereby exploiting the full power of the underlying RDBMS.

Note: Version support, spatial constraints, and multi-table joins are not currently supported for raster data.

Tip: See the SDE30QueryFactory in the FME Functions and Factories manual. This factory also exploits the powerful query capabilities of the SDE 3.0/ArcSDE 8.x/ArcSDE 9.x.

See the @SDEsql function in the *FME Functions and Factories* manual. This function allows arbitrary Structured Query Language (SQL) statements to be executed against the SDE's underlying database.

FME and ESRI ArcSDE (SDE30) Compatibility

This section relates to accessing native ArcSDE with the SDE30 Reader and Writer.

It does not apply to FME's Geodatabase Readers and Writers; these access SDE via ArcObjects and have always required an appropriately-licensed version of ArcGIS software to be installed.

Does FME work with ArcSDE 9.3?

In order to avoid problems of version incompatibility, as of FME version 2009, FME no longer installs the required libraries for reading and writing ArcSDE.

You should instead obtain these libraries from an ESRI product to ensure compatibility between FME and your SDE database.

There are three methods for obtaining the required libraries:

1. Install ArcGIS Desktop 9.3, OR
2. Install the SDE C SDK (available on your ArcGIS Server 9.3 DVD) and set up the environment variable described below, OR
3. Install ArcEngine and check for the environment variable described below

For options 2 or 3 you will need to set up a system environment variable as follows:

Variable Name - ARCGISHOME
Value - <ArcGIS Directory>\arcsde\

(that is, pointing to the folder that contains the bin folder which contains the SDE .dll files)

Does FME work with ArcSDE 9.2?

Yes - however the same requirements exist as described above for ArcSDE 9.3. Install a 9.2 product and ensure that the system environment variable **ARCGISHOME** has been set as described above.

Does FME work with ArcSDE 9.1?

Yes. However you will need to download and install some other ESRI product with newer versions of the dlls than are available in an ArcGIS 9.1 install. The simplest solution is to download and install ArcGIS Explorer and then add the ARCGISHOME variable as described above.

Does an ArcGIS 9.x installation make FME compatible with ArcSDE 8.3?

Yes, but for reading features only. An FME installed along with ArcGIS 9.x libraries can, at most, only read features from ArcSDE 8.3.

For more information see this ESRI knowledge base article.

ESRI ArcSDE Quick Facts

Format Type Identifier	SDE30 SDERASTER (Reader) SDERASTERMAP (Writer) SDERASTERCATALOG (Writer)
Reader/Writer	Both
Licensing Level	<ul style="list-style-type: none"> Professional for SDE30 Reader; ESRI for SDE30 Writer; Professional for SDERASTER; ESRI for SDERASTERMAP and SDERASTERCATALOG
Dependencies	ESRI ArcGIS Desktop
Dataset Type	Database
Feature Type	table name
Typical File Extensions	Not applicable
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	Yes
Spatial Index	No
Schema Required	Always
Transaction Support	Yes
Geometry Type	Yes
Encoding Support	Yes

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	no		polygon	yes
circular arc	no		raster	yes
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	yes
line	yes		z values	yes
none	yes			

Raster-specific Quick Facts

Band Interpretations	Red8, Red16, Green8, Green16, Blue8, Blue16, Gray8, Gray16, Int8, UInt8, Int16, UInt16, Int32, UInt32, Real32, Real64
Palette Key Interpretations	UInt8, UInt16
Palette Value Interpretations	RGB24, RGBA32
Nodata Value	Stored based on location (see FME SDE Highlights section)
Cell Origin (x, y)	0.5, 0.5
Rotation Support	No
GCP Support	No
World File Support	No
TAB File Support	No

Notes:

Only rasters with color palettes can be written with the ArcSDE writer. Other types of palettes must either be converted to color palettes or be removed or resolved to continuous rasters in order to be written. Raster catalogs must be registered with the Geodatabase via ArcCatalog, in order for them to be viewed properly. Unregistered raster catalogs will appear as tables.

FME SDE Highlights

The SDE reader and writer modules provide the FME with the ability to store data in and retrieve data from ESRI's SDE.

The SDE modules deliver the following capabilities.

- **Programmatic Table Creation:** Tables need not be created before a data import operation. All table creation details are handled by the FME.
- **Programmatic Index Creation:** Non-spatial column indices can be specified within FME mapping files. These indices are used to enhance the performance of the non-spatial component of searches.
- **Programmatic Spatial Column and Attribute Verification:** When loading data into an existing spatial database, the FME verifies that the table definitions specified in the mapping file match the existing RDBMS definitions.
- **Versioning Support:** FME enables data to be read from a particular version of an SDE database, and also allows data to be written to a specific version of an SDE database.
- **Transaction Support:** Transactions are fully supported enabling a partially complete load operation to be resumed later, without the loss of or duplication of data.
- **Attribute Query Support:** SQL **WHERE** clauses can be specified to limit the data being exported.
- **Multi-Table Query Support:** The queries may combine multiple RDBMS tables thereby exploiting the full SQL capabilities of the underlying RDBMS.
- **Multiple SDE Connections:** The FME is capable of having multiple SDE connections open to the same or different SDE databases. This may be used for selective replication between different SDE sites, for moving data from a preparation SDE to a production SDE, or for building result sets of features from multiple SDE databases.
- **Spatial Query Support:** FME exploits the spatial query capabilities of the SDE.

- **Non-Spatial Table Support:** Any table can be read or written to an SDE database with FME whether or not it is spatially enabled. For example, if your SDE uses Oracle then FME can be used to read, write, or create regular Oracle tables.
- **Rejected Features' Pipeline:** Features whose geometry is rejected by SDE can be sent through an FME Pipeline and modified so that they can be given a second opportunity to be inserted into SDE.
- **Geodatabase Reading/Writing Support:** Features can be read from or written to an existing feature class located on an Enterprise Geodatabase.
- **Unicode Support:** With SDE 9.2 and later, FME can read from and write to text fields encoded in UTF-16.
- **Fully Automatic Import and Export:** FME's SDE support provides fully automated import and export of data through the FME's Graphical User Interface (GUI). This is ideal for quick data imports or quick data exports.

Tip: The query support enables the data export operation to be highly focused, thereby reducing the amount of data that is exported.

- **Raster Support:** Raster reading and writing can be done through FME, and includes support for multiple bands, nodata values, mosaicking, compression and pyramiding as well as storage of data in either raster maps or raster catalogs.

Note: Nodata values in ArcSDE raster tables are stored as locations rather than explicit values. FME maps these nodata locations in the raster to a single, unused value within the data range using statistics calculated on the raster band(s). Therefore, in order to retrieve nodata values from a raster table, statistics must be calculated prior to reading from it. This can be done either through the FME ArcSDE raster writer by setting the statistics option to 'AUTO', or by manually calculating statistics on the raster after it has been written using ArcGIS applications or command line tools.

The SDE modules within FME are designed to work with other SDE products. For example, a user with a simple GUI search engine can easily identify all features satisfying a complicated query, then use FME with the SDE reader module to process these features.

Connecting to SDE

Connecting to SDE is the same for the reader and the writer. They both use the same directives in the same manner. There are two possible ways to connect to SDE:

- connecting to the ArcSDE service, which in turn communicates with the underlying database (a three-tier architecture), and
- connecting to the underlying database directly (a two-tier architecture).

With a direct connection (the two-tier architecture), ArcSDE does not need to be installed and an ArcSdeServer license is only needed if writing to the database; reading does not require a license.

Regular Connection

The connection parameters needed for a regular connection are as follows:

DATASET

Required/Optional: *Required*

This statement identifies the SDE database from which features are retrieved/written. In SDE, this dataset is referred to as the **DATABASE**. This is required no matter what the underlying RDBMS of the SDE. Some RDBMSes, such as Oracle, do not require a value, whereas others, such as SQLServer, do. For databases that do not require the value, the value **not_used** is specified by convention.

Example:

SDE30_DATASET testdset

Workbench Parameter: *Source/Destination ESRI ArcSDE Dataset*

SERVER

Required/Optional: *Required*

This statement identifies the SDE server used to read data from the dataset.

Example:

```
SDE30_SERVER tuvok
```

Workbench Parameter: *Server*

INSTANCE

Required/Optional: *Required*

The instance to which the FME is to connect. The usual value for systems with a single SDE instance is `esri_sde`. The instance can also be of the form `port:<port-number>`.

Example:

```
SDE30_INSTANCE esri_sde
```

Workbench Parameter: *Instance Name*

USERID

Required/Optional: *Optional if connecting in OSA mode*

The user name required to access the SDE database.

If the userid and/or password are missing or not set, then the reader will try and connect with Operating System Authentication.

Example:

```
SDE30_USERID ronny
```

Workbench Parameter: *User ID*

PASSWORD

Required/Optional: *Optional if connecting in OSA mode*

The password associated with the specified user ID.

If the userid and/or password are missing or not set, then the reader will try and connect with Operating System Authentication.

Example:

```
SDE30_PASSWORD ronpassword
```

Workbench Parameter: *Password*

VERSION_NAME

Required/Optional: *Optional*

The SDE version to which FME connects. The version must already exist and the current user must have privileges set so that it can access the version. If the `VERSION_NAME` directive is not used, then the FME attempts to connect to `SDE.DEFAULT`. If there is no SDE schema, FME then attempts to connect to `dbo.DEFAULT`. If the name is not prefixed by the owner of the version, then it is assumed that the owner is the current user. This directive is only applicable when dealing with versioned tables. This directive is not available when reading/writing raster data, since versioning is not currently supported.

Default: `SDE.DEFAULT`

Example:

SDE30_VERSION_NAME ron.working-version

Workbench Parameter: *Version*

Direct Connection

The parameters needed to make a direct connection to SDE depend on the underlying database. In order to make a direct connection, the SDE must be of the same major & minor version as the client libraries with which the ArcSDE reader/writer was built.

For example, a direct connection to an ArcSDE 9.1 instance could only be made with a reader or writer built using 9.1 libraries.

Underlying Database	Mandatory Directive	Value
Oracle (option 1)	DATASET	Any value can be specified as the value does not get used; however, a value must be supplied.
	INSTANCE	sde:oracle or sde:oracle9i (for 9i connections to use the right driver)
	USERID	<username>
	PASSWORD	<password>@<Oracle Net Service Name>
Oracle (option 2)	DATASET	Any value can be specified as the value does not get used; however, a value must be supplied.
	INSTANCE	sde:oracle:;/local=<sqlnetalias>
	USERID	<username>
	PASSWORD	<password>
MS SQL Server	DATASET	<database_name>
	INSTANCE	sde:sqlserver:<SQL Server Instance Name> or sde:sqlserver:<SQL Server Instance Name>\<Named Instance> (for connecting to a named instance)
	USERID	<username>
	PASSWORD	<password>
DB2 (option 1)	DATASET	<db alias name specified through DB2 Configuration Assistant>
	SERVER	remote (if client application is remote, otherwise do

Underlying Database	Mandatory Directive	Value
		not specify)
	INSTANCE	sde:db2
	USERID	<username>
	PASSWORD	<password>
DB2 (option 2)	DATASET	Any value can be specified as the value does not get used; however, a value must be supplied.
	SERVER	remote (if client application is remote, otherwise do not specify)
	INSTANCE	sde:db2: <db alias name specified through DB2 Configuration Assistant>
	USERID	<username>
	PASSWORD	<password>
Informix	DATASET	Any value can be specified as the value does not get used; however, a value must be supplied.
	SERVER	remote (if client application is remote, otherwise do not specify)
	INSTANCE	sde:informix: <odbc data source name>
	USERID	<username>
	PASSWORD	<password>

The directive **VERSION_NAME** can also be used to specify the version when making a direct connection.

Please refer to http://webhelp.esri.com/arcgisdesktop/9.2/index.cfm?TopicName=Properties_of_a_direct_connection_to_an_ArcSDE_geodatabase for more information on setting up the direct connect environment and tips on proper usage.

Reader Overview

The SDE reader begins by starting an SDE session with the server upon which the SDE dataset resides. Once connected, the SDE reader queries the SDE and passes the resulting features – that is, rows – on to the FME for processing.

When reading features from the SDE, the tables from which features are retrieved are specified in the mapping file using the **<ReaderKeyword>_IDS**. An optional spatial component and **WHERE** clause may also be specified. If no spatial or attribute constraints are specified, then all features from the identified table(s) are read.

The SDE reader requires that a `<ReaderKeyword>_IDs` statement be specified, identifying the tables from which data is to be retrieved. If no identifiers (IDs) are specified, then no features are read from the database.

The table below summarizes the different feature retrieval modes supported by the SDE reader module for a Spatial Retrieval search type. The next section contains an in-depth discussion of each of these keywords.

Search Keyword Suffix	Description
IDs	<p>Specifies the tables from which features are to be retrieved. If no tables are specified, then no features are retrieved. Each ID specified may be simple or compound.</p> <p>A simple ID is an ID that specifies only one table.</p> <p>A compound ID is one in which several tables are specified – this is not supported when reading raster data.</p> <p>When a compound ID is specified, features are constructed by joining several tables together during the query.</p> <p>A single IDs statement consists of one or more IDs, each of which may be simple or compound.</p>
SEARCH_ENVELOPE	<p>Specifies the spatial extent of the feature retrieval. Only features that have the relationship specified by <code>SEARCH_METHOD</code> with the envelope are returned.</p> <p>The only valid value for <code>SEARCH_METHOD</code> for raster features is <code>SDE_ENVELOPE</code>.</p>
SEARCH_FEATURE	<p>Specifies a feature with an arbitrary number of coordinates as the search feature. Only features that have the relationship specified by <code>SEARCH_METHOD</code> with the search feature are returned.</p> <p>Currently only valid for vector features.</p>
SEARCH_METHOD_FILTER	<p>Specifies if the features returned will or will not satisfy the spatial constraint. If <code>FALSE</code> is specified, then all features returned will not satisfy the spatial constraint. If <code>TRUE</code> is specified, then the features returned will satisfy the spatial constraint. Specifying <code>FALSE</code> enables you to select all features which are not contained by a feature, for example.</p> <p>Value: <code>FALSE</code> <code>TRUE</code></p> <p>Default: <code>TRUE</code></p>
WHERE	<p>Specifies the attribute constraint that a feature must have to be retrieved. Any valid SQL WHERE clause may be entered here as this value is passed directly to the underlying RDBMS for processing.</p> <p>When compound IDs are used, the <code>WHERE</code> clause specifies how the tables are joined during the query.</p> <p>Currently only valid for vector features.</p>

Note: Search feature and multi-table joins are not currently supported for raster tables.

Reader Directives

The suffixes shown are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for reading vector data is `SDE30`, the default for reading raster data is `SDERASTER`.

RETRIEVE_ALL_SCHEMAS

Required/Optional: *Optional*

Specifies whether to retrieve schemas for all the tables in the database. This directive is only applicable when generating a workspace/mapping file or when calling `IFMEUniversalReader::readSchema()` through FME Objects. The type of tables returned depends on whether the `SDERASTER` or `SDE30` reader is being used. When using the raster reader then the raster tables are returned and when using the vector reader both vector and non-spatial tables are returned.

Value: YES | NO

Default Value: NO

RETRIEVE_ALL_TABLE_NAMES

Required/Optional: *Optional*

Specifies whether to retrieve all the table names in the database. This directive can only be used within FME Objects applications through `IFMEUniversalReader::readSchema()`. Unlike the `RETRIEVE_ALL_SCHEMAS` directive, the schema features contain only the feature type which represents the table name from the database. When using the raster reader then the raster tables are returned and when using the vector reader both vector and non-spatial tables are returned.

Value: YES | NO

Default Value: NO

PERSISTENT_CONNECTION

Required/Optional: *Optional*

Specifies whether to create a connection to SDE that persists and can be shared by other SDE Readers, Writers, and SDE30QueryFactories. When set to **YES**, the connection will remain open until FME shuts down, even if this reader is finished using it. Otherwise, the connection will be closed when the reader is shut down (unless another reader/writer/queryfactory is still using the connection).

Creating a new connection is an expensive operation. Depending on how FME is being used (that is, if there are multiple instances of the SDE Reader/Writer being used, or if the `SDE30QueryFactory` is being used to query/update the same SDE), the performance may improve by setting this directive to **YES**.

Value: YES | NO

Default Value: NO

Example:

```
SDE30_PERSISTENT_CONNECTION YES
```

Workbench Parameter: *Make Connection Persistent*

WHERE

Required/Optional: *Optional*

Note: Currently only valid for vector features.

The specified **WHERE** clause is passed to the SDE for processing. **WHERE** clauses may be arbitrarily complex, limited only by the underlying RDBMS. The **WHERE** clause may specify join operations; for example, when a retrieval operation is from multiple tables.

When a join operation is being performed, the FME features have all of the attributes from the various tables combined into each feature. If more than one table has a column with the same name, then the feature attributes for those columns will be qualified with the table name to ensure that each attribute is unique. If a query operation is performed across multiple tables, the spatial component will only be taken from the primary table. In fact, none of the secondary tables should have a spatial column (layer or raster column). If the primary table does not have any spatial component, then the feature will not have any spatial component. See *IDs for a description of how compound IDs are specified*.

If a table contains a layer (vector spatial column), then it is also possible to select one particular feature using the following **WHERE** clause syntax:

```
SE_ROW_ID = <integer>
```

where **<integer>** is some integer value and **SE_ROW_ID** is a virtual column provided by the SDE. Only the '=' operator can be used when the query involves **SE_ROW_ID**. A query involving **SE_ROW_ID** cannot be arbitrarily complex. It can ONLY be of the form **SE_ROW_ID = <integer>**.

In the event of an error, the FME will log the entire **<WHERE clause>**. This clause can then be debugged in the native RDBMS.

Value: *<WHERE clause>*

The attribute constraint used to limit the features which are retrieved based on attribution. When the **WHERE** clause references multiple tables, these tables must all be specified as a compound **SDE30_ID** value. See *IDs for a complete discussion*.

Example 1:

The **WHERE** clause specified below instructs the FME to retrieve features from the database for the table(s) identified by **<ReaderKeyword>_IDs**, and for those rows in which the **NUMLANES** column has a value of **2** and the **SURFACE** column has a value of **'GRAVEL'**. Notice the use of the double quotes around the compound **WHERE** clause.

```
SDE30_WHERE "NUMLANES=2 AND SURFACE='GRAVEL'"
```

Example 2:

The **WHERE** clause below illustrates how to perform a join on 3 different tables named **REPLICATION**, **DIST_CENTER**, and **REPLICATION_LAYERS**. Each of these tables must be specified in the associated **SDE30_IDs** clause for this example to work. Again, notice the use of the double quotes around the compound **WHERE** clause. Also note the use of the continuation characters when building a long **WHERE** clause.

```
SDE30_WHERE \  
"REPLICATION.REPLICATION_DATE is null AND" \  
"REPLICATION.AREA_CD = DIST_CENTER.AREA_CD AND" \  
"REPLICATION.IDENT = REPLICATION_LAYERS.IDENT AND" \  
"DIST_CENTER.TOWN_CD = '$(TOWN)'"
```

Workbench Parameter: *Where Clause*

REMOVE_TABLE_QUALIFIER

Required/Optional: *Optional*

Specifies whether to keep or remove the table name prefix. If the ArcSDE resides on a database (that is, MS SQL Server) where a specific value for database is set, then the full name for a table is **<database_name>.<owner_name>.<table_name>**. If the ArcSDE is located on a database (that is, Oracle) that does not require the database

field, then the full name of a table is `<owner_name>.<table_name>`. Setting this keyword to **YES** indicates that the reader should return the table name without any prefixes. This is useful when:

- creating a workspace that will be passed on to another organization using the same table names,
- performing a translation to another database format but with a different user name, and
- when writing to a file-based format but not wanting the prefix in the name of the feature type.

When this keyword is set to **YES** during the generation of a mapping file or workspace, the source feature types will be the table names without any prefix; otherwise, they will contain the owner name as a prefix. It is recommended that this keyword not be changed in value after generating the mapping file/workspace as it is possible for no features to be successfully passed onto the writer (since the writer is expecting feature types with different names).

Note that even when **REMOVE_TABLE_QUALIFIER** is set to **YES**, if the table is owned by a user other than the current user, the `<owner_name>` prefix will **not** be dropped so that the reader will find the correct table; however, the `<database_name>` prefix will still be dropped.

Value: YES | NO

Default Value: NO

Example:

```
SDE30_REMOVE_TABLE_QUALIFIER YES
```

Workbench Parameter: *Remove Schema Qualifier*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

✳ Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

Mapping File Example:

```
SDERASTER_SEARCH_ENVELOPE 601190 5437839 611110 5447549
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The `COORDINATE_SYSTEM` directive, which specifies the coordinate system associated with the data to be read, must always be set if the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` to the reader `COORDINATE_SYSTEM` prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

SEARCH_FEATURE

Required/Optional: *Optional*

Note: Currently only valid for vector features.

The **SEARCH_FEATURE** clause provides a mechanism for specifying an arbitrarily complex search feature. The **SEARCH_FEATURE** clause works with the **SEARCH_METHOD** clause to define the spatial constraint.

Parameter	Contents
[<xCoord> <yCoord>]+	A list of the coordinates defining the geometry of the query feature.

Example:

The example below defines an equivalent feature to the **SDE30_SEARCH_ENVELOPE** example shown above using the **SDE30_SEARCH_FEATURE** clause.

```
SDE30_SEARCH_FEATURE    601190 5437839 601190 5447549 \  
611110 5447549 611110 5437839 \  
601190 5437839
```

Workbench Parameter: *Search Feature*

SEARCH_METHOD

Required/Optional: *Required when SEARCH_FEATURE is used*

This statement specifies the type of spatial relationship the query features must have with the **SEARCH_FEATURE** or **SEARCH_ENVELOPE**, whichever is used, in order to be returned.

When **SEARCH_ENVELOPE** is used, the default value is SDE_ENVELOPE for vector mapping files/workspaces, SDE_AREA_INTERSECT for vector FME Objects applications, and SDE_ENVELOPE for all raster related translations. When **SEARCH_FEATURE** is used, there is no default value and so this keyword must be specified.

When reading rasters, SDE_ENVELOPE is the only valid value and the **SEARCH_FEATURE** keyword is not supported.

Value:

The value of the **search_method** can be one of the following:

Parameter	Contents
SDE_ENVELOPE	Features must be within the envelope of the feature.
SDE_ENVELOPE_BY_GRID	Features must be within the envelope of the feature and are returned in grid order.
SDE_COMMON_POINT	Features must have a point in common with the query feature.
SDE_LINE_CROSS	Features must cross the query feature.
SDE_COMMON_LINE	Features must have a common line segment with the query feature.
SDE_CP_OR_LC	Features must have either a common point or a line crossing.
SDE_AI_OR_ET	Features must intersect or must share an edge.
SDE_AREA_INTERSECT	Features must intersect the query feature. This retrieves area, linear, and point features contained in or that intersect the area of the query feature.
SDE_AI_NO_ET	Features must intersect but must not have any edge touching with the query feature.
SDE_CONTAINED_IN	The returned features contain the query feature. A candidate feature that is an area will be returned when it encloses the query feature. If the candidate feature is a line, then it is returned if its path is coincident with the query feature. If the query feature is a point and the candidate feature is not an area, then the candidate feature will be returned if one of its vertices is the same as the query feature.
SDE_CONTAINS	The returned feature is contained by the query feature. If both the features are linear features, then the returned feature must lie on the search feature's path. Point features that lie on a search feature vertex are also returned.

Parameter	Contents
SDE_CONTAINED_IN_NO_ET	Features must be contained within the query feature and not have any edge touching.
SDE_CONTAINS_NO_ET	Features must contain the query feature but must not share any edge.
SDE_POINT_IN_POLY	Returned feature must be an area feature that contains the first coordinate of the search feature.
SDE_IDENTICAL	The returned feature must be spatially identical to the query feature.

Example:

`SDE30_SEARCH_METHOD SDE_AREA_INTERSECT`

Workbench Parameter: *Search Method*

SEARCH_METHOD_FILTER

Required/Optional: *Optional*

Specifies if the features returned will or will not satisfy the spatial constraint. If FALSE is specified, then all features returned will not satisfy the spatial constraint. If TRUE is specified, then the features returned will satisfy the spatial constraint. Specifying FALSE enables you to select all features which are not contained by a feature.

Value: *TRUE | FALSE*

Default Value: *TRUE*

Example:

`SDE30_SEARCH_METHOD_FILTER FALSE`

Workbench Parameter: *Search Method Filter*

SEARCH_ORDER

Required/Optional: *Optional*

Note: Currently only valid for vector features.

Specifies the order that the underlying search is performed by the SDE.

Values:

OPTIMIZE – let SDE decide which to perform first

SPATIAL_FIRST – perform spatial query first

ATTRIBUTE_FIRST – perform tabular query first

Default Value: *OPTIMIZE*

Example:

`SDE30_SEARCH_ORDER ATTRIBUTE_FIRST`

Workbench Parameter: *Search Method Order*

IDs

Required/Optional: *Required*

This statement specifies the tables from which features are to be retrieved. There may be multiple **SDE30_IDS** statements within a single FME mapping file, in which case the input set of tables comprises the union of all **SDE30_IDS** statements. The SDE reader module only extracts features from the identified tables.

If a read operation is performing a join operation then the **SDE30_IDS** for the join operation are specified using a compound ID.

Compound IDs have the following form:

```
SDE30_IDS A(B,C)
```

where the primary table in the query is A and the join operation during the read is joining with secondary tables B, and C. Secondary tables cannot contain layers (vector spatial columns) or raster columns.

The join operation that combines the tables is specified on the **SDE30_WHERE** clause. See the section **WHERE**, for a description of the **SDE30_WHERE** clause. The second example below illustrates how the **SDE30_IDS** are specified when the reader performs a multi-table join operation. Joins are not supported when reading rasters.

The general form of **SDE_IDS** is the name of the tables from which features are retrieved, separated with spaces:

```
<[table name[(sec_table[,sec_table]*)]]+>
```

Example 1:

For simple single table extracts, the **SDE30_IDS** is a list of table names as shown below, where features are read from the tables **roads**, then **streets**. No table combine operation occurs here. Each ID is treated as a separate query to the database.

```
SDE30_IDS roads streets
```

Example 2:

For more sophisticated queries in which join operations are desired, the **SDE30_IDS** has the following form. Note that there are no spaces between tables associated with a single query – spaces are used to separate different table identifiers. Each table identifier may be simple, as in Example 1, or compound, as in this example. The following ID statement is the counterpart to the second example for the **SDE30_WHERE** clause.

```
SDE30_IDS REPLICATION(DIST_CENTER,REPLICATION_LAYERS)
```

Note: Joins are not supported when reading rasters.

ENVELOPE_QUERY_OPTIMIZATION

Required/Optional: *Optional*

Note: Valid only for vector features.

By default (that is, when this directive is set to **NO** or is not specified at all), the SDE Reader compares the envelope query used (if an envelope query is being used) to the largest possible extents for a given layer (vector spatial column). The largest possible extents are calculated using the X, Y origin of the layer and the X,Y scale of the layer. If the query envelope completely covers the largest possible extents for the current layer being read, then the envelope query is ignored for the current layer.

If this directive is specified with a value of **YES**, then the SDE Reader compares the envelope query used (if an envelope query is being used), to the extents of the layer as indicated by running the SDE administration command:

```
sdelayer -o describe_long -l <table name, layer name>
```

However, for this optimization to work, it is important that the extents of the layer are correct before running a translation. The extents can be updated using the SDE administration command:

```
sdelayer -o alter -l <table name, layer name> -E calc
```

If the extents are actually larger than currently set in SDE, running a translation may result in ignoring the query envelope used and returning erroneous features (i.e., features that would not be within the query envelope). However, if the extents are correct and if the query envelope completely covers the extents of the current layer, it is okay to ignore the envelope query for the current layer.

The purpose of this directive is to attempt to reduce the amount of time SDE spends executing a query on a table. One query is performed on each table being read (a multi-table join is considered one table). The greater the number of rows in a table, the longer a query may take.

If an envelope query is not being used, then setting this directive to either **YES** or **NO** will have no effect.

Value: YES | NO

Default Value: NO

Example:

```
SDE30_ENVELOPE_QUERY_OPTIMIZATION YES
```

CHILD_VERSION_NAME

Required/Optional: *Optional*

Note: This directive is only applicable on releases of ArcSDE that support versioning. It is also only valid for vector features.

Specifies the name of a child version to create using the version specified by **VERSION_NAME** as the parent version. All tables will then be read from this (child) version rather than from the parent version. All the tables read when this keyword is specified must be multiversioned and have read, insert, update, and delete permissions with the current user. This is because this keyword is designed to be used when checking out a copy of the data in the parent version, with the intention that the child version will be modified and possibly reconciled and posted back to the parent version. The version will be created as a public version and the description given to the version will be "Safe Software Created Version".

If **CHILD_VERSION_NAME** specifies a version that already exists, an error will be output. The child version will be owned by the user specified by **USERID**; therefore, if the owner is specified as part of the value for this directive, the owner must be the same as **USERID**. If the translation is aborted, then the child version created will be deleted. By default, this directive is left empty and therefore no child version is created.

Note: Version names are case-sensitive and hence the value supplied for this directive is also case-sensitive.

Example:

In the example below, **jim** is the name of the user who will own the child version when it gets created and **jim** is also the value that is currently used for the directive **USERID**. Specifying a different value for the user here than the one used in **USERID** will cause an error. Remember that it is not necessary to prefix the child version name with the user.

```
SDE30_CHILD_VERSION_NAME jim.field_edits
```

Workbench Parameter: *Child Version Name*

MAX_FEATURES

Required/Optional: *Optional*

When specified, determines the maximum number of features the reader is allowed to read. Setting this keyword to zero means no limits are imposed on how many features can be read. This can be useful when it is uncertain how many features satisfy a given query, or when you want to read all the features in a given table unless there are many more than expected.

Value: *positive integer*

Default Value: 0

Example:

`SDE30_MAX_FEATURES 0`

Workbench Parameter: *Max features to read*

RASTER_PYRAMID_LEVEL_TO_READ

Required/Optional: *Optional*

When specified, determines the reduced resolution pyramid level to read raster data from. Zero denotes the base, full resolution pyramid. The raster must have had pyramids built when it was written to ArcSDE, and the pyramid level specified must be in the range of valid pyramid levels built.

Value: *positive integer*

Default Value: *0*

Example:

`SDE30_MAX_FEATURES 2`

USE_UNIFIED_DATE_ATTRS

Required/Optional: *Optional*

Specifies whether we want to use unified date attributes, where the date and time are read into one attribute, or whether we want to use split date attributes, where two attributes are produced, one with only the date and another with both the date and time.

The value of this keyword should not be changed. It is automatically set to YES in new mapping files and workspaces. To maintain backwards compability, if this keyword is not present, the reader will behave as though the keyword is set to NO.

Value: *YES | NO*

Default Value: *YES (in new mapping files and workspaces), NO otherwise*

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

*** Workbench Parameter**

Additional Attributes to Expose

Complete Reader Example

Simple Reader Example

The example below configures an SDE reader to extract features from the dataset `testdset` located on server `tuvok`. Only features located on the table named roads, that fall within the specified envelope and have an attribute

`NUMLANES` with a value of 2 and an attribute `SURFACE` with a value of `GRAVEL`, are read from the SDE.

```
SDE30_DATASET testdset
SDE30_SERVER tuvok
SDE30_WHERE "NUMLANES=2 AND SURFACE='GRAVEL'"
SDE30_USERID joe
SDE30_PASSWORD bounce
SDE30_INSTANCE esri_sde
SDE30_VERSION_NAME joe-version
SDE30_SEARCH_ENVELOPE 601190 543783 611110 5447549
SDE30_IDS roads
SDE30_SEARCH_METHOD SDE_AREA_INTERSECT
```

Multi-table Join Example

The example below shows the specification of a multi-table join being performed during reading from the SDE database. The user specifies the tables upon which the query is to be performed using the `SDE30_IDS` clause, while the `SDE30_WHERE` clause specifies how the tables are to be combined to select the features to read. Once the compound ID is specified along with the `WHERE` clause, the features are processed as any other feature. The feature type assigned to the feature is the name of the primary table – in this example, this is `REPLICATION`.

```
SDE30_DATASET testdset
SDE30_SERVER tuvok
SDE30_USERID joe
SDE30_PASSWORD bouncy
SDE30_INSTANCE esri_sde
SDE30_VERSION_NAME joe-version

# Specify the tables upon which the table join is to be performed.
SDE30_IDS REPLICATION(DIST_CENTER,REPLICATION_LAYERS)

# Now specify the WHERE clause which specifies how the three
# tables above are to be combined
SDE30_WHERE \
"REPLICATION.REPLICATION_DATE is null AND " \
"REPLICATION.AREA_CD = DIST_CENTER.AREA_CD AND " \
"REPLICATION.IDENT = REPLICATION_LAYERS.IDENT AND " \
"DIST_CENTER.TOWN_CD = '${TOWN}' "
```

Writer Overview

The SDE writer module stores FME features in an SDE database. The SDE writer module provides the following capabilities.

- **Transaction Support:** The SDE writer provides transaction support that eases the data loading process. Occasionally, a data load operation terminates prematurely due to data difficulties. The transaction support provides a mechanism for reloading corrected data without data loss or duplication.
- **Table Creation:** The SDE writer module uses the information within the FME mapping file/workspace to automatically create SDE tables as needed. If the tables will be storing geometry, then feature classes, raster maps, or raster catalogs can be created.
- **Table Validation:** When data is loaded into an existing table, the SDE writer module performs validation operations between the layer or raster column definition in the mapping file and that within the SDE. All discrepancies found are logged. All critical discrepancies result in the data load operation being halted.
- **Versioning Support:** The SDE writer provides versioning support that allows users to modify data to existing tables without affecting what other viewers see in their own versions of the SDE database. **Note:** Not currently supported for raster tables.
- **Geometry Demotion:** Demotes incorrect polygons to linear shapes if there are data problems, but the data is to be loaded "as is" into the SDE.

- **Non-Homogeneous Aggregate Loading:** Provides the ability to load non-homogeneous aggregates into SDE layers by splitting the aggregates into several homogeneous aggregates.
- **Non-spatial Index Creation:** The SDE writer module can also define non-spatial indices. Indices are specified to increase the performance of searches having a non-spatial component.
- **Update Capability:** The SDE writer module enables features to be updated in SDE through the use of specified keys and the **UPDATE** mode of operation. If area replacement of features is desired, this can also be accomplished by combining the capabilities of the SDE writer with the **SDE30QueryFactory**. **Note:** Updating is the only way to add data to a raster map table, and is referred to as mosaicking.
- **Rejected Features' Pipeline:** Features initially rejected by SDE can be sent through an FME Pipeline where changes can be made to the feature so that it can be accepted by SDE on its second attempt.
- **Geodatabase Writing Support:** Features can be written to an existing feature class located on an Enterprise Geodatabase.

Writer Directives

This section describes the keywords the SDE writer module recognizes. Each of the keywords is prefixed by the current `<writerKeyword>_` when they are placed in a mapping file. By default, the `<writerKeyword>` for writing vector data is **SDE30**, the default for writing rastermap data is **SDERASTERMAP**, and the default for writing raster catalog data is **SDERASTERCATALOG**.

RECONCILE_AND_POST

Required/Optional: *Optional*

Note: Valid only for vector features. Not supported for raster tables.

This directive determines which changes to reconcile between the child version (i.e., the version specified by the connection-related directive **VERSION_NAME**) and its parent version. Conflicts must be resolved manually using ESRI ArcGIS. Valid values are **INSERTS**, **UPDATES**, **DELETES**, and **ALL**. More than one value can be specified as long as each value is separated by a space. A post of the child version to its parent will be automatically performed when **ALL** is specified, or when **INSERTS**, **UPDATES**, and **DELETES** are all specified. Upon successfully posting the child version to its parent, whether or not the child version is deleted is dependent upon the value of the **DELETE_CHILD_AFTER_RECONCILE_AND_POST** directive. The directive is also used to determine whether the child version is deleted when it is identical to its parent version, in which case no reconciliation or posting is needed.

The post will be performed automatically when **ALL** or **INSERTS**, **UPDATES**, and **DELETES** is specified. If an error occurs during the post phase, then all changes made during both the reconcile and post phases will be rolled back.

Value: Any combination of **INSERTS**, **UPDATES**, **DELETES**, and **ALL**. Each value must be separated by a space.

- **INSERTS** – features inserted in the parent version
- **UPDATES** – features updated in the parent version
- **DELETES** – features deleted in the parent version
- **ALL** – encompasses **INSERTS**, **UPDATES**, and **DELETES**.

Example:

In the example below, all the updates and deletes made to the child version will be reconciled with the parent version. If a conflict occurs, then none of the reconciled changes will be saved. If **INSERTS** was also specified, then a post back to the parent version would also occur, but since neither it nor **ALL** is specified, only a reconciliation will be performed.

```
SDE30_RECONCILE_AND_POST UPDATES DELETES
```

Workbench Parameter: *Reconcile and Post*

TABLES_TO_RECONCILE

Required/Optional: *Optional*

Note: Valid only for vector features. Not supported for raster tables.

This optional statement specifies a list of tables (separated by spaces) which should be reconciled. Using this directive, it is possible to use an ArcSDE writer just to reconcile and post changes (i.e., not to write features). It is also possible to specify additional tables to reconcile that were not written to during the current translation. This directive only gets used if the directive **RECONCILE_AND_POST** is specified. If no tables are specified (and **RECONCILE_AND_POST** is specified), then only the tables written to during the translation will be reconciled. If no tables were written to during the translation, then no tables will be reconciled.

Value: <[table name]*> separated by spaces. If a table is owned by a different user, then the table name must be prefixed by the owner.

Example:

```
SDE30_TABLES_TO_RECONCILE countries rivers cities
```

Workbench Parameter: *Tables to Reconcile*

DELETE_CHILD_AFTER_RECONCILE_AND_POST

Required/Optional: *Required*

Note: Valid only for vector features. Not supported for raster tables.

This directive determines whether to delete the child version following a reconcile and post, including the case where the child and parent version are identical. A value of 'YES' will delete the child version, while a value of 'NO' will leave it intact. The default value is 'YES'.

Value: *YES or NO.*

Example:

In the example below, the child version will not be deleted after the reconcile and post operation completes.

```
SDE30_DELETE_CHILD_AFTER_RECONCILE_AND_POST NO
```

Workbench Parameter: *Delete child state following reconcile and post*

TRANSACTION

Required/Optional: *Optional*

This statement instructs the SDE writer module to use transactions when loading data into the SDE. The SDE writer does not write any features to the SDE until the feature is reached that belongs to <last successful transaction> + 1. Specifying a value of 0 causes the SDE writer to use transactions and to write every feature to the SDE. Normally, the value specified is zero – a non-zero value is only specified when a data load operation is being rerun.

If the **SDE30_TRANSACTION** statement is not specified, then transactions are not used during the data load operation.

Value: <last successful transaction>

The transaction number of the last successful transaction. When loading data for the first time, set this value to 0.

Example:

```
SDE30_TRANSACTION 0
```

Workbench Parameter: *Last Successful Transaction*

TRANSACTION_INTERVAL

Required/Optional: *Optional*

This statement informs FME about the number of features to be placed in each transaction before a transaction is committed to the database. When set to **VARIABLE** the SDE writer checks each feature for the **fme_db_transaction** attribute, for which there are 4 valid values:

- COMMIT_BEFORE - The current transaction is committed before writing the feature.
- COMMIT_AFTER - The current transaction is committed immediately after writing the feature.
- ROLLBACK_AFTER - The current transaction is rolled back immediately after writing the feature.
- IGNORE - The feature is written and no transaction handling occurs.

When the attribute is not found on the feature, then a value of **IGNORE** is assumed.

If the **SDE30_TRANSACTION_INTERVAL** statement is not specified, then a value of 100 is used as the transaction interval.

Value: <transaction_interval>

Either the number of features in a single transaction, or the value **VARIABLE**.

Default value: 100

WARNING: If the SDE30_TRANSACTION statement is not specified, then transactions are not used during the data load operation, even if the SDE30_TRANSACTION_INTERVAL is specified.

Example:

```
SDE30_TRANSACTION_INTERVAL 50
```

Workbench Parameter: *Features to Write Per Transaction*

STRICT_LOAD

Required/Optional: *Optional*

Note: Valid only for vector features. Not supported for raster tables.

This statement instructs FME to be very strict when loading spatial data from a feature into the SDE. When FME encounters a feature whose geometry cannot be converted into an SDE shape allowed by the layer (vector spatial column) for which it is destined, FME terminates the data load, logs the feature, and aborts the current transaction. In comparison to the **CONTINUE_TRANSLATION_BAD_DATA** directive, **STRICT_LOAD** is very limited in the type of errors that it can ignore. Use **CONTINUE_TRANSLATION_BAD_DATA** when it is desirable to ignore the majority of errors that may occur during a data load.

Value: YES | NO

Default Value: NO

Example:

```
SDE30_STRICT_LOAD YES
```

Workbench Parameter: *Strict SDE Load*

FORCE_IN_AGGREGATES

Required/Optional: *Optional*

Note: Valid only for vector features. Not supported for raster tables.

This statement instructs FME to make an extra effort to store multi-part polygon shapes (aggregates of polygons) into the SDE. When placed in this mode, the FME breaks apart aggregates that, according to the SDE, contain both polygons and lines, and attempt to store them as two feature aggregates. This is designed to assist with the loading of multi-part polygonal data in which some of the polygons are non-compliant with SDE's definition of a polygon.

To use this mode effectively, you must ensure that any polygonal layer (vector spatial column) for which this is applicable also allows for the storage of multi-part lines.

Value: YES | NO

Default Value: NO

Workbench Parameter: *Force In Aggregates*

DEFAULT_Z_VALUE

Required/Optional: *Optional*

Note: Valid only for vector features. Not supported for raster tables.

The value to be used for the z coordinate when a 2D feature is forced to become 3D because the layer (vector spatial column) is defined as being 3D. The z value specified for this directive must be larger than the z origin.

Value: *any real number*

Default Value: 0

Example:

```
SDE30_DEFAULT_Z_VALUE 52.3
```

Workbench Parameter: *Default Z Value*

LEAVE_LAYER_EXTENTS

Required/Optional: *Optional*

Note: Valid only for vector features. Not supported for raster tables.

By default, FME updates the layer (vector spatial column) extents when loading data into the SDE. This directive tells the FME not to perform this processing, thereby leaving the layer extent untouched.

Value: YES | NO

Default Value: NO

Example:

```
SDE30_LEAVE_LAYER_EXTENTS NO
```

Workbench Parameter: *Leave Layer Extents*

SPLIT_DONUTS

Required/Optional: *Optional*

Note: Valid only for vector features. Not supported for raster tables.

This directive is used when donut polygons are not to be stored as donuts, but rather simple polygons. When set to YES donut polygons are not stored in the SDE.

Value: YES | NO

Default Value: NO

Example:

```
SDE30_SPLIT_DONUTS NO
```

Workbench Parameter: *Store Donuts as Polygons*

CONTINUE_TRANSLATION_BAD_DATA

Required/Optional: *Optional*

This statement instructs the SDE writer to continue a translation even when an error occurs while attempting to load the data (the error may or may not be related to the data). A warning message will be output for each feature that could not be written to the SDE. The warning message will explain what went wrong. This directive is useful when trying to load bad data. Many more errors can be ignored using this directive than by using **STRICT_LOAD**. The **STRICT_LOAD** directive pertains only to converting geometry (from a feature) into an SDE shape to be written to a layer (vector spatial column).

When set to **ROLLBACK_THEN_CONTINUE**, if a feature fails to be written then the current transaction will be rolled back rather than committed when it comes time to commit the transaction. This means that none of the features in the rolled back transaction will be written to SDE. The translation will continue regardless of the error encountered. Transactions must be used when this value is specified. When used in conjunction with the **REJECTED_PIPELINE_DIRECTORY** keyword, if a feature returned from a pipeline fails to be written then the current transaction will be rolled back instead of committed.

If set to **YES** and transactions are being used, all transactions will be committed; however, failed features will not be written to SDE.

Value: YES | NO | ROLLBACK_THEN_CONTINUE

Default Value: NO

Example:

```
SDE30_CONTINUE_TRANSLATION_BAD_DATA ROLLBACK_THEN_CONTINUE
```

Workbench Parameter: *Skip Bad Data*

REJECTED_PIPELINE_DIRECTORY

Required/Optional: *Optional*

This statement instructs the FME where to find the pipeline file(s) to be used. A pipeline is used when a failure occurs in writing a feature. When this statement is specified and there was an error in writing a feature, the writer first attempts to open a pipeline specific to the current table. The writer looks for a file called **<tableName>_pipeline.fmi** in the directory specified by this statement. If the file is not found, then the writer looks for a default pipeline called **default_pipeline.fmi** in the same directory. If neither of these files are found, then the translation is stopped.

If a pipeline file is found, then an FME pipeline is created using all the factories from the file. The pipeline can do almost anything a regular FME pipeline can do. However, only the first feature from the pipeline is retrieved. If the pipeline does not return any feature, then the writer does not insert into SDE the row that corresponds to the feature. At the present time, the feature is recorded as written in the statistics portion of the FME log, whether or not it was actually inserted into SDE.

If this directive is used with the **CONTINUE_TRANSLATION_BAD_DATA** directive set to **YES** or **ROLLBACK_THEN_CONTINUE**, and a feature is returned from the pipeline, then if the returned feature causes an error while being written to the SDE it will not cause the translation to stop. Rather a warning message, explaining why the feature couldn't be written, will be logged and the translation will continue. If **ROLLBACK_THEN_CONTINUE** was specified, the current transaction will be rolled back instead of committed when it comes time to commit the transaction.

If this statement is not specified, then no pipeline will be created by the writer for features rejected by SDE. A pipeline is only created if this statement is specified and a failure occurs in writing a feature

Value: *The absolute or relative path of a directory containing pipeline files. If a relative path and the command line FME are used, then the path is relative to the location where FME is called from. If a relative path and the Universal Translator are used, then the path is relative to the location of the mapping file. If the path contains spaces in it, the path should be double-quoted.*

Example:

```
SDE30_REJECTED_PIPELINE_DIRECTORY c:\sde\pipelines
```

Workbench Parameter: *Rejected Pipeline Directory*

WRITER_MODE

Required/Optional: *Optional*

Note: For more information on this directive, see the chapter *Database Writer Mode*.

This statement instructs the FME as to the type of mode in which it is to operate. If the value specified is **INSERT**, then the features being written to the database are not checked to see if duplicate key values are in the database. This is useful when loading new data into the SDE.

When the **WRITER_MODE** directive is set to **UPDATE**, the writer will check to see if the attribute **fme_db_operation** exists on the feature. If the attribute is set to **INSERT**, the feature will be inserted; if the value is **UPDATE**, the feature will be updated; and if the value is **DELETE**, the feature will be deleted. If the attribute is set to any other value, the translation will be aborted and an error message logged. This functionality is designed to be used on tables that have an ArcSDE-maintained column, although it will also work on tables containing a spatial column.

If the writer mode is **UPDATE** and the attribute **fme_db_operation** is set to **UPDATE** or is not found on the feature, then the configuration parameter **SDE_UPDATE_FIELDS** (specified on a **SDE30_DEF** line) can be used to identify which features to update (it is never used for deleting features). It will only be needed if the table does not contain an ArcSDE-maintained (object ID) column and the table does not have a spatial column; however, if it is specified, then it will get used. If the configuration parameter **SDE_UPDATE_FIELDS** is specified, then the selected columns will make up a key. To avoid updating multiple features all at once, the user is responsible for ensuring that the specified key uniquely identifies a single feature within the SDE database. By using the RDBMS indices appropriately, the user should also ensure that a table scan of the underlying database will not result from each feature update operation.

If the **SDE30_WRITER_MODE** statement is not specified, then **INSERT** mode is used.

Writing a Raster Map

When writing a raster map to a table in SDE, the writer mode functions in a slightly different manner. **INSERT** mode inserts the raster into the table, overwriting any data that already pre-existed. **UPDATE** mode specifies that the writer is to mosaic the raster data to the pre-existing data, thereby updating the single row in the table. Raster catalogs may be inserted and updated to in a similar manner as vector data.

In order to perform a successful update, several conditions must be met by all the raster data that is to be mosaicked: the coordinate systems must be the same, the pixel depth must be the same, and the raster data itself must be either palette colored or continuous (they cannot be mixed). There is also a requirement for cell size and alignment to be the same, but the SDE writer will correct for these automatically, so they need not be altered. There is also no need to alter the writer mode when mosaicking, since the writer will automatically detect and correct this based on whether or not the data is pre-existing, in order not to overwrite it. The only way to overwrite existing raster data in a raster map is to set either the **SDE_DROP_TABLE** or the **SDE_TRUNCATE_TABLE** flag to **YES**.

Value: *INSERT | UPDATE | DELETE*

Default Value: *INSERT*

Example:

```
SDE30_WRITER_MODE INSERT
```

Workbench Parameter: *Writer Mode*

BUFFERED_WRITES

Required/Optional: *Optional*

When specified, the buffered writing of the SDE is used which dramatically decreases the load time of data into the SDE.

Value: *YES | NO*

Default Value: *NO*

Example:

`SDE30_BUFFERED_WRITES YES`

Workbench Parameter: *Use Buffered Writes*

MAX_OPEN_TABLES

Required/Optional: *Optional*

Specifies the maximum number of streams that can be open simultaneously. Each stream writes to a particular table and so this directive determines the maximum number of tables that can be open and written to simultaneously. If this directive is not specified, or is given the value 0, then the SDE writer will set the maximum number of streams open simultaneously to 4 less than the number specified by `MAXSTREAMS` in `giomgr.defs`.

Value: *The maximum number of tables that can be open simultaneously*

Example:

`SDE30_MAX_OPEN_TABLES 30`

Workbench Parameter: *Maximum Num of Open Tables*

ADD_LAYERS_TO_EXISTING_TABLES

Required/Optional: *Optional*

Specifies whether an existing business table within ArcSDE should have a layer (vector spatial column) added to it. To be eligible for this schema modification, the first feature written to the table must contain vector geometry.

Value: *YES | NO*

Default Value: *YES*

Example:

`SDE30_ADD_LAYERS_TO_EXISTING_TABLES NO`

Workbench Parameter: *Add Layers to Existing Tables*

INTEGER_OVERRIDE_DEFINITION

Required/Optional: *Optional*

Specifies a definition to use for all integer column types when creating new tables. Any of the allowed FME attribute types for the ArcSDE Writer can be used as values for this directive. Additionally, the following can also be used: `number(<width>)` or `number(<width>, <decimal>)`. By default, this directive is not set, and so integer columns are stored using the C language long integer data type.

Values:

- *any allowed FME attribute types for the ArcSDE writer*
- *number(<width>) or number(<width>, <decimal>)*

Example:

In the example here, the ArcSDE database will use `char(30)` instead of `integer` as the type for all integer columns.

`SDE30_INTEGER_OVERRIDE_DEFINITION char(30)`

Workbench Parameter: *Integer Definition*

Writing to an Enterprise Geodatabase

The SDE Writer is capable of writing features to an existing feature class that is located on an Enterprise Geodatabase. However, portions of the `SDE30_DEF` line must match the definition of the existing feature class. The following configuration parameters must match the existing definition:

1. SDE_LAYER
2. SDE_DIMENSION
3. SDE_CAD
4. SDE_MEASURED
5. SDE_ANNOTATED

All other configuration parameters do not need to be correct, and they will be ignored. Additionally, the non-spatial columns in the feature class do not need to be defined on the **SDE30_DEF** line.

FME Raster Features

FME raster features represent raster data and use several concepts that are unlike those used in the handling of vector data. See *About FME Rasters*.

SDE supports rasters with an arbitrary number of bands, provided all bands are the same data type and no band has a palette. SDE also supports rasters with a single band that has a palette.

SDE Table Representation

When reading from SDE, it is not necessary that the source tables be defined. This is also true when writing to existing tables. However, if the SDE writer is going to create the tables, then definitions must be supplied. This is true even when the table already exists but the **SDE_DROP_TABLE** parameter has been set to Yes. Within FME mapping files, SDE tables are defined using the **<writerKeyword>_DEF** statement, whereas within Workbench they are defined by adding destination feature types.

When creating new tables, it is important to understand that the decision for which type of table to create is not based on the DEF line itself but rather the first feature written to the table. This means that if the first feature contains no geometry, then a business table (no spatial columns) will be created; if the first feature contains vector geometry, then a feature class (business table + layer) will be created; and if the first feature contains raster geometry, either a raster map or raster catalog will be created. The purpose of the DEF line is to specify what the table looks like.

If the table already exists in the database (and the user is not deleting it using the **SDE_DROP_TABLE** parameter) then its schema will not be altered. The only exception to this is when the first feature, within a translation, written to a pre-existing business table contains vector geometry and the directive **ADD_LAYERS_TO_EXISTING_TABLES** is set to Yes or is not specified. In this case a layer will be added to the business table. The addition of the layer turns the business table into a feature class. If the **SDE_RASTERMAP** writer is used to write to an existing raster catalog, or the **SDE_RASTERCATALOG** writer is used to write to an existing rastermap, the translation will fail.

To define a simple table with no spatial or raster column using FME, the definition is in this form:

```
<writerKeyword>_DEF <tableName> \
    [<columnName> <columndef>]*
```

A more general format of a table definition – in which a spatial column, along with attribute indices can be defined – is given here.

```
<writerKeyword>_DEF <tableName> \
    [<columnName> <columndef>] * \
    [
        [SDE_INDEX <indexName> \
SDE_INDEX_CONFIG <configKeyword> \
            SDE_COLUMN_NAME <columnName>[,<columnName>]* \
            SDE_UNIQUE <TRUE|FALSE|YES|NO> \
            SDE_SORT_ORDER ASCEND|DESCEND
        ] * \
        [SDE_UPDATE_FIELDS <columnName> [,<columnName>]* \
        [SDE_STORAGE_TYPE <SDE_BINARY|WKB|SQL|NORMALIZED>]
        SDE_LAYER <spatialColumnName> \
        [SDE_COORD_SYS_ID <coordSysID#>] \
        [SDE_COORD_SYS_DESCRIPTION <description>] \
```

```

[SDE_PRECISION <32 | 64>] \
SDE_GRID{0} <grid0size> \
[SDE_GRID{1} <grid1size>] \
[SDE_GRID{2} <grid2size>] \
SDE_DIMENSION < 2 | 3 > \
[SDE_CONFIG_KEYWORD <configkeyword>] \
SDE_MEASURED < Yes | No > \
SDE_ANNOTATED < Yes | No > \
SDE_AREA < Yes | No > \
SDE_LINE < Yes | No > \
SDE_POINT < Yes | No > \
SDE_SIMPLE_LINE < Yes | No > \
SDE_NIL < Yes | No > \
SDE_MULTIPART < Yes | No > \
SDE_CAD < Yes | No > \
SDE_DROP_TABLE < Yes | No > \
SDE_TRUNCATE_TABLE < Yes | No > \
SDE_XORIGIN <minimum_x> \
SDE_YORIGIN <minimum_y> \
SDE_SCALE <scale> \
SDE_ZORIGIN <minimum_y> \
SDE_ZSCALE <scale> \
SDE_MEASURED_ORIGIN <minimum_y> \
SDE_MEASURED_SCALE <scale> \
SDE_TOLERANCE <tolerance> \
SDE_MEASURED_TOLERANCE <tolerance> \
SDE_ZTOLERANCE <tolerance> \
[SDE_DESCRIPTION <layer description>] \
[SDE_MINIMUM_FID <minimumFidNumber>]
]

```

Another form of the general format of a table definition includes the definition of a raster column. An example is given here for a raster map.

Note: During creation of a new rastermap table, the writer will create a reserved ArcSDE column called NAME. This additional column will be created and populated with the value ESRI_SDERASTERDATASET. For raster catalogs, this column is optional, and populated by default with the value of the fme_basename attribute for each row in the table.

```

<writerKeyword>_DEF <tableName> \
  [<columnName> <columnDef>] * \
  [SDE_COORD_SYS_ID <coordSysID#>] \
  [SDE_COORD_SYS_DESCRIPTION <description>] \
  [SDE_CONFIG_KEYWORD <configkeyword>] \
  SDE_RASTER <rasterColumnName> \
  [SDE_COMPRESS_TYPE < NONE | LZ77 | JPEG | JPEG2000>] \
  [SDE_PYRAMID_INTERPOLATION< NONE | NEAREST_NEIGHBOR |
    BILINEAR | BICUBIC >] \
  [SDE_PYRAMID_LEVEL_TYPE < NONE | AUTO | CUSTOM >] \
  [SDE_PYRAMID_MAX_LEVEL <maxLevel>] \
  [SDE_RASTER_STATS_TYPE < NONE | AUTO >] \
  [SDE_DESCRIPTION <raster description>] \
  [SDE_DROP_TABLE < Yes | No > \] \
  [SDE_TRUNCATE_TABLE < Yes | No > \] \
  [SDE_COMPRESS_COLORMAP < Yes | No > \] \
  [SDE_RASTER_MOSAIC_MODE <NONE | MERGE | DELETE>]

```

The table definition for a raster catalog includes additional parameters for creating the spatial (footprint) column.

Note: If a table definition is given for a table that does not yet exist and the only column defined in the definition is the spatial or raster column, then the writer will create an object ID column maintained by ArcSDE called OBJECTID. This additional column will be created because ArcSDE does not allow tables to contain only a spatial/raster column.

```

<writerKeyword>_DEF <tableName> \
  [<columnName> <columndef>] * \
    [SDE_COORD_SYS_ID <coordSysID#>] \
    [SDE_COORD_SYS_DESCRIPTION <description>] \
    [SDE_CONFIG_KEYWORD <configkeyword>] \
      SDE_RASTER <rasterColumnName> \
      [SDE_COMPRESS_TYPE < NONE | LZ77 | JPEG | JPEG2000>] \
      [SDE_PYRAMID_INTERPOLATION< NONE | NEAREST_NEIGHBOR |
        BILINEAR | BICUBIC >] \
      [SDE_PYRAMID_LEVEL_TYPE < NONE | AUTO | CUSTOM >] \
      [SDE_PYRAMID_MAX_LEVEL <maxLevel>] \
      [SDE_RASTER_STATS_TYPE < NONE | AUTO >] \
      [SDE_DESCRIPTION <raster description>] \
      [SDE_MINIMUM_FID <minimumFidNumber>] \
      [SDE_DROP_TABLE < Yes | No > \] \
      [SDE_TRUNCATE_TABLE < Yes | No > \] \
      [SDE_XORIGIN <minimum_x>] \
      [SDE_YORIGIN <minimum_y>] \
      [SDE_SCALE <scale>]

```

<tableName>

This specifies the name of the SDE table being defined by the **SDE30_DEF** statement. The name must conform to the conventions and restrictions of the underlying RDBMS database.

The following example shows the first portion of the definition for a table named roads.

```
SDE30_DEF roads . . .
```

Attribute Definitions

This section of the **SDE30_DEF** statement defines the attributes for a table. A table must have at least one attribute.

- The **<attribute name>** specified within the FME mapping file must obey the following rules:
 - Attribute Names must be in uppercase.
 - Attribute Names must obey all length and character restrictions of the SDE.
- The **<attribute definition>** defines the type and optionality of the attribute, and has the following form:

```
<attribute type>,(optional|required)
```

- The supported attribute types are listed in the following table.

FME Attribute Type
smallint
integer
float
double
char(n)
blob
date
guid

The directive **optional** or **required** immediately follows the attribute type and indicates if the attribute is required. If nothing is specified, then the value defaults to optional.

The following example creates a required attribute called **NUMOFLANES** which is an **integer** type.

NUMOFLANES integer, required

smallint

This type is used to represent 16-bit integer values.

integer

This type is used to represent 32-bit integer values.

float

This type is used to represent 32-bit float values.

double

This type is used to represent 64-bit integer values

char(n)

This type is used to represent character values with a length not exceeding **n** characters. With SDE 9.2 and later, the **char(n)** column maps to a unicode column encoded in UTF-16 if the DBTUNE parameter UNICODE_STRING is set to TRUE or is not present.

blob

This is used to store arbitrary binary data in the SDE. See **@Reformat** and **@File** in the *FME Functions, Factories and Transformers* manual for a description of the **@Reformat** and **@File** functions, and for information on how to load and retrieve data into a blob attribute.

With the use of blob types coupled with **@System** and **@File** it is possible to store any arbitrary data with an SDE feature. If a feature has sound, video, images, or documents, or all of the above, they can be zipped up to form a compact package using **@System**. Next, **@File** can be used to load the zip file into an attribute of the SDE. The contents are then loaded directly in the database for later retrieval.

date

This is used to store and retrieve date information to the SDE.

When a date field is read by the SDE, two attributes are set in the FME feature. The first attribute has the name of the database column, and its value is of the form **YYYYMMDD**. This is compatible with all other FME dates.

The second attribute has a suffix of **.full** and is of the form **YYYYMMDDHHMMSS**. It specifies the date and the time, with the time portion specified using the 24-hour clock.

For example, if a date field called **UPDATE_DATE** is read, the following attributes will be set in the retrieved FME feature:

```
UPDATE_DATE = '19980820'  
UPDATE_DATE.full = '19980820201543'
```

When writing to the SDE, the writer looks for both attributes. Either may be in the form **YYYYMMDD** or **YYYYMMDDHHMMSS**. If both attributes are specified, then the value specified in **UPDATE_DATE.full** is used.

guid

This type is used to represent Globally Unique Identifiers (GUIDs), which are stored as text strings of length 36 within FME. The format of a GUID is 8 hexadecimal digits followed by a hyphen, then three groups of 4 hexadecimal digits, each followed by a hyphen, and then 12 hexadecimal digits. Note that { and } braces found at the beginning and end of the GUID are removed by the Reader and added on by the Writer if not present.

When writing to a required GUID field, the Writer will automatically generate a GUID if no value is supplied for it on the feature.

Example:

```
414EF035-DCDF-4DAD-96DA-E86C0DA661B2
```

SDE_INDEX <indexName>

This section of the **SDE30_DEF** line defines one or more non-spatial index columns. Non-spatial column indices are used to increase the performance of the non-spatial component of queries.

Each index definition is identified by a unique name and has the components shown in the following table.

Parameter	Contents
SDE_INDEX_CONFIG	The configuration keyword that describes the storage characteristics of the index tables. Example value is DEFAULTS .
SDE_COLUMN_NAME	A comma-separated list of the columns that make up the index.
SDE_UNIQUE	A flag to indicate if the index values are unique or not. Acceptable values are N or Y .
SDE_SORT_ORDER	The sort order of the index. Indicates whether the index returns the records in ascending or descending order. Acceptable values are ASCEND or DESCEND .

The following example defines an index called **countryCapital**. The index is ascending and is not unique. The index is built on the columns **COUNTRY** and **CAPITAL**. The index table storage characteristics are taken from the **dbtune.sde** file entries defined by **DEFAULTS**.

```
SDE_INDEX countryCapital \  
  SDE_INDEX_CONFIG DEFAULTS \  
  SDE_COLUMN_NAME COUNTRY,CAPITAL \  
  SDE_UNIQUE N \  
  SDE_SORT_ORDER ASCEND \  
  \
```

Configuration Parameters

There are a number of configuration parameters in the **SDE30_DEF** line that are used to define spatial column characteristics. They are described in the following table.

Note: The values populated in the settings box set values for configuration parameters.

Parameter	Contents
SDE_LAYER	This defines the name of the spatial column within the table being defined. The spatial column is the column that contains the geometry of the feature. The following example gives the spatial column a name of SHAPE . SDE_LAYER SHAPE It is recommended that SHAPE be used as the name.
SDE_PRECISION	This optional field specifies whether to set the precision to 32-bit or 64-bit. If not specified in a workspace/mapping file, then it will be set to 32-bit; however, all newly created workspaces/mapping files specify this field and set it to 64-bit. When writing to ArcSDE 8.x or older, 32-bit precision

Parameter	Contents
	will automatically get used since 64-bit support was added in ArcSDE 9.0.
SDE_COORD_SYS_ID	<p>This optional field specifies the coordinate system of the spatial column. This is only used during the initial creation of a spatial column. The value is an integer value that corresponds to one of the predefined coordinate systems specified in ESRI's <i>Projection Engine</i> documentation which is shipped with every SDE 30.</p> <p>Either SDE_COORD_SYS_ID or SDE_COORD_SYS_DESCRIPTION can be specified, but not both.</p> <p>If it is not specified, then the coordinate system will be taken from the first feature written to each table.</p>
SDE_COORD_SYS_DESCRIPTION	<p>This optional field specifies the coordinate system of the spatial column. This approach enables the entire projection to be specified using a description as defined in the ESRI Projection Engine documentation that is shipped with every SDE 30.</p> <p>As mentioned above, you specify either SDE_COORD_SYS_ID or SDE_COORD_SYS_DESCRIPTION, but not both.</p> <p>If it is not specified, then the coordinate system will be taken from the first feature written to each table.</p>
SDE_GRID{0}	<p>This is specified as part of a spatial column definition. It gives the size of the spatial index in the coordinate system of the layer (vector spatial column).</p> <p>When set to -1, a spatial index will not be created. This is useful when a valid spatial index is not known. It also improves the speed of writing features.</p> <p>After the translation, the "Calculate Default Spatial Grid Index" tool (from ArcToolbox > Data Management Tools > Feature Class) can be used to calculate a valid spatial index. When Grid{0} is set to -1, level 2 & 3 grids do not get built even though the values for these levels get stored with the layer information.</p> <p>The following example defines the grid size of 200:</p> <p>SDE_GRID{0} 200</p>
SDE_GRID{1}	<p>This optional parameter defines the level 2 grid element size. This is not needed for the majority of spatial columns. If specified, this must be at least 3 times the size of SDE_Grid{0}.</p> <p>If it is not desired, then either the value should not be specified or it should be given a value of 0.</p> <p>The following example defines a grid size of 600 for level 1</p>

Parameter	Contents
	grid: <code>SDE_GRID{1} 600</code>
SDE_GRID{2}	<p>This optional parameter defines the level 3 grid element size. This level grid is rarely required. If specified, this must be at least 3 times the <code>SDE_GRID{1}</code>.</p> <p>If it is not desired, then either the value should not be specified or it should be given a value of <code>0</code>.</p> <p>The following example defines a grid size of 4000 for the level 2 grid:</p> <code>SDE_GRID{2} 4000</code>
SDE_DIMENSION	<p>The SDE requires that all features within a feature class have the same dimension. This parameter defines the dimension of the layer (vector spatial column). Currently, the dimension can be either 2 or 3.</p> <p>The example below defines the layer to have a dimension of 2:</p> <code>SDE_DIMENSION 2</code>
SDE_UPDATE_FIELDS	<p>The list of field names that are used by the SDE Writer when it is operating in <code>UPDATE</code> mode. If the table is either registered as multi-versioned or contains a spatial column, then this configuration parameter is optional. In general, this should identify a unique feature but can also be used to update multiple features if desired.</p> <p>The following example sets the update fields to be country and capital:</p> <code>SDE_UPDATE_FIELDS COUNTRY,CAPITAL</code>
SDE_XORIGIN	<p>The minimum <code>x</code> value of the spatial column being defined. No coordinate values can be less than the value specified here.</p> <p>For raster catalogs, this value should be calculated from the lower left corner of the lower-leftmost raster to be added to the catalog. If the value is unspecified, the footprint column will not be created; however, the footprint column will automatically be created when the table is registered with Geodatabase.</p> <p>The example below defines the lower extent of a spatial column to be -180:</p> <code>SDE_XORIGIN -180</code>
SDE_YORIGIN	<p>The minimum <code>y</code> value of the spatial column being defined. No coordinate values can be less than the value specified here.</p> <p>For raster catalogs, this value should be calculated from</p>

Parameter	Contents
	<p>the lower left corner of the lower-leftmost raster to be added to the catalog. If the value is unspecified, the footprint column will not be created; however, the footprint column will automatically be created when the table is registered with Geodatabase.</p> <p>The example below defines the lower extent of a spatial column to be -90:</p> <p><code>SDE_YORIGIN -90</code></p>
SDE_SCALE	<p>The scale of the spatial column. This defines the number of units per user coordinate stored within the spatial column. For raster catalogs, if this value is unspecified, the footprint column will not be created; however, the footprint column will automatically be created when the table is registered with Geodatabase.</p> <p>The example below defines the scale to be 100:</p> <p><code>SDE_SCALE 100</code></p> <p>This is equivalent to 2 decimal places to the right of the decimal in user coordinates.</p>
SDE_ZORIGIN	<p>The minimum z value stored within the spatial column. The example below defines the minimum z value to be 0:</p> <p><code>SDE_ZORIGIN 0</code></p>
SDE_ZSCALE	<p>The scale of the spatial column z coordinate. This defines the number of units per user coordinate stored within the spatial column.</p> <p>The example below defines the z scale to be 100:</p> <p><code>SDE_ZSCALE 100</code></p> <p>This is equivalent to 2 decimal places to the right of the decimal in user coordinates.</p>
SDE_MEASURED_ORIGIN	<p>The minimum measure value that is stored within the spatial column.</p> <p>The example below defines the minimum measure value to be 0:</p> <p><code>SDE_MEASURED_ORIGIN 0</code></p>
SDE_MEASURED_SCALE	<p>The scale of the spatial column measured value. This defines the number of units per user coordinate that are stored within the spatial column.</p> <p>The example below defines the measured scale to be 100:</p> <p><code>SDE_MEASURED_SCALE 100</code></p> <p>This is equivalent to 2 decimal places to the right of the decimal in user coordinates.</p>

Parameter	Contents
SDE_DESCRIPTION	<p>The description of the spatial column, which is just free text.</p> <p>SDE_DESCRIPTION Roadwork</p>
SDE_MINIMUM_FID	<p>The minimum feature ID assigned to shapes stored in the layer (vector spatial column). When the SDE stores shapes in a table, each shape is given an ID number that is unique throughout the table. If not specified, then the Feature ID starts at 1 for each spatial column.</p> <p>The only time this value needs to be specified is when tricks are being performed using the underlying RDBMS in which you want the Feature ID to be unique through a set of tables rather than throughout a single table.</p> <p>The example below results in the feature IDs starting at 100000 for the table upon which the statement is specified:</p> <p>SDE_MINIMUM_FID 100000</p>
SDE_CONFIG_KEYWORD	<p>The SDE configuration keyword specifies the storage parameters for the layer (vector spatial column) or raster column.</p> <p>Note that in releases before ArcGIS 9.3, the configuration keyword specified must be present in the \$SD-EHOME/etc/dbtune.sde file.</p> <p>If not specified, the keyword DEFAULTS will be used.</p> <p>For more information, search <i>parameter name-configuration string pairs</i> in ESRI ArcGIS Server help files.</p> <p>The example below uses a configuration keyword of TEST:</p> <p>SDE_CONFIG_KEYWORD TEST</p>
SDE_MEASURED	<p>Y – The spatial column allows measures to be specified on each coordinate of the features.</p> <p>N – The spatial column does not allow measures.</p>
SDE_ANNOTATED	<p>Y – The spatial column allows annotation to be specified.</p> <p>N – The spatial column does not allow annotations.</p>
SDE_AREA	<p>Y – The spatial column allows area features to be stored.</p> <p>N – The spatial column does not allow area features to be stored.</p>
SDE_LINE	<p>Y – The spatial column allows linear features to be stored. Line features are those linear features that may touch or cross over themselves.</p> <p>N – The spatial column does not allow linear features to be stored.</p>

Parameter	Contents
SDE_POINT	<p>Y – The spatial column allows point features to be stored.</p> <p>N – The spatial column does not allow points.</p>
SDE_SIMPLE_LINE	<p>Y – The spatial column allows simple lines to be stored. Simple lines are lines that do not touch or cross over themselves.</p> <p>N – The spatial column does not allow simple lines.</p>
SDE_NIL	<p>Y – The spatial column allows NIL features to be stored. NIL features are features that have a shape object with no coordinates.</p> <p>N – The spatial column does not allow NIL features.</p>
SDE_MULTIPART	<p>Y – The spatial column allows features that have multiple parts. Multi-part features must be homogeneous. That is, all parts must be either area, linear, or point within a single feature.</p> <p>N – The spatial column does not allow features which have multiple parts.</p>
SDE_CAD	<p>Y – The layer (vector spatial column) allows CAD data to be stored with it. This is for CAD client layers. FME is not capable of storing data in the CAD blob associated with the layer.</p> <p>N – The layer does not allow CAD data.</p>
SDE_STORAGE_TYPE	<p>SDE_BINARY – the feature geometry for the layer (vector spatial column) is stored in SDE binary mode. This is the default and the only type that is supported for SDE 3.x.</p> <p>WKB – the feature geometry for the layer is stored in SDE using the OGC Well Known Binary form. ArcSDE 8.x only.</p> <p>SQL – Stored as SQL or well known text format. ArcSDE 8.x only.</p> <p>NORMALIZED – Normalized format (used for Oracle Spatial Only). ArcSDE8.x only.</p>
SDE_DROP_TABLE	<p>Specifies that the SDE writer drop the table before writing, and create a new one. For raster tables, the associated raster column and band information tables will be dropped as well. If the table does not exist, it will be created when the data is written. The writer expects that the general table type (i.e. raster, feature class/vector, business/non-spatial) of the new table will be the same as the table being deleted, with the exception of business tables where it is possible to delete a business table but create a feature class.</p> <p>The following example sets the drop table flag to false.</p>

Parameter	Contents
	<p data-bbox="591 268 862 296"><code>SDE_DROP_TABLE NO</code></p> <p data-bbox="558 306 708 333">Default: NO</p> <p data-bbox="558 344 769 371">Values: YES NO</p>
SDE_TRUNCATE_TABLE	<p data-bbox="558 394 1344 569">Specifies that the SDE writer truncate the table before writing. For raster tables, the associated raster column and band information tables will be truncated as well. If the table does not exist, it will be created when the data is written.</p> <p data-bbox="558 579 1333 606">The following example sets the truncate table flag to false.</p> <p data-bbox="591 617 927 644"><code>SDE_TRUNCATE_TABLE NO</code></p> <p data-bbox="558 655 708 682">Default: NO</p> <p data-bbox="558 693 769 720">Values: YES NO</p>
SDE_TOLERANCE	<p data-bbox="558 741 1333 1167">The cluster tolerance of the XY values in the spatial column. This value represents an extremely small distance used to resolve inexact intersection locations of coordinates during clustering operations. The XY tolerance is the minimum distance allowed between XY coordinates before they are considered equal. It is used in clustering operations such as topology validation, buffer generation, polygon overlay and for some editing operations. Tolerance is only valid for ArcSDE 9.2 and newer tables, and is not used for raster data. If a tolerance value is not specified, a default value will be used based on a conversion of 0.001 meters in the unit of the source coordinate system.</p> <p data-bbox="558 1209 1305 1236">The example below defines the xy tolerance to be 0.001:</p> <p data-bbox="591 1247 894 1274"><code>SDE_TOLERANCE 0.001</code></p>
SDE_MEASURED_TOLERANCE	<p data-bbox="558 1297 1333 1724">The cluster tolerance of the measured values in the spatial column. This value represents an extremely small distance used to resolve inexact intersection locations of coordinates during clustering operations. The measured tolerance is the minimum distance allowed between M values before they are considered equal. It is used in clustering operations such as topology validation, buffer generation, polygon overlay and for some editing operations. Tolerance is only valid for ArcSDE 9.2 and newer tables, and is not used for raster data. If a tolerance value is not specified, a default value will be used based on a conversion of 0.001 meters in the unit of the source coordinate system.</p> <p data-bbox="558 1766 1313 1829">The example below defines the measured tolerance to be 0.001:</p> <p data-bbox="591 1839 1036 1866"><code>SDE_MEASURED_TOLERANCE 0.001</code></p>

Parameter	Contents
SDE_ZTOLERANCE	<p>The cluster tolerance of the Z values in the spatial column. This value represents an extremely small distance used to resolve inexact intersection locations of coordinates during clustering operations. The Z tolerance is the minimum distance allowed between Z values before they are considered equal. It is used in clustering operations such as topology validation, buffer generation, polygon overlay and for some editing operations. Tolerance is only valid for ArcSDE 9.2 and newer tables, and is not used for raster data. If a tolerance value is not specified, a default value will be used based on a conversion of 0.001 meters in the unit of the source coordinate system.</p> <p>The example below defines the Z tolerance to be 0.001: SDE_ZTOLERANCE 0.001</p>
SDE_RASTER	<p>This defines the name of the raster column within the table being defined. The raster column is the column that defines the geometry of the table as raster and contains the geometry of the features in the table. If a table definition has both a spatial and a raster column, the spatial column will be ignored.</p> <p>The following example gives the raster column a name of RASTER. SDE_RASTER RASTER</p> <p>It is recommended that RASTER be used as the name.</p>
SDE_COMPRESS_TYPE	<p>This defines the type of compression to for the raster table being defined.</p> <p>The following example gives the raster column a compression type of LZ77. SDE_COMPRESS_TYPE LZ77</p> <p>Default: NONE Values: NONE LZ77 JPEG JPEG2000</p> <p>Note: LZ77 is the only valid compression option for images with a colormap. Also, JPEG2000 compression is only available for servers running SDE version 9.0 or later, and on rasters with an 8-bit pixel depth and no colormap.</p>
SDE_PYRAMID_INTERPOLATION	<p>This defines the interpolation type of pyramid creation for the table being defined.</p> <p>The following example gives the raster column a pyramid interpolation type NEAREST_NEIGHBOR. SDE_PYRAMID_INTERPOLATION NEAREST_NEIGHBOR</p> <p>Default: NONE</p>

Parameter	Contents
	<p>Values: NONE NEAREST_NEIGHBOR BILINEAR BICUBIC</p> <p>Note: A value of NONE disables pyramid creation and disregards the other pyramid settings. Also note that nearest neighbor is the only valid pyramid setting for classified raster data.</p>
SDE_PYRAMID_LEVEL_TYPE	<p>This defines the way the maximum pyramid level is set for the raster table being defined.</p> <p>The following example gives the raster column sets a maximum pyramid level to be automatically calculated.</p> <p>SDE_PYRAMID_LEVEL_TYPE AUTO</p> <p>Default: NONE</p> <p>Values: NONE AUTO CUSTOM</p> <p>Note: A value of NONE disables pyramid creation and disregards the other pyramid settings.</p>
SDE_PYRAMID_MAX_LEVEL	<p>This defines the maximum pyramid level to be created for the table being defined.</p> <p>The following example gives the raster column sets a maximum pyramid level to be automatically calculated.</p> <p>SDE_PYRAMID_MAX_LEVEL AUTO</p> <p>Note: This setting is only used if the pyramid level type is set to the value CUSTOM.</p>
SDE_RASTER_STATS_TYPE	<p>This defines the type of statistics calculation for the raster table being defined.</p> <p>The following example gives the user automatic determination of a statistics calculation function.</p> <p>SDE_RASTER_STATS_TYPE AUTO</p> <p>Default: NONE</p> <p>Values: NONE AUTO</p> <p>Note: A value of NONE turns statistics calculation off for this table.</p>
SDE_RASTER_MOSAIC_MODE	<p>Specifies the mosaic mode that will be used when mosaicking data to an SDE rastermap. The default is MERGE. A value of NONE means that new data will completely replace existing raster data, and no mosaic is applied. MERGE causes the data to be mosaicked, replacing existing pixel values with new pixel values where they overlap, and leaving all other data untouched. The nodata values in the existing raster are not altered. DELETE mode does not mosaic any new data, but rather has the sole purpose of deleting data in the existing raster. This is accomplished by deleting pixel data where the new raster overlaps the existing raster and the value for that pixel location in the new</p>

Parameter	Contents
	<p>raster is nodata. MERGE and DELETE modes are ignored on insert, and are only valid for a mosaic operation.</p> <p>The following example sets the mosaic mode to merge.</p> <pre>SDE_RASTER_MOSAIC_MODE MERGE</pre> <p>Default: MERGE Values: NONE MERGE DELETE</p> <p>Note: The DELETE mode is only supported by ArcSDE version 9.0 and later.</p>
SDE_RASTER_COMPRESS_COLORMAP	<p>This is used with palette colored rasters written to a raster map in SDE. It specifies whether the colormap is to be compressed or untouched. Compressing a colormap will remove any invalid entries, and possibly make future mosaic operations to the same table faster and less likely to approximate colors.</p> <p>The following example sets the compress colormap flag to true:</p> <pre>SDE_RASTER_COMPRESS_COLORMAP YES</pre> <p>Default: YES Values: YES NO</p>

Example of a Feature Class Definition

A typical SDE30 definition looks like this:

```
SDE30_DEF WORLD \
  SDE_LAYER WORLD_GEOM \
  SDE_GRID{0} 1000000 \
  SDE_DIMENSION 2 \
  SDE_CONFIG_KEYWORD DEFAULTS \
  SDE_MEASURED Y \
  SDE_ANNOTATED Y \
  SDE_AREA Y \
  SDE_LINE Y \
  SDE_POINT Y \
  SDE_SIMPLE_LINE Y \
  SDE_NIL Y \
  SDE_MULTIPART Y \
  SDE_XORIGIN -19000000 \
  SDE_YORIGIN -19000000 \
  SDE_SCALE 10 \
  SDE_ZORIGIN 0 \
  SDE_ZSCALE 1 \
  SDE_MEASURED_ORIGIN 0 \
  SDE_MEASURED_SCALE 1 \
  SDE_TOLERANCE 0.0004 \
  SDE_MEASURED_TOLERANCE \
  SDE_ZTOLERANCE \
  SDE_INDEX countryCapital \
    SDE_INDEX_CONFIG DEFAULTS \
    SDE_COLUMN_NAME COUNTRY,CAPITAL \
    SDE_UNIQUE N \
    SDE_SORT_ORDER ASCEND \
  COUNTRY char(3) \
```

```
CAPITAL char(30) \  
SDE_UPDATE_FIELDS COUNTRY,CAPITAL
```

Example of a Raster Column Definition

A typical SDERASTERMAP definition looks like this:

```
SDERASTERMAP_DEF WORLD  
  SDE_DROP_TABLE Y \  
  SDE_TRUNCATE_TABLE N \  
  SDE_CONFIG_KEYWORD DEFAULTS \  
  SDE_RASTER RASTER \  
  SDE_COMPRESS_TYPE LZ77 \  
  SDE_PYRAMID_INTERPOLATION NONE \  
  SDE_PYRAMID_LEVEL_TYPE NONE \  
  SDE_PYRAMID_MAX_LEVEL 0 \  
  SDE_RASTER_STATS_TYPE AUTO \  
  SDE_RASTER_COMPRESS_COLORMAP Y \  
SDE_RASTER_MOSAIC_MODE MERGE
```

And an SDERASTERCATALOG definition might look like this:

```
SDERASTERCATALOG_DEF WORLD \  
  SDE_DROP_TABLE N \  
  SDE_TRUNCATE_TABLE N \  
  SDE_CONFIG_KEYWORD DEFAULTS \  
  SDE_RASTER RASTER \  
  SDE_COMPRESS_TYPE LZ77 \  
  SDE_PYRAMID_INTERPOLATION NONE \  
  SDE_PYRAMID_LEVEL_TYPE NONE \  
  SDE_PYRAMID_MAX_LEVEL 0 \  
  SDE_RASTER_STATS_TYPE AUTO \  
  SDE_RASTER_COMPRESS_COLORMAP Y \  
SDE_XORIGIN -400000 \  
  SDE_YORIGIN -400000 \  
  SDE_SCALE 1000000
```

Using Versioning with the SDE Reader, Writer, and QueryFactory

Database states will be created by FME only when updating/inserting/deleting from a versioned table/feature class. Therefore, only the SDE writer or an **SDE30QueryFactory** in **DELETE** or **UPDATE** mode is able to create a database state.

The SDE reader and the **SDE30QueryFactory** in **QUERY** mode will never create a database state. All changes made during a single translation to a specific version (on the same SDE), even if they were made by different **SDE30QueryFactories** or different SDE writers, are placed into one (and the same) child state. Versioning must be used when updating/inserting/deleting from a versioned table/feature class. If versioning is not used in these cases, than an "Insufficient permissions" error will be generated and the translation stopped.

Note: Versioning is not currently supported for raster data.

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Attribute Name	Contents
sde30_type	The type of geometric entity stored within the feature. The valid values are listed below: sde30_point

Attribute Name	Contents
	sde30_nil sde30_line sde30_area sde30_circle sde30_ellipse sde30_simple_line sde30_raster
sde30_measures	This is present for features that have measures when reading. To write measures, you simply build this list with one value for each vertex in the feature being written. This is a comma separated list of floating values which correspond to the vertex measures. The first value is for the first vertex, second for the second and so on.
SE_ROW_ID	For tables that are spatially enabled with vector data, this is the value for the internal SDE row number as defined by SDE.

Features read from, or written to, the SDE also have an attribute for each column in the database table.

Points

sde30_type: sde30_point

Features with this value are point features or a multi-part feature consisting of points. This is used by both the reader and the writer.

NIL Coordinates

sde30_type: sde30_nil

Features with this value are features or multi-part features consisting of no coordinates. This is used by both the reader and the writer.

Lines

sde30_type: sde30_line

Features with this value are features or multi-part features consisting of linear features. This type of linear feature is allowed to touch or cross over itself. This is used by both the reader and the writer.

Simple Lines

sde30_type: sde30_simple_line

Features with this value are features or multi-part features consisting of linear shapes. This type of linear feature is not allowed to touch or cross over itself. This is used by both the reader and the writer.

Areas

sde30_type: sde30_area

Features with this value are features or multi-part features consisting of area shapes. An area shape is a shape that forms either a polygon or a donut polygon.

Circles

sde30_type: sde30_circle

Features with this value are point features with the point specifying the centre of the circle. The rest of the circle is described using the following attributes.

Attribute Name	Contents
sde30_radius	The radius of the circle.
sde30_num_points	The number of points to use when creating the circle.

Features of this type are only used by the writer for reasons of convenience and are stored in the SDE as polygons. When read, they come out as area features tagged with **sde30_type** of **sde30_area**.

Ellipse

sde30_type: sde30_ellipse

Features with this value are point features with the point specifying the centre of the ellipse. An area shape is a shape that forms either a polygon or a donut polygon.

Attribute Name	Contents
sde30_major_axis	The major axis of the ellipse.
sde30_minor_axis	The minor axis of the ellipse.
sde30_num_points	The number of points to use when creating the ellipse.
sde30_rotation	The angle of the ellipse given in degrees measured counterclockwise from horizontal.

Features of this type are only used by the writer for reasons of convenience and are stored in the SDE as polygons. When read, they come out as area features tagged with **sde30_type** of **sde30_area**.

Rasters

sde30_type: sde30_raster

Raster features are stored in SDE as either raster maps or rows in raster catalogs. Raster maps are always read as one feature, whereas each row in a raster catalog is read as a separate raster feature.

Attribute Name	Contents
sde30_raster_compression	Overrides the value of the table level parameter SDE_COMPRESS_TYPE , on a feature-by-feature basis. This attribute is only used for raster catalogs, and will be ignored if specified on features to be added to a raster map.
sde30_raster_pyramid_interp_type	Overrides the value of the table level parameter SDE_PYRAMID_INTERPOLATION , on a feature-by-feature basis. This attribute is only used for raster catalogs, and will be ignored if specified on features to be added to a raster map.

Attribute Name	Contents
sde30_raster_pyramid_level_type	Overrides the value of the table level parameter SDE_PYRAMID_LEVEL_TYPE , on a feature-by-feature basis. This attribute is only used for raster catalogs, and will be ignored if specified on features to be added to a raster map.
sde30_raster_pyramid_max_level	Overrides the value of the table level parameter SDE_PYRAMID_MAX_LEVEL , on a feature-by-feature basis. This attribute is only used for raster catalogs, and will be ignored if specified on features to be added to a raster map.
sde30_raster_stats_type	Overrides the value of the table level parameter SDE_RASTER_STATS_TYPE , on a feature-by-feature basis. This attribute is only used for raster catalogs, and will be ignored if specified on features to be added to a raster map.
sde30_raster_mosaic_mode	Overrides the value of the table level parameter SDE_RASTER_MOSAIC_MODE , on a feature-by-feature basis. This attribute is only used for raster maps, and will be ignored if specified on features to be added to a raster catalog.

Annotation

The SDE30 enables annotation information to be attached to any feature within its database. Unlike other systems where text or annotations are standalone features, in the SDE30 annotation is an optional part of any feature. It should be noted that annotations can only be stored in spatial columns where annotation is permitted.

The following attributes are used to store the annotation information within an FME feature. If the **sde30_text_string** is specified and no location or position information is stipulated, then the **text_string** is placed at the first coordinate of the associated feature and given a rotation of 0.

Attribute Name	Contents
sde30_text_string	The annotation string.
sde30_text_size	The size of the text in user units. Default: 1.0
sde30_text_gap_ratio	The gap between the characters in the text. Default: 0.0
sde30_text_level	The annotation level. Default: 1
sde30_text_x_offset	The x coordinate of the first point of the annotation offset. Default: 0.0

Attribute Name	Contents
sde30_text_y_offset	The y coordinate of the first point of the annotation offset. Default: 0.0
sde30_text_symbol	The annotation symbol number. Default: 1
sde30_text_x	The location of the text when the text is placed with a single point. This is used in conjunction with sde30_rotation . If not specified, then the first point of the associated shape is used.
sde30_text_y	The location of the text when the text is placed with a single point. This is used in conjunction with sde30_rotation . If not specified, then the first point of the associated shape is used.
sde30_text_z	The location of the text when it is placed with a single point. This is used in conjunction with sde30_rotation . If not specified, then the first point of the associated shape is used.
sde30_text_x_location	An array of x coordinates that define the placement shape for the annotation. The same number of coordinates must be specified in the following comma-separated arrays: sde30_text_x_location , sde30_text_y_location and sde30_text_z_location (optional)
sde30_text_y_location	An array of y coordinates that defines the placement shape for the annotation. The same number of coordinates must be specified in the following comma-separated arrays: sde30_text_x_location , sde30_text_y_location and sde30_text_z_location (optional)
sde30_text_z_location	An array of z coordinates that defines the placement shape for the annotation. The same number of coordinates must be specified in the following comma-separated arrays: sde30_text_x_location , sde30_text_y_location and sde30_text_z_location (optional)
sde30_text_x_leader	An array of x coordinates that defines the leader line for the shape annotation. The same number of coordinates must be specified in the following comma-separated arrays: sde30_text_x_leader , sde30_text_y_leader and sde30_text_z_leader (optional)

Attribute Name	Contents
sde30_text_y_leader	An array of y coordinates that defines the leader line for the shape annotation. The same number of coordinates must be specified in the following comma-separated arrays: <code>sde30_text_x_leader</code> , <code>sde30_text_y_leader</code> and <code>sde30_text_z_leader</code> (optional)
sde30_text_z_leader	An array of z coordinates that defines the leader line for the shape annotation. The same number of coordinates must be specified in the following comma-separated arrays: <code>sde30_text_x_leader</code> , <code>sde30_text_y_leader</code> and <code>sde30_text_z_leader</code> (optional)
sde30_rotation	The rotation of the annotation measured from the horizontal in a counterclockwise direction. This is only used when the annotation location is specified using <code>sde30_text_x</code> , <code>sde30_text_y</code> , and <code>sde30_text_z</code> . Default: 0
sde30_justification	The justification of the text relative to its offset point. Range: <code>sde30_upper_left</code> <code>sde30_upper_center</code> <code>sde30_upper_right</code> <code>sde30_center_left</code> <code>sde30_center_center</code> <code>sde30_center_right</code> <code>sde30_lower_left</code> <code>sde30_lower_center</code> <code>sde30_lower_right</code> Default: <code>sde30_lower_left</code>

Troubleshooting

Connecting to ArcSDE

Problems sometimes arise when attempting to connect to an ArcSDE database. This is almost always due to a misconfiguration in the user's environment.

- Ensure you have configured the services file so that the port number specified for the SDE instance you are connecting to matches that of the server.

Transactional Version

You cannot browse for Transactional Versions (or Tables) in the Parameters box if:

1. Multiple SDE instances have been installed in the database,
2. The instance you are connecting to is not the default SDE instance,
3. You are not connecting as the user that owns the instance, or
4. You have not manually entered a valid Transactional Version.

In that case, you must manually enter the name of a valid Transactional Version before proceeding. It will normally be "<owning username>.DEFAULT".

Miscellaneous

- It is not possible to write to an ArcSDE business table with one column when it is a blob column. The table must have an additional column as well.

ESRI Geodatabase Reader/Writer

Note: To use FME's ESRI Geodatabase Reader/Writer, you must also install ArcGIS® Desktop.

The Geodatabase reader and writer modules allow FME to store data in and retrieve data from ESRI's Geodatabase. Support is provided for translating several aspects of a Geodatabase, and with the size of ESRI's ArcObjects, further expansion of the reader/writer will almost certainly continue to cover more diverse aspects of the format.

Geodatabase Quick Facts

Format Type Identifier	<ul style="list-style-type: none">• GEODATABASE_SDE (ArcSDE)• GEODATABASE_MDB (Access)• GEODATABASE_FILE (File-based)• GEODATABASE_SDE_RASTER_DATASET (ArcSDE)• GEODATABASE_FILE_RASTER_DATASET (File-based)
Reader/Writer	Both
Licensing Level	<ul style="list-style-type: none">• File: Base• File Raster: Professional• MDB: Base• SDE: ESRI Edition• SDE Raster: ESRI Edition
Dependencies	ArcGIS Desktop
Dataset Type	<ul style="list-style-type: none">• Database (ArcSDE)• File (Access)• Directory (File-based)
Feature Type	Feature Class name
Typical File Extensions	<ul style="list-style-type: none">• for Personal Geodatabase: .mdb , .accdb• for File-based Geodatabase: .gdb
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Always
Schema Required	Yes
Transaction Support	Yes
Enhanced Geometry	Yes
Geometry Type	geodb_type
Encoding Support	Yes

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	yes		polygon	yes
circular arc	yes		raster	yes
donut polygon	yes		solid	yes
elliptical arc	yes		surface	yes
ellipses	yes		text	yes
line	yes		z values	yes
none	yes			

Overview

The Geodatabase reader and writer translates several different types of features:

- table-level metadata
- reading and writing of geometric features such as points, multipoints, polylines, and polygons
- reading and writing of non-spatial table data
- reading and writing of annotations, including leader lines and feature-linked annotations
- reading and writing of dimensions
- reading and writing of geometric network features, including simple junctions, complex junctions (reading only), simple edges and complex edges
- reading and writing of relationships, including attributed relationships
- reading and writing of raster data
- reading and writing of multipatch features, including textured surfaces

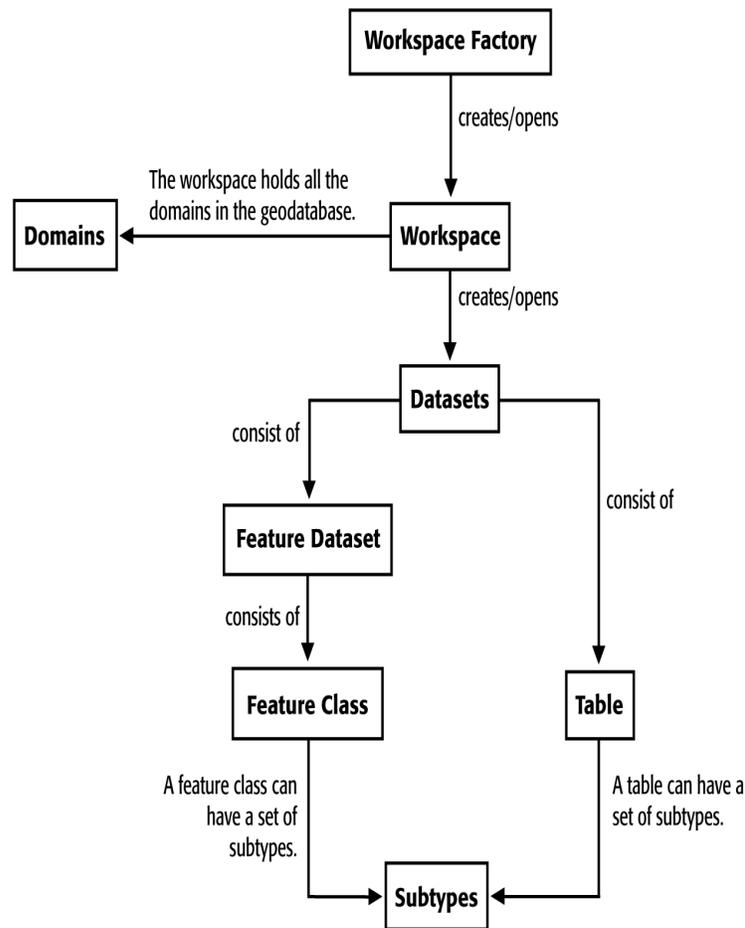
The Geodatabase modules also provide the following capabilities:

- **Programmatic Table Creation:** Tables need not be created before a data import operation. All table creation details are handled by the FME.
- **Transaction Support:** Transactions are fully supported, enabling a partially complete load of operation to be resumed later, without the loss or duplication of data.
- **Update/Delete Support:** In addition to appending features, the Geodatabase writer provides the ability to update and delete existing features in the Geodatabase.
- **Attribute Query Support:** SQL WHERE clauses can be specified to limit the data being exported.
- **Spatial Query Support:** FME exploits the spatial query capabilities of Geodatabase using both search envelopes and search features with a multitude of filtering options. This allows greater control to translate only the relevant spatial data.
- **Non-Spatial Table Support:** Any table can be read or written to a Geodatabase with FME, whether or not it contains spatial data. The FME can read, write, and create regular RDBMS tables (such as those in Oracle) in Geodatabase.
- **Versioning Support:** FME enables data to be read from a particular transactional version or historical marker of an Enterprise Geodatabase, and also allows data to be written to a specific transactional version of an Enterprise Geodatabase.
- **Enhanced Geometry Model Support:** Both the Geodatabase Reader and Writer support the enhanced geometry model. The addition of enhanced geometry model support allows lines and polygons containing arcs to be maintained, rather than stroked.
- **Fully Automatic Import and Export:** The FME's Geodatabase support provides fully automated import and export of data through the FME's Graphical User Interface (GUI). This is ideal for quick data imports and exports.

- Mapping File Customization: The FME's ability to generate mapping files for user customization allows greater and more precise control over Geodatabase translations.
- Unicode Support: Geodatabase text columns are stored in the UTF-16 encoding. FME can read and write this data.
- Archiving Support: The Geodatabase reader can retrieve archived data from a table that has archiving enabled, through the use of a WHERE clause constrained by the gdb_from_date and gdb_to_date attributes.

Conceptual Diagram of Geodatabase

Below is a simplified diagram of some of the objects within Geodatabase. Labels describe the relationship between objects.



Workspace Factory - An object that allows you to connect to a workspace. The workspace Factory is format-dependent. This means that a different Workspace Factory is used to open an ArcSDE Workspace than to open an MS Access Workspace.

Workspace - a datasource that contains datasets. The workspace represents the actual Geodatabase being created or opened.

Dataset - A collection of data that is grouped together. Examples include feature classes, tables, and feature datasets.

Table - A container of non-spatial data. It is never part of a feature dataset.

Feature Class - A container of spatial data. May be part of feature dataset or standalone.

Feature Dataset - A container of feature classes. All the feature classes within the feature dataset share the spatial reference of the feature dataset.

Domain - An object that is used to constrain a field in a feature class or table to an allowable value for that field.

Subtype - An object that is used to separate rows (features) in a table (feature class) into different groups so that different default values or validation rules can be used for each group.

Reader Overview

The Geodatabase reader begins by opening the Geodatabase dataset that resides within a server/file system. Once opened, the Geodatabase reader queries the Geodatabase and passes the resulting features – that is, rows – on to the FME for processing. Every feature that is read is tagged with its original integer object ID (this is the object ID Geodatabase assigns it) under the attribute name `geodb_oid`.

Unlike other formats supported by FME, the Geodatabase reader has several different reader types to account for file- and enterprise-based datasets, as well as vector and raster. When reading from an Enterprise Geodatabase, the `<ReaderType>` is `GEODATABASE_SDE` or `GEODATABASE_SDE_RASTER_DATASET`, when reading from a Personal Geodatabase (an MS Access file), the `<ReaderType>` is `GEODATABASE_MDB`, and when reading from a File-based Geodatabase (a directory ending in `.gdb`), the `<ReaderType>` is `GEODATABASE_FILE` or `GEODATABASE_FILE_RASTER_DATASET`. By default, the `<ReaderKeyword>` is the same as the `<ReaderType>`.

When reading features from the Geodatabase, the tables from which features are retrieved are specified in the mapping file using the `<ReaderKeyword>_IDs`.

The Geodatabase reader uses the `<ReaderKeyword>_IDs` statement to identify the tables from which data is to be retrieved. If no identifiers (IDs) are specified and no DEF lines are specified and the Enterprise Geodatabase reader is used, then no features are read from the database. However, if no identifiers (IDs) are specified and no DEF lines are specified and the Personal Geodatabase or File-based Geodatabase reader is used, then all features are read from the database.

The table below summarizes the different feature retrieval modes supported by the Geodatabase reader module. The word *table* refers to both non-spatial tables and feature classes. However, *feature class* applies only to feature classes and not tables. The next section contains a detailed description of each directive.

Search Type	Search Directive Suffix	Description
Non-Spatial and Spatial Retrieval	IDs	Specifies the tables from which features are to be retrieved. If no tables are specified and the Personal Geodatabase or File-based Geodatabase reader is being used then all features are retrieved. If no tables are specified and the Enterprise Geodatabase reader is being used then no features are retrieved.
	WHERE	Specifies the attribute constraint that a feature must have to be retrieved. The where clause follows the SQL syntax of the underlying database, except that ORDER BY, GROUP BY, nested queries, and aggregate functions (i.e. MAX, COUNT) cannot be used.

Search Type	Search Directive Suffix	Description
Spatial Retrieval	SEARCH_ENVELOPE	Specifies the spatial extent of the feature retrieval. Only features that have the relationship specified by SEARCH_METHOD with the envelope are returned. This cannot be specified at the same time as a SEARCH_FEATURE is specified.
	SEARCH_FEATURE	Specifies a feature with an arbitrary number of coordinates as the search feature. Only features that have the relationship specified by SEARCH_METHOD with the search feature are returned. This cannot be specified at the same time as a SEARCH_ENVELOPE is specified. Also, when specifying the search_feature, make sure it has a simple geometry. If it does not have a simple geometry, then its geometry is always simplified by the Geodatabase reader.

Reader Directives – Geodatabase Feature Classes

This section describes the directives that are recognized by the Core Geodatabase reader module.

Each directive is prefixed by the current <ReaderKeyword>_ when placed in a mapping file. Unless otherwise specified, the <ReaderKeyword> for the Geodatabase reader is the same as the <ReaderType>.

The following directives are used by all Geodatabase types when reading feature classes, and are not applicable to raster datasets.

FEATURE_READ_MODE

Required/Optional: *Optional*

This directive provides the ability to read table-level metadata when set to Metadata. In this mode, the reader outputs one feature per feature type. The geodb_type of the feature is geodb_metadata and the entire XML metadata document belonging to the Geodatabase table is found in the attribute geodb_metadata_string. Where applicable, the following attributes are also supplied: fme_feature_identifier which indicates the name of the object ID field, fme_num_entries (personal geodb only) which indicates the number of features in the table, fme_contains_spatial_column which indicates whether the table has a geometry column (i.e. in ESRI ArcGIS terms, whether the table is a feature class), fme_geometry{0} which indicates the types of geometry the feature class contains, fme_dimension which indicates whether the feature class is 2D or 3D. If the table is a feature class, the geometry of the metadata feature returned is a polygon, representing the extents of the feature class and the coordinate system of the feature class also gets set on the feature. When reading metadata, the IDs and DEF keywords are used to determine which feature types should have metadata read from them.

When set to Features, the reader outputs features stored within tables.

Parameter: <feature_read_mode>

Values: *Features | Metadata*

Default Value: *Features*

Workbench Parameter: *Feature Read Mode*

Example:

```
GEODATABASE_SDE_FEATURE_READ_MODE Metadata
```

WHERE**Required/Optional:** *Optional*

An SQL-like (determined by the underlying database) WHERE clause that selects only certain records for extraction from the Geodatabase.

The specified WHERE clause is passed to the Geodatabase for processing. The WHERE clause can be almost like an SQL clause (using the syntax supported by the underlying database) except that ORDER BY, GROUP BY, nested queries, and aggregate functions (i.e. MAX, COUNT) cannot be used.

This WHERE clause applies to all tables retrieved. For more specific queries, see the Reader directive DEF.

Workbench Parameter: *Where Clause*

Example:

The WHERE clause specified below instructs the FME to retrieve features from the Geodatabase for the tables that are listed on the <ReaderKeyword>_IDs lines (unless no _IDs lines are specified in which case all tables are examined). The features retrieved must have for their ObjectID a value greater than 10 and their City attribute must be Vancouver.

```
GEODATABASE_SDE_WHERE ObjectID > 10 AND \
  City ='Vancouver'
```

DEF**Required/Optional:** *Optional*

Describes tables. Normally these lines are automatically generated within a mapping file using FME. When reading from an Enterprise Geodatabase, the table names on the DEF lines may be prefixed by the user ID of the person who created the table, followed by a period (for example, <userid>.<tablename>). The only table names that must be prefixed by a user ID are those tables that were not created using your own user ID. However, you may still not be able to access another person's tables if your user ID doesn't have the correct privileges.

This directive is usually automatically generated while generating a mapping file for a specific Geodatabase, but there is one way it can be customized. An automatically generated DEF might look like this:

```
GEODATABASE_MDB_DEF IndexGrid \
  geodb_type geodb_polyline \
  GEODB_OID integer \
  OBJECTID integer \
  Entity char(254) \
  Handle char(254) \
  Layer char(254) \
  Color integer \
  Linetype char(254) \
  Elevation double \
  Thickness double \
  SHAPE_Length double
```

Note: The returned feature types will match the table names on the DEFs exactly (including the character case). As a result, if the DEF's table name was IndexGrid (with no owner prefix) then the feature type of the returned features would also be IndexGrid. If the DEF's table name was sde.IndexGrid then the feature type would be sde.IndexGrid.

Customizing Reader DEF Lines

With already generated and working DEF lines, a WHERE clause can be added just like a normal attribute on a DEF line above, except that instead of the type (such as double), the value will be an SQL WHERE clause. This clause MUST be in double quotation marks (") and must conform to the same restrictions as the WHERE clause directive listed above. In addition, the clause cannot be continued on to the next line so a continuation character (\) cannot appear in the middle of the clause.

Example:

The WHERE clause specified below instructs the FME to retrieve features from the feature class IndexGrid with the constraint that the Color value must be 5. Note that the WHERE clause is case-sensitive, and can appear on any line.

```
GEODATABASE_MDB_DEF IndexGrid \  
  geodb_type geodb_polyline \  
  GEODB_OID integer \  
  OBJECTID integer \  
  Entity char(254) \  
  Layer char(254) \  
  Color integer \  
  Linetype char(254) \  
  Elevation double \  
  Thickness double \  
  WHERE "Color = 5" \  
  SHAPE_Length double
```

Workbench Parameter: <WorkbenchParameter>**IDs****Required/Optional:** *Optional*

This statement specifies the tables from which features are to be retrieved. There may be multiple GEODATABASE_<SDE|MDB|FILE>_IDs statements within a single FME mapping file, in which case the input set of tables comprises the union of all GEODATABASE_<SDE|MDB|FILE>_IDs statements. The Geodatabase reader module only extracts features from the identified tables. If no GEODATABASE_<SDE|MDB|FILE>_IDs lines appear in the mapping file, then all tables with DEF lines will be used as the input set. If the Personal Geodatabase or File-based Geodatabase reader is being used and there are no DEF lines and no IDs, then all the tables will be read. If the Enterprise Geodatabase reader is being used and there are no DEF lines and no IDs, then no tables will be read. (This behavior is different from reading Personal and File-based Geodatabases.)

The returned feature types will match the IDs exactly (including the character case). As a result, if the IDs was tableOne (with no owner prefix) then the feature type would also be tableOne. If the IDs was sde.tableOne then the feature type would be sde.tableOne.

Parameter: <[table name]+>

Note: The table name must be exactly the same as it appears on the DEF line. For Enterprise Geodatabases, the tables on the DEF line may be prefixed by the user ID and a period (for example, <userid>.). If this is the case, then corresponding tables on the IDs line must also be prefixed by the username and a period.

Workbench Parameter: *Feature Types to Read***Enterprise Geodatabase Example:**

As shown below, the GEODATABASE_SDE_IDS is a list of table names. In the example, features are read from the table roads, and then from the table streets. Both tables are owned by a user named jacob. Each ID is treated as a separate query to the database. In this example, the assumption is made that the DEF lines for these tables also contain the user ID and a period. If this was not the case, then no tables would be found because the table names on the IDs line would not match up with the table names on the DEF lines, even though they may be referring to the same table.

```
GEODATABASE_SDE_IDS jacob.roads jacob.streets
```

Personal and File-based Geodatabase Example:

As shown below, the GEODATABASE_<MDB|FILE>_IDs is a list of table names. In the example, features are read from the table roads, and then from the table streets. Each ID is treated as a separate query to the database. Reading from a File-based Geodatabase would look exactly the same except that the reader directive GEODATABASE_MDB would be replaced with GEODATABASE_FILE.

```
GEODATABASE_MDB_IDS roads streets
```

SEARCH_ENVELOPE

Required/Optional: *Optional*

Specifies a rectangular area to be used in conjunction with the SEARCH_METHOD directive for extraction of spatial features. This cannot be specified at the same time as a SEARCH_FEATURE is specified.

Parameters:

<min-x>

The minimum x coordinate in the coordinate system of the feature(s) being retrieved.

<min-y>

The minimum y coordinate in the coordinate system of the feature(s) being retrieved.

<max-x>

The maximum x coordinate in the coordinate system of the feature(s) being retrieved.

<max-y>

The maximum y coordinate in the coordinate system of the feature(s) being retrieved.

Workbench Parameter: *Minimum X, Minimum Y, Maximum X, Maximum Y*

Example:

```
GEODATABASE_MDB_SEARCH_ENVELOPE 6190 57239 6310 57549
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]

* Workbench Parameter

Clip To Envelope

SEARCH_FEATURE

Required/Optional: *Optional*

The SEARCH_FEATURE clause provides a mechanism for specifying an arbitrarily complex search feature. The SEARCH_FEATURE clause works with the SEARCH_METHOD clause to define the spatial constraint, but cannot be specified at the same time as a SEARCH_ENVELOPE is specified.

Note: It is recommended you use a simplified search feature. If you don't, the reader will simplify the search feature and this could produce unexpected results.

Parameter: [*<xCoord> <yCoord>*]+ (A list of the coordinates defining the geometry of the query geometry.)

Workbench Parameter: *Search Feature*

Example:

The example below defines an equivalent geometry to the GEODATABASE_MDB_SEARCH_ENVELOPE example shown above using the GEODATABASE_<SDE|MDB|FILE>_SEARCH_FEATURE clause.

```
GEODATABASE_SDE_SEARCH_FEATURE 6190 57239 6190 57549 \  
6310 57549 6310 57239 \  
6190 57239
```

SEARCH_ORDER

Required/Optional: *Optional*

Specifies the order that the underlying search is performed on the Geodatabase. This directive determines whether the spatial component or the attribute component of a query is performed first. The benefit of using this directive is for efficiency. For example, if the attribute component would filter the data more than would the spatial component, then it would be desirable to use the SEARCH_ORDER directive to force the attribute component to be used first. If the directive is not used, then by default the spatial component is used first.

Parameter: *<search_order>*

Value: *SPATIAL_FIRST | ATTRIBUTE_FIRST*

Workbench Parameter: *Search Order*

Example:

```
GEODATABASE_SDE_SEARCH_ORDER SPATIAL_FIRST
```

SEARCH_METHOD

Required/Optional: *Optional*

This directive specifies the type of spatial relationship the queried spatial features must have with either the SEARCH_ENVELOPE or the SEARCH_FEATURE in order to be returned. (Note that only one of SEARCH_ENVELOPE and SEARCH_FEATURE may be specified at a time.)

Parameter: *<search_method>*

Values: The values for the search method mostly follow the basic Clementini relationships that are used by ESRI. For further information on these relationships, see *Exploring ArcObjects Vol. II: Geographic Data Management Chapter 8*.

The value of the SEARCH_METHOD can be one of the following:

- GEODB_INTERSECTS: Features must intersect with the query geometry.
- GEODB_ENVELOPE_INTERSECTS: The envelopes of the features must intersect with the envelope of the query geometry.
- GEODB_TOUCHES: Features must touch the query geometry.
- GEODB_OVERLAPS: The query geometry overlaps the features returned.
- GEODB_CROSSES: The query geometry crosses the feature returned.
- GEODB_WITHIN: The query geometry is within the features returned.
- GEODB_CONTAINS: The query geometry contains the features returned.

Default value: *GEODB_INTERSECTS*

Workbench Parameter: *Search Method*

Example:

```
GEODATABASE_MDB_SEARCH_METHOD GEODB_CONTAINS
```

SPLIT_AT_ARCS (only applicable with classic geometry)

Required/Optional: *Optional*

This directive specifies whether or not to vectorize arcs. When set to NO arcs that are a piece of paths or polygons are vectorized - arcs not part of a larger geometry will remain as arcs. When set to YES during workspace/mapping file generation, all polygon feature classes will indicate that they contain polyline geometry rather than polygon geometry because it is assumed that the value for this directive will remain as YES for the translation. Changing the value to NO for the translation may produce unexpected results.

When this directive is set to YES for the translation, arcs are not vectorized and lines/polygons containing arcs are split up into arcs and lines. Each piece receives all the user-defined attributes, and gets tagged with an additional two attributes: `geodb_segment_index` and `geodb_original_geometry`. The attribute `geodb_segment_index` is zero-based (i.e., the first piece has an index of 0) and can be used to piece back together the original geometry. The second attribute, `geodb_original_geometry`, indicates whether the original geometry was a line, polygon, or a donut. A piece will only be tagged as a donut if it was a hole in a polygon or if it was a shell that contained holes. As a result, an island that does not contain any holes will have `geodb_original_geometry` set to polygon. If the piece is an arc or an ellipse, it will contain additional attributes describing its characteristics.

When this directive is set to NO for the translation, arcs are vectorized. As a result, polygon/polyline features with arc segments retain their original geometry, rather than being split up into individual pieces. When 3D arcs are vectorized, the z values of the arc are linearly interpolated from the start point to the end point.

When features are read using enhanced geometry, this directive will be ignored. To split enhanced geometry paths, use the PathSplitter transformer.

Note: This directive is not valid when reading relationship classes.

Parameter: *<split_at_arcs>*

Values: *YES | NO*

Default: *NO*

Workbench Parameter: *<WorkbenchParameter>*

Example:

```
GEODATABASE_SDE_SPLIT_AT_ARCS YES
```

TRANSLATE_SPATIAL_DATA_ONLY

Required/Optional: *Optional*

This directive is used for translating spatial data only. When set to YES, non-spatial tables, relationships, domains, and subtypes will not be translated. If this directive is specified when generating a workspace or mapping file, then no schemas will be returned for non-spatial tables.

Parameter: `<translate_spatial_data_only>`

Values: YES | NO

Default Value: NO

Workbench Parameter: *Spatial Data Only*

Example:

```
GEODATABASE_MDB_TRANSLATE_SPATIAL_DATA_ONLY YES
```

RESOLVE_DOMAINS

Required/Optional: *Optional*

This directive specifies whether to resolve attributes that have a coded value domain associated with them (either a default domain, or one set up through a subtype). This means that when an attribute of a feature has a coded value domain associated with it, another attribute will also be added that represents the textual description of the coded attribute. The new attribute will be `<attribute-name>_resolved`, where `<attribute-name>` is the name of the attribute containing the code. This attribute will only be added when `<attribute-name>` contains a non-NULL value.

Parameter: `<resolve_domains>`

Values: YES | NO

Default Value: NO

Workbench Parameter: *Resolve Domains*

Example:

```
GEODATABASE_MDB_RESOLVE_DOMAINS YES
```

RESOLVE_SUBTYPE_NAMES

Required/Optional: *Optional*

This directive specifies whether to resolve the subtype field of a feature. A feature that exists in a table that has subtypes will have an attribute that is the subtype field. The subtype field will hold an integer value that specifies which subtype the feature belongs to, and this integer value also has a string name equivalent called the description. If YES is specified for this directive, the corresponding description will be added as an attribute on the feature, and the attribute will be `geodb_subtype_name`. When set to YES during the generation of a mapping file/workspace, the schema for a table with subtypes will contain the attribute `geodb_subtype_name`.

Parameter: `<resolve_subtype_names>`

Values: YES | NO

Workbench Parameter: *Resolve Subtypes*

Example:

```
GEODATABASE_SDE_RESOLVE_SUBTYPE_NAMES YES
```

IGNORE_NETWORK_INFO

Required/Optional: *Optional*

This directive determines whether to read the network information belonging to a network feature. When set to YES, junctions will be treated as point features, and edges will be treated as polyline features, with the `geodb_type` being set to `geodb_point` and `geodb_polyline`, respectively. When set to NO, Geodatabase specific attributes describing network information such as network connectivity will be inserted on the feature. The geometry of the feature remains

the same regardless of the value given to this directive. The speed of reading network features is vastly improved if the network info is ignored.

Parameter: <ignore_network_info>

Values: YES | NO

Default Value: NO

Workbench Parameter: Ignore Network Info

Example:

```
GEODATABASE_SDE_IGNORE_NETWORK_INFO YES
```

IGNORE_RELATIONSHIP_INFO

Required/Optional: *Optional*

This directive determines whether to read relationship features present in a source dataset. When set to YES, feature types containing simple relationship will be ignored, and feature types containing attributed relationships will be treated as non-spatial tables. When set to NO, relationships will be read normally as either simple or attributed. The speed of reading features is vastly improved if relationships are ignored.

Parameter: <ignore_relationship_info>

Values: YES | NO

Default Value: NO

Workbench Parameter: Ignore Relationship Info

Example:

```
GEODATABASE_SDE_IGNORE_RELATIONSHIP_INFO YES
```

OMIT_GENERIC_OBJECTID_ATTRIBUTE

This directive is only used when generating a workspace or reading schema features using FME Objects.

This directive determines whether each schema should add an additional integer attribute called geodb_oid, a generic attribute representing the object id field. Valid values are YES and NO. This attribute can be useful when the real object id field is unknown because the value for this attribute will be the object id's of the features read. Regardless of the value for this directive, the real object id field will always be supplied on the schema. This directive affects schema generation only. This means that the attribute will still be on the feature during reading, even if the attribute wasn't on the schema.

Parameter: <omit_generic_objectid_attr>

Values: YES | NO

Default Value: NO

Example:

```
GEODATABASE_SDE_OMIT_GENERIC_OBJECTID_ATTRIBUTE YES
```

SPLIT_COMPLEX_EDGES

This directive determines whether complex edge features should be split. When split, complex edge features are read at the **element** level rather than the **feature** level. The element level represents the logical view of the geometric network. As a result, no network connectivity information is lost. When split, each FME feature stores the following attributes:

Attribute Name	Contents
geodb_element_id	The element ID of the logical edge element.

Attribute Name	Contents
geodb_element_index	An attribute created and assigned by FME. It is used to order the edge elements within a complex feature. The index begins at zero, not one.
geodb_from_junction_element_id	The junction element ID that corresponds to the <i>from endpoint</i> . Note: This is the <i>from endpoint</i> of the edge element, not the edge feature.
geodb_to_junction_element_id	The junction element ID that corresponds to the <i>to endpoint</i> . Note: This is the <i>to endpoint</i> of the edge element, not the edge feature.

The following complex edge attributes are not present on the FME feature: geodb_junction_feature_count and geodb_edge_element_count. Even though elements are being read, the geodb_type of each feature is still geodb_complex_edge.

If an error occurs when retrieving the geometry for an edge element, then the geometry is skipped but the network attributes are still read.

Note: This directive is not valid when reading relationship classes.

Parameter: <split_complex_edges>

Values: YES | NO

Default Value: NO

Workbench Parameter: Split Complex Edges

Example:

```
GEODATABASE_MDB_SPLIT_COMPLEX_EDGES YES
```

RETRIEVE_ALL_SCHEMAS

Required/Optional: *Optional*

This specification is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

When set to 'Yes', indicates to the reader to return all the schemas of the tables in the database.

If this specification is missing then it is assumed to be 'No'.

Range: YES | NO

Default: NO

RETRIEVE_ALL_TABLE_NAMES

Required/Optional: *Optional*

This specification is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

Similar to RETRIEVE_ALL_SCHEMAS; this optional directive is used to tell the reader to only retrieve the table names of all the tables in the source database. If RETRIEVE_ALL_SCHEMAS is also set to "Yes", then RETRIEVE_ALL_SCHEMAS will take precedence. If this value is not specified, it is assumed to be "No".

Range: YES | NO

Default: NO

CHECK_SIMPLE_GEOM

Required/Optional: *Required*

This directive specifies whether a check should be performed on features read from geodatabase to determine whether they are simple. This is an expensive check and impacts reader performance.

Parameter: `<check_simple_geom>`

Range: YES | NO

Default: NO

Workbench Parameter: *Check for Simple Geometry*

Example:

```
GEODATABASE_MDB_CHECK_SIMPLE_GEOM YES
```

READ_FEAT_LINKED_ANNOS

Required/Optional: *Required*

This directive specifies whether feature-linked annotations should have their text, angle and position properties merged as attributes onto the main feature that they are linked to, when reading. If set to yes, this will produce a list attribute as detailed in the section on Annotations with all annotation attributes set and the annotation table(s) need not be read explicitly. If set to no, feature-linked annotations will be read normally as annotations when encountered.

Parameter: `<read_feat_linked_annos>`

Range: YES | NO

Default: NO

Workbench Parameter: *Merge Feature Linked Annotations*

Example:

```
GEODATABASE_MDB_READ_FEAT_LINKED_ANNOS YES
```

SPLIT_MULTI_PART_ANNOS

Required/Optional: *Optional*

This directive specifies whether or not to split multi-part annotations into separate features for each 'element' when reading. If set to yes, a single feature for each element (usually a word) in a multi-part annotation will be produced on reading, resulting in feature-specific attributes such as angle and text position being stored according to the location of each element. If set to no, multi-part annotations will be read normally, as a single feature storing a single set of attributes describing the positioning of the text.

Parameter: `<split_multi_part_annos>`

Values: YES | NO

Default: NO

Workbench Parameter: *Split Multi-Part Annotations*

Example:

```
GEODATABASE_SDE_SPLIT_MULTI_PART_ANNOS YES
```

VALIDATE_FEATURES

Required/Optional: *Optional*

This directive specifies whether or not to validate features passed to the geodatabase writer. When set to 'yes', validation is performed on the subtype, attribute rules, relationship rules, network connectivity rules and any custom rules present on the feature class. Failed features will be logged with an extended error message describing the reason for the failure. When set to 'no', validation is not performed.

Parameter: <validate_features>

Values: YES | NO

Default: NO

Workbench Parameter: <workbench Parameter>

Example:

```
GEODATABASE_SDE_VALIDATE_FEATURES YES
```

BEGIN_SQL{n}

Occasionally you must execute some ad-hoc SQL prior to opening a table. For example, it may be necessary to ensure that a view exists prior to attempting to read from it.

Upon opening a connection to read from a database, the reader looks for the directive <ReaderKeyword>_BEGIN_SQL{n} (for n=0,1,2,...), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the FME_SQL_DELIMITER keyword, embedded at the beginning of the SQL block. The single character following this keyword will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;  
DELETE FROM instructors;  
DELETE FROM people  
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

* Workbench Parameter

SQL Statement to Execute Before Translation

END_SQL{n}

Occasionally you must execute some ad-hoc SQL after closing a set of tables. For example, it may be necessary to clean up a temporary view after writing to the database.

Just before closing a connection on a database, the reader looks for the directive <ReaderKeyword>_END_SQL{n} (for n=0,1,2,...), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the FME_SQL_DELIMITER directive, embedded at the beginning of the SQL block. The single character following this directive will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;
DELETE FROM instructors;
DELETE FROM people
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

* Workbench Parameter

SQL Statement to Execute After Translation

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Reader Directives – Enterprise Geodatabase

These directives are used when connecting to an Enterprise Geodatabase.

CONNECTION_FILE

Required/Optional: *Optional*

This statement identifies the pathname of a connection file to be used to connect to an Enterprise Geodatabase. A connection file provides the necessary information to connect to the SDE server, such as the server name or the user-name. The connection file must be a *.sde file and have the proper format for a connection file as defined by ESRI. If you specify a connection file, you do not need to specify the directives SERVER, INSTANCE, USER, PASSWORD, VERSION, HISTORICAL_VERSION_NAME and HISTORICAL_VERSION_TIMESTAMP. (If, however, a password was not specified in the connection file, you will be prompted for a password.)

The pathname of the connection file to be used to connect to an Enterprise Geodatabase.

Parameter: <connection_file>

Workbench Parameter: *Connection File*

Example:

```
GEODATABASE_SDE_CONNECTION_FILE C:\GeoDB\connect.sde
```

DATASET

Required/Optional: *Required*

For Enterprise Geodatabases, this is the name of the dataset from which features are retrieved. In an Enterprise Geodatabase, this dataset is referred to as the DATABASE. Some RDBMS's, such as Oracle, do not require a value, whereas others, such as SQLServer, do. For databases that do not require the value, the value not_used is specified by convention.

Workbench Parameter: *Source ESRI Geodatabase (ArcSDE) Dataset*

Example:

```
GEODATABASE_SDE_DATASET testdset
```

SERVER

Required/Optional: *Required*

The name of the Geodatabase server used to read data from the dataset.

Parameter: *<server>*

Workbench Parameter: *Server*

Example:

```
GEODATABASE_SDE_SERVER dax
```

INSTANCE

Required/Optional: *Required*

The Enterprise Geodatabase instance to which FME connects. The usual value for systems with a single ArcSDE instance is esri_sde8.

Parameter: *<instance>*

Workbench Parameter: *Instance Name*

Example:

```
GEODATABASE_SDE_INSTANCE esri_sde8
```

USERID

Required/Optional: *Optional if connecting in OSA mode*

User ID of the Enterprise Geodatabase user.

If the userid and password are missing or not set, then the reader will try and connect with AUTHENTICATION_MODE set to OSA (Operating System Authentication).

Parameter: *<userid>*

Workbench Parameter: *User ID*

Example:

```
GEODATABASE_SDE_USERID jacob
```

PASSWORD

Required/Optional: *Optional if connecting in OSA mode*

Password for the user account.

If the userid and password are missing or not set, then the reader will try and connect with AUTHENTICATION_MODE set to OSA (Operating System Authentication).

Parameter: *<password>*

Workbench Parameter: *Password*

Example:

```
GEODATABASE_SDE_PASSWORD jacobpassword
```

REMOVE_TABLE_QUALIFIER

Specifies whether to keep or remove the table name prefix. For instance, the table name might appear in the form *<owner_name>.<table_name>*.

Setting this directive to YES indicates that the reader should return the table name without the *owner_name* prefix. This is useful when:

- creating a workspace that will be passed on to another organization using the same table names,
- performing a translation to another database format but with a different user name, and
- writing to a file-based format but not wanting the prefix in the name of the feature type.

When this directive is set to YES during the generation of a mapping file or workspace, the source feature types will be the table names without any prefix; otherwise, they will contain the owner name as a prefix. It is recommended that this directive not be changed in value after generating the mapping file/workspace as it is possible for no features to be successfully passed onto the writer (since the writer is expecting feature types with different names).

Notes

- Even when REMOVE_TABLE_QUALIFIER is set to YES, if the table is owned by a user other than the current user, the *<owner_name>* prefix will not be dropped so that the reader will find the correct table.
- When the underlying database is SQL or DB2, the schema qualifier (*<owner_name>*) is always dropped, regardless of this setting.

Required/Optional

Optional

Values

YES | NO (default)

Mapping File Syntax

```
REMOVE_TABLE_QUALIFIER YES
```

* Workbench Parameter

Remove Schema Qualifier

SEARCH_ENVELOPE

Required/Optional: *Optional*

Specifies a rectangular area to be used in conjunction with the SEARCH_METHOD directive for extraction of spatial features. This cannot be specified at the same time as a SEARCH_FEATURE is specified.

Parameters:

<min-x>

The minimum x coordinate in the coordinate system of the feature(s) being retrieved.

<min-y>

The minimum y coordinate in the coordinate system of the feature(s) being retrieved.

<max-x>

The maximum x coordinate in the coordinate system of the feature(s) being retrieved.

<max-y>

The maximum y coordinate in the coordinate system of the feature(s) being retrieved.

Workbench Parameter: *Minimum X, Minimum Y, Maximum X, Maximum Y*

Example:

```
GEODATABASE_MDB_SEARCH_ENVELOPE 6190 57239 6310 57549
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

VERSION

Required/Optional: *Optional, so long as either a transactional or historical version is provided*

The version of the dataset to be read (used in multi-versioned databases). The version name must be prefixed by the owner of the version and a period. Note that the version name is case-sensitive.

Parameter: *<version>*

Workbench Parameter: *Transactional Version*

Example:

```
GEODATABASE_SDE_VERSION jim.versionone
```

HISTORICAL_VERSION_NAME

Required/Optional: *Optional, so long as either a transactional or historical version is provided*

The historical marker name of the dataset to be read (used in multi-versioned databases that have archiving enabled). Note that the version name is case-sensitive, and that historical data is read-only.

Parameter: *<historical_version_name>*

Workbench Parameter: *Historical Marker*

Example:

```
GEODATABASE_SDE_HISTORICAL_VERSION_NAME newmarker
```

HISTORICAL_VERSION_TIMESTAMP

Required/Optional: *Optional, so long as either a transactional or historical version is provided*

The specific date and time of the archived dataset to be read (used in multi-versioned databases that have archiving enabled). Note that the date and time must be provided in the correct format for the underlying database (see ArcGIS help for more information), and that historical data is read-only.

Parameter: *<historical_version_timestamp>*

Workbench Parameter: *Historical Timestamp*

Example:

```
GEODATABASE_SDE_HISTORICAL_VERSION_TIMESTAMP "1/1/2006 12:00:01 AM"
```

CHILD_VERSION_NAME

Required/Optional: *Optional*

This optional directive specifies the name of a child version to create. The new version will be the child of the version specified by the VERSION directive. If CHILD_VERSION_NAME specifies a version that already exists, an error will be output. After the child version is created, data is read from the Geodatabase using this version instead of from VERSION. No default is provided for this directive, so no child version created in the default case.

Parameter: *<child_version_name>*

Workbench Parameter: *Child Version Name*

Example:

```
GEODATABASE_SDE_CHILD_VERSION_NAME check_out
```

ARCHIVE_WHERE

Required/Optional: *Optional*

This optional directive specifies the where clause used to constrain features read from an archived table. The dates must be in FME date format, and will be converted to the format expected by the underlying database. One or both of the GDB_FROM_DATE or GDB_TO_DATE column names must be specified in order to form a valid where clause. If the GDB_FROM_DATE is not specified, the creation date of the archive will be assumed. If the GDB_TO_DATE is not specified, the current date will be assumed. ArcGIS uses transaction time to record changes to the archive, not valid time. Also note that the features returned will be simple; no complex feature information such as feature-linked annotations or network roles are available.

Workbench Parameter: *Archive Where Clause*

Example of a 'moment' query:

The following demonstrates an example of using the archive where clause to perform a 'moment' query, which will return all features existing in the database as of 9:00 am on May 1st, 2007

```
GEODATABASE_SDE_ARCHIVE_WHERE GDB_FROM_DATE <= 20070501090000 AND GDB_TO_DATE > 20070501090000
```

Example of a 'range' query:

The following demonstrates an example of using the archive where clause to perform a 'range' query, which will return all features inserted after 9:00 am on January 1st, 2007 and updated or deleted before 11:59 pm on December 31st, 2007

```
GEODATABASE_SDE_ARCHIVE_WHERE GDB_FROM_DATE > 20070101090000 AND GDB_TO_DATE < 20071231235959
```

PERSISTENT_CONNECTION

Required/Optional: *Optional*

Specifies whether to create a connection to SDE that persists and can be shared by other Geodatabase SDE Readers and Writers. When set to YES, the connection will remain open until FME shuts down, even if this reader is finished using it. Otherwise, the connection will be closed when the reader is shut down (unless another reader/writer is still using the connection).

Creating a new connection is an expensive operation. Depending on how FME is being used (that is, if there are multiple instances of the Geodatabase SDE Reader/Writer being used), the performance may improve by setting this directive to YES.

Value: YES | NO

Default Value: NO

Workbench Parameter: *Make Connection Persistent*

Example:

```
GEODATABASE_SDE_PERSISTENT_CONNECTION YES
```

SEARCH_ENVELOPE

Required/Optional: *Optional*

Specifies a rectangular area to be used in conjunction with the SEARCH_METHOD directive for extraction of spatial features. This cannot be specified at the same time as a SEARCH_FEATURE is specified.

Parameters:

<min-x>

The minimum x coordinate in the coordinate system of the feature(s) being retrieved.

<min-y>

The minimum y coordinate in the coordinate system of the feature(s) being retrieved.

<max-x>

The maximum x coordinate in the coordinate system of the feature(s) being retrieved.

<max-y>

The maximum y coordinate in the coordinate system of the feature(s) being retrieved.

Workbench Parameter: *Minimum X, Minimum Y, Maximum X, Maximum Y*

Example:

```
GEODATABASE_MDB_SEARCH_ENVELOPE 6190 57239 6310 57549
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

Reader Directives – Personal and File-based Geodatabase

The directive listed below is used when connecting to a Personal Geodatabase (.mdb or .accdb file) or a File-based Geodatabase (directory ending in .gdb).

DATASET

Required/Optional: Required

The file (Personal Geodatabase) or directory (File-based Geodatabase) from which data is to be read.

For Personal Geodatabases, an .mdb or .accdb file is specified when connecting to a Personal Geodatabase.

For File-based Geodatabases, a directory ending in **.gdb** is specified.

Personal Geodatabases Example:

```
GEODATABASE_MDB_DATASET "C:\FME\personalGeodb.mdb"
```

File-based Geodatabases Example:

```
GEODATABASE_FILE_DATASET "C:\FME\montgomery.gdb"
```

Workbench Parameter

Source ESRI Geodatabase (File-based) File

SEARCH_ENVELOPE

Required/Optional: *Optional*

Specifies a rectangular area to be used in conjunction with the SEARCH_METHOD directive for extraction of spatial features. This cannot be specified at the same time as a SEARCH_FEATURE is specified.

Parameters:

<min-x>

The minimum x coordinate in the coordinate system of the feature(s) being retrieved.

<min-y>

The minimum y coordinate in the coordinate system of the feature(s) being retrieved.

<max-x>

The maximum x coordinate in the coordinate system of the feature(s) being retrieved.

<max-y>

The maximum y coordinate in the coordinate system of the feature(s) being retrieved.

Workbench Parameter: *Minimum X, Minimum Y, Maximum X, Maximum Y*

Example:

```
GEODATABASE_MDB_SEARCH_ENVELOPE 6190 57239 6310 57549
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

Improving the Speed of Translations Using the Geodatabase Reader

You can speed up translations involving the Geodatabase Reader by not resolving subtypes and coded value domains.

These operations add extra processing to each row of tables that contain subtypes or coded value domains.

Simple Reader Example

The example below configures a Geodatabase reader to extract features from the dataset testdset located on server tuvok. Only features located on the layer named roads that fall within the specified envelope are read from the Geodatabase.

```
GEODATABASE_SDE_DATASET testdset
GEODATABASE_SDE_SERVER tuvok
GEODATABASE_SDE_USERID joe
GEODATABASE_SDE_PASSWORD bounce
GEODATABASE_SDE_INSTANCE esri_sde8
GEODATABASE_SDE_VERSION sde.DEFAULT
GEODATABASE_SDE_HISTORICAL_VERSION_NAME New Marker
GEODATABASE_SDE_HISTORICAL_VERSION_TIMESTAMP "1/1/2006 12:00:01 AM"
GEODATABASE_SDE_SEARCH_ENVELOPE 601190 543783 611110 5447549
GEODATABASE_SDE_SEARCH_METHOD GEODB_CONTAINS
GEODATABASE_SDE_IDS roads
```

Writer Overview

The underlying format that the Geodatabase writer module uses to store FME features depends on the <WriterType> :

- GEODATABASE_MDB (Personal Geodatabases)
- GEODATABASE_SDE (Enterprise Geodatabases)
- GEODATABASE_FILE (File-based Geodatabases)
- GEODATABASE_SDE_RASTER_DATASET (Enterprise Geodatabases)
- GEODATABASE_FILE_RASTER_DATASET (File-based Geodatabases)

Note that use of the word *table* is meant to refer to both non-spatial tables and feature classes.

The Geodatabase writer module provides the following capabilities:

- **Versioning Support:** The Geodatabase writer provides the ability to write to a specific version of an Enterprise Geodatabase.
- **Update/Delete Support:** The Geodatabase writer provides the ability to update and delete existing features in the Geodatabase.
- **Transaction Support:** The Geodatabase writer provides transaction support that eases the data loading process. Occasionally, a data load operation terminates prematurely due to data difficulties. The transaction support provides a mechanism for reloading corrected data without data loss or duplication.
- **Table Creation:** The Geodatabase writer module uses the information within the FME mapping file to automatically create tables or feature classes as needed. If a feature class is to be created, then certain parameters, such as the grid size and whether or not the features will contain Z values, can be specified. Personal Geodatabase table names are limited to a maximum length of 52 characters.
- **Automated Calculation of Extents and Grid 1 Size:** When writing to a Personal Geodatabase it is possible to set the writer in a mode whereby it will determine what the extents and grid 1 size of the dataset are and will use these values when creating new feature classes. This is not available with the Enterprise Geodatabase or File-based Geodatabase writer; however, valid grid sizes can be calculated and set when writing to Enterprise Geodatabase and File-based Geodatabase.
- **Ignoring Failed Features:** The Geodatabase writer allows the user to continue with a translation, even when a feature would normally cause the translation to fail. The user is given the ability to choose how many failed features are allowed to fail before deciding to halt the translation altogether. Additionally, the user can specify a directory in which to store failed features, which will be stored in the FME Feature Store format.
- **Mosaicking:** The Geodatabase writer provides the ability to mosaic raster data to an existing raster dataset, by overlaying new data on top of old data and building a single, seamless data representation. All rasters being mosaicked must share the same coordinate system, cell size and data type, which is determined by the first raster written to the Geodatabase.

Writer Directives – all Geodatabase Types

This section describes the directives the Geodatabase writer module recognizes when writing to a Feature Class.

Each directive is prefixed by the current <WriterKeyword>_ when it is placed in a mapping file. By default, the <WriterKeyword> for the Geodatabase writer is the same as the <WriterType>.

These directives are only relevant when writing to feature classes, and are not used for raster datasets.

Tip: Due to the complexity and capabilities of a Geodatabase, writing features to one can be difficult, particularly when trying to create feature classes. The Personal Geodatabase writer has a mode whereby it will calculate the extents, scales, and grid sizes on your behalf.

To take advantage of this ability, set the `XY_SCALE` directive to zero. If a non-zero value for the directive `GRID_1` is specified, then the specified value will be used instead of using the grid 1 size calculated by the Personal Geodatabase writer. If no z values are found while calculating the extents, the minimum Z value will be `default_Z - 20,000` and the Z scale will be `10,000`. The value for `default_Z` will be taken from the directive `DEFAULT_Z_VALUE`. Please note that feature classes that exist before the translation will not have their extents and grid 1 size changed.

Even when the writer calculates the x,y,z origins, scales and grid 1 size, the DEF lines can override the calculated values by setting the configuration parameter `GEODB_XYSCALE` to a non-zero value. See `GEODB_XY_SCALE` for more information. (This functionality is not available when using the Enterprise Geodatabase or File-based Geodatabase writer.)

Note: It is important to understand that the initial values assigned to the writer directives, where possible, come from values in the settings box. The DEF line configuration parameters that correspond to the directives will not be assigned values. This prevents the problem whereby the values for the directives are changed, but never get used because they are overridden by the corresponding DEF line configuration parameters. The DEF line configuration

parameters should only be used when the feature classes have differing dimensions (2d or 3d), origins, scales, grid sizes, and measure support.

The directives listed below are used by all Geodatabases.

WRITER_MODE

Required/Optional: *Optional*

Note: For more information on this directive, see the chapter [Database Writer Mode](#).

This statement instructs the Geodatabase writer on the type of mode in which it is to operate. When the WRITER_MODE directive is set to UPDATE or DELETE, the writer will check to see if the attribute `fme_db_operation` exists on the feature. A value of INSERT for this attribute means the feature will be inserted with no extra update processing; a value of UPDATE means the feature will be updated; and a value of DELETE means the feature will be deleted. If the attribute is set to any other value, the translation will be aborted and an error message logged. If the attribute is not set, the mode will be that indicated by the writer directive WRITER_MODE.

To update or delete a feature, the object ID must be on the feature passed into the Geodatabase writer. The object ID must be stored in an attribute with the same name as the object id field in the destination table. For example, if the destination table has an object ID field called `O_ID`, then this attribute must exist and be populated with the correct value on the FME feature.

As with inserting features, updating and deleting features on versioned tables with an ArcSDE requires that the TRANSACTION_TYPE directive be set to VERSIONING.

Parameter: *<writer_mode>*

Values: *UPDATE | DELETE | INSERT*

Default Value: *INSERT*

- INSERT – All features are inserted;
- UPDATE – By default, the features will be updated;
- DELETE – By default, the features will be deleted

Workbench Parameter: *Writer Mode*

Example:

```
GEODATABASE_SDE_WRITER_MODE INSERT
```

TRANSACTION_TYPE

Required/Optional: *Optional*

This statement indicates which transaction mechanism the Geodatabase writer should use. Within ArcGIS, there are currently two transaction mechanisms: edit sessions and (regular) transactions. An edit session corresponds to a long transaction. During an edit session, edits made by other users do not become visible until the edit session is ended. If a translation does not complete successfully and the Geodatabase writer is using an edit session, then all the edits will be discarded.

Values: *VERSIONING | EDIT_SESSION | TRANSACTIONS | NONE*

- VERSIONING: Starts an edit session and then ends it when the translation is finished. This value should be used when writing to a versioned table in an Enterprise Geodatabase.
- EDIT_SESSION: Starts an edit session and then ends it when the translation is finished. This value should be used when edits are made to tables that have custom behavior associated with them.
- TRANSACTIONS: Starts the (regular) transaction mechanism. This can be used only when writing to non-versioned tables that do not have custom behavior.

- NONE: No transaction mechanism is used. This can be used only when writing to non-versioned tables that do not have custom behavior.

Default Value: *TRANSACTIONS*

Note: Currently there is no difference between choosing VERSIONING and EDIT_SESSION.

Workbench Parameter: *Transaction Type*

Example:

```
GEODATABASE_MDB_Transaction_TYPE EDIT_SESSION
```

TRANSACTION

Required/Optional: *Optional*

Transactions do not get used unless the TRANSACTION_TYPE directive is set to TRANSACTIONS. This statement instructs the Geodatabase writer when to begin to write features to the Geodatabase. The writer does not write any features to the Geodatabase until a feature is reached that belongs to <last successful transaction> + 1. Specifying a value of 0 causes the Geodatabase writer to use transactions and to write every feature to the Geodatabase. Normally, the value specified is zero – a positive non-zero value is only specified when a data load operation is being rerun.

If the GEODATABASE_<SDE|MDB|FILE>_TRANSACTION statement is not specified and transactions are being used (TRANSACTION_TYPE is set to TRANSACTIONS) then a default value of 0 is used.

Parameter: <transaction #> This is the transaction number of the last successful transaction. When loading data for the first time, set this value to 0.

Workbench Parameter: *Transaction Number*

Example:

```
GEODATABASE_MDB_TRANSACTION 0
```

TRANSACTION_INTERVAL

This statement tells FME the number of features to be placed in each transaction before a transaction is committed to the database.

If the GEODATABASE_<SDE|MDB|FILE>_TRANSACTION_INTERVAL statement is not specified, then a default value of 1000 is used as the transaction interval.

When TRANSACTION_TYPE is set to VERSIONING or EDIT_SESSION, this value is used to determine how many features to place in each edit operation within the edit session.

Required/Optional

Optional

Values

Default Value: 1000

Mapping File Syntax

```
GEODATABASE_MDB_TRANSACTION_INTERVAL 50
```

Parameter: <transaction_interval> The number of features in each transaction.

Workbench Parameter

Features to Write Per Transaction

Note: The current transaction is committed and a new transaction is started whenever a new table is created or opened, even if the transaction interval was not reached.

HAS_Z_VALUES

Required/Optional: *Optional*

This directive determines whether or not the dataset contains z coordinates. The value of this directive may be overridden by the DEF line parameter of GEODB_HAS_Z_VALUES if a value is specified for it. Valid values for this directive are YES, NO, or AUTO_DETECT. When set to AUTO_DETECT, the writer determines the dimension of the feature class by checking the dimension of the first feature headed for that feature class.

Parameter: *<has_z_values>*

Values: *YES | NO | AUTO_DETECT*

Default Value: *AUTO_DETECT*

Workbench Parameter: *Contains Z Values*

Example:

```
GEODATABASE_MDB_HAS_Z_VALUES yes
```

DEFAULT_Z_VALUE

Required/Optional: *Optional*

This directive determines the z value to use when writing a 2D feature to a 3D feature class.

If the GEODATABASE_<SDE|MDB|FILE>_DEFAULT_Z_VALUE statement is not specified, then a default value of 0 is used.

Parameter: *<default_z_value>* The value to use for the Z coordinate(s) when writing a 2D feature to a 3D feature class.

Values: *real numbers*

Default Value: *0*

Workbench Parameter: *Default Z Value*

Example:

```
GEODATABASE_SDE_DEFAULT_Z_VALUE -11.5
```

X_ORIGIN

Required/Optional: *Optional*

The X-coordinate of the origin for all feature classes (individual origins can be set – see Geodatabase Table Representation) and all feature datasets. This is used as an offset because coordinate data is stored as positive integers, relative to the origin, ranging from 0 to 2147483647 (so if the X origin is set below 0, then the maximum value will also drop, and vice versa).

This directive is used only when creating new feature classes.

Note: This directive is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class/feature dataset being created.

Parameter: *<x_origin>*

Value: *real number*

Default Value: 0

Workbench Parameter: *X Origin*

Example:

```
GEODATABASE_MDB_X_ORIGIN -120.29
```

Y_ORIGIN

Required/Optional: *Optional*

The Y-coordinate of the origin for all feature classes (individual origins can be set – see Geodatabase Table Representation) and all feature datasets. This is used as an offset because coordinate data is stored as positive integers, relative to the origin, ranging from 0 to 2147483647 (so if the Y origin is set below 0, then the maximum value will also drop, and vice versa).

This directive is used only when creating new feature classes.

Note: This directive is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class/feature dataset being created.

Parameter: *<y_origin>*

Value: *real number*

Default Value: 0

Workbench Parameter: *Y Origin*

Example:

```
GEODATABASE_MDB_Y_ORIGIN -32.55
```

Z_ORIGIN

Required/Optional: *Optional*

The Z-coordinate of the origin for all feature classes (individual origins can be set – see Geodatabase Table Representation) and all feature datasets. This is used as an offset because coordinate data is stored as positive integers, relative to the origin, ranging from 0 to 2147483647 (so if the Z origin is set below 0, then the maximum value will also drop, and vice versa).

This directive is used only when creating new feature classes.

Note: This directive is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class/feature dataset being created.

Parameter: *<z_origin>*

Value: *real number*

Default Value: 0

Workbench Parameter: *Z Origin*

Example:

```
GEODATABASE_SDE_Z_ORIGIN 120
```

XY_SCALE

Required/Optional: *Optional*

A scaling conversion factor from world units to integer system units for all feature classes (individual scales can be set – see [Geodatabase Table Representation](#)) and all feature datasets. This is used to specify the level of precision to keep when storing XY coordinates, since all coordinates are stored as integers. Depending on the scale, it changes the precision of the coordinates stored. For example, if you have the coordinate (5.354, 566.35) and you set the XY_SCALE to be 100, then the coordinate stored will be (5.35, 566.35).

When writing to a Personal Geodatabase, if this value is set to 0 then the x,y,z origins and scales will automatically be calculated. These calculated values will be used instead of the values supplied by the writer directives for the x,y,z origins and scales. The grid 1 size will also be calculated, but will only be used if the value for the GRID_1 directive is 0. Even when the writer calculates the x,y,z origins, scales and grid 1 size, the DEF lines can override the calculated values by setting the configuration parameter GEODB_XYSCALE to a non-zero value. See GEODB_XY_SCALE for more information.

This directive is used only when creating new feature classes.

Note: This directive is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class/feature dataset being created.

Parameter: <xy_scale>

Value: real number greater than 0.

When writing to a Personal Geodatabase, zero can be specified, in which case the writer will calculate the extents and scales itself. Only used when creating new feature classes.

Default Value: 100 when writing to an Enterprise Geodatabase; 0 when writing to a Personal Geodatabase

Workbench Parameter: Scale

Example:

```
GEODATABASE_MDB_XY_SCALE 1000
```

Z_SCALE

Required/Optional: *Optional*

A scaling conversion factor from world units to integer system units for all feature classes (individual scales can be set – see [Geodatabase Table Representation](#)) and all feature datasets. This is used to specify the level of precision to keep when storing Z coordinates, since all coordinates are stored as integers. Depending on the scale, it changes the precision of the coordinates stored. For example, if you have the z coordinate 5.354 and you set the Z_SCALE to be 100, then the coordinate stored will be 5.35.

This directive is used only when creating new feature classes.

Note: This directive is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class/feature dataset being created.

Parameter: <z_scale>

Value: real number greater than 0.

Default Value: Enterprise Geodatabase writer: 100; Personal Geodatabase writer: 0

The default value for a Personal Geodatabase is 0 because by default the writer calculates the extents and scales, and therefore ignores the value assigned to this directive.

Workbench Parameter: Z Scale

Example:

```
GEODATABASE_SDE_Z_SCALE 10
```

HAS_MEASURES

Required/Optional: *Optional*

This directive determines whether or not the dataset contains measures. The value of this directive will be overridden by the DEF line parameter GEODB_HAS_MEASURES if a value is specified for it.

If the GEODATABASE_<SDE|MDB|FILE>_HAS_MEASURES statement is not specified, then a default value of NO is used.

This directive is used only when creating new feature classes.

Parameter: <has_measures>

Values: YES | NO

Default Value: NO

Workbench Parameter: *Contains Measures*

Example:

```
GEODATABASE_MDB_HAS_MEASURES yes
```

MEASURES_ORIGIN

Required/Optional: *Optional*

The minimum measures value possible. This is used as an offset because measure data is stored as positive integers (0 to 2147483647) relative to the measures origin. This value is applied to all feature classes and feature datasets, although individual feature classes can override this value using the DEF line parameter GEODB_MEASURES_ORIGIN.

This directive is used only when creating new feature classes.

Note: This directive is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class/feature dataset being created.

Parameter: <measures_origin>

Values: *real number*

Default Value: 0

Workbench Parameter: *Measures Origin*

Example:

```
GEODATABASE_MDB_MEASURES_ORIGIN -412.98
```

MEASURES_SCALE

Required/Optional: *Optional*

A scaling conversion factor from world units to integer system units for all feature classes. Individual feature classes can override this value using the DEF line parameter GEODB_MEASURES_SCALE. This is used to specify the level of precision to keep when storing measures, since all measures are stored as integers. Depending on the scale, it changes the precision of the measures stored. For example, if you have the measure 566.354 and you set the MEASURES_SCALE to be 100, then the measures stored will be 566.35. The value is only used when creating a new feature class.

Note: This directive is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class/feature dataset being created.

Parameter: <measures_scale>

Values: *real number greater than 0*

Default Value: *100*

Workbench Parameter: *Measures Scale*

Example:

```
GEODATABASE_MDB_MEASURES_SCALE 10000
```

GRID_1

Required/Optional: *Optional*

This sets the global grid 1 size for the whole translation. It may be overridden if the DEF line has the setting for GRID{1} parameter. Valid values are real numbers greater than zero.

When using the Enterprise or File-based Geodatabase writer, if the value is 0 and no value is specified for the GRID{1} DEF line parameter (or it is 0) then the grid size will be automatically calculated. The File-based Geodatabase writer will also automatically calculate sizes for grids 2 & 3. This directive is only used when creating new feature classes.

Parameter: *<grid_1_size>*

Values: *real number greater than 0*

Default Value: *Personal or File-based Geodatabase writer: 0; Enterprise Geodatabase writer: 1000*

Workbench Parameter: *Grid 1 Size*

Example:

```
GEODATABASE_MDB_GRID_1 45.6
```

ANNOTATION_UNITS

This directive allows you to specify which map units should be used when creating a new annotation feature class. Its value will be applied to all annotation feature classes created by the writer identified by <WriterKeyword>.

A Multi-Writer should be used when annotation feature classes with different map units need to be created. This directive is not used when opening an existing annotation feature class. If the writer creates an annotation feature class, and the directive ANNOTATION_UNITS is set to unknown_units (the default value), then the writer tries to determine what type of unit the spatial reference uses and sets ANNOTATION_UNITS to the closest unit that is greater than or equal to it (with respect to its meters per unit value). If a local/unknown coordinate system is used, the units are set to meters.

Required/Optional

Optional

Mapping File Syntax

```
GEODATABASE_MDB_ANNOTATION_UNITS nautical_miles
```

Parameter

<annotation_units>

Values

unknown_units (default), decimal_degrees, inches, points, feet, yards, miles, nautical_miles, millimeters, centimeters, meters, kilometers, and decimeters

*** Workbench Parameter**

Annotation Units

SIMPLIFY_GEOM

Note: You can use this directive only if you have installed ArcGIS 9. It will not work with ArcGIS 8.

If it is set to YES, then the geometry being written out is simplified (if it is currently a non-simple geometry).

Required/Optional

Optional

Values

YES | NO (default)

* Workbench Parameter

Simplify Geometry

Writer Directives – All Geodatabase Types

This section describes the directives that the Geodatabase writer module recognizes.

Each directive is prefixed by the current <WriterKeyword>_ when it is placed in a mapping file. By default, the <WriterKeyword> for the Geodatabase writer is the same as the <WriterType>.

IGNORE_FAILED_FEATURE_ENTRY

Required/Optional: *Optional*

This directive tells the Geodatabase writer whether or not it should ignore features that would normally cause the translation to fail. It allows you to ignore features that are topologically incorrect, are not supported by the Geodatabase writer, or conflict with the definition of the table to which it is to be inserted (that is, they are outside of the geometry envelope specified by the feature class). Additionally, polygons, donuts, or aggregates of polygons/donuts that cannot be reoriented will be ignored.

If the GEODATABASE_<SDE|MDB|FILE>_IGNORE_FAILED_FEATURE_ENTRY statement is not specified or given the value NO, then features are not ignored and will cause the translation to fail when encountered. Additionally, the three other associated directives described below will now be put into effect.

Parameter: <ignore_failed_features>

Values: YES | NO

Default Value: NO

Workbench Parameter: *Ignore Failed Features*

Example:

```
GEODATABASE_SDE_IGNORE_FAILED_FEATURE_ENTRY YES
```

MAX_NUMBER_FAILED_FEATURES

Required/Optional: *Optional*

This directive informs the Geodatabase writer of the number of features to ignore before causing a translation to fail due to a problematic feature. (However, the translation may still fail for other reasons.)

This directive is only applicable if IGNORE_FAILED_FEATURE_ENTRY is set to YES.

Parameter: <max_number_failed_features>

Values: *To ignore all failed features: -1; otherwise 0 or a positive integer.*

Workbench Parameter: *Max Number of Features to Ignore*

Example:

```
GEODATABASE_SDE_MAX_NUMBER_FAILED_FEATURES 100
```

DUMP_FAILED_FEATURES

Required/Optional: *Optional*

This directive gives the user the option of storing the failed features to an FFS file so that they can be viewed at a later time. For this statement to be used, GEODATABASE_<SDE|MDB|FILE>_IGNORE_FAILED_FEATURE_ENTRY must be specified and have the value YES.

Parameter: <dump_failed_features>

Values: YES | NO

Workbench Parameter: *Dump Failed Features to File*

Default: NO

FFS_DUMP_FILE

Required/Optional: *Optional*

This directive allows you to choose where the file containing failed features should be stored. The failed features will be stored in the FME Feature Store format. The file will be created automatically, but will only get created if there is a failed feature. If this directive is specified and a failed feature is encountered, then if a file with the same name as given to this directive already exists, it will be overwritten. This directive must be specified if

```
GEODATABASE_<SDE|MDB|FILE>_DUMP_FAILED_FEATURES
```

is specified and has the value YES.

Parameter: <ffs_dump_file>

Values: *path and filename*

If either the path or the filename contains a space, the value must be enclosed in double quotation marks. The filename must end in the extension .ffs.

Workbench Parameter: *Failed Feature Dump Filename*

Example:

```
GEODATABASE_MDB_FFS_DUMP_FILE "c:\user temp\bad features.ffs"
```

BEGIN_SQL{n}

Occasionally you must execute some ad-hoc SQL prior to opening a table. For example, it may be necessary to ensure that a view exists prior to attempting to read from it.

Upon opening a connection to read from a database, the reader looks for the directive <ReaderKeyword>_BEGIN_SQL{n} (for n=0,1,2,...), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the FME_SQL_DELIMITER keyword, embedded at the beginning of the SQL block. The single character following this keyword will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;  
DELETE FROM instructors;  
DELETE FROM people  
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

✳ Workbench Parameter

SQL Statement to Execute Before Translation

END_SQL{n}

Occasionally you must execute some ad-hoc SQL after closing a set of tables. For example, it may be necessary to clean up a temporary view after writing to the database.

Just before closing a connection on a database, the reader looks for the directive `<ReaderKeyword>_END_SQL{n}` (for $n=0, 1, 2, \dots$), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the `FME_SQL_DELIMITER` directive, embedded at the beginning of the SQL block. The single character following this directive will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;
DELETE FROM instructors;
DELETE FROM people
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

✳ Workbench Parameter

SQL Statement to Execute After Translation

Writer Directives – Enterprise Geodatabase

The directives listed below are used when connecting to an Enterprise Geodatabase and writing to either feature classes or raster datasets.

The `CONNECTION_FILE`, `SERVER`, `USERID`, `PASSWORD`, and `INSTANCE` directives operate in the same manner as they do for the Geodatabase reader.

The other writer-specific directives are discussed in the following sections.

CONNECTION_FILE

Required/Optional: *Optional*

The pathname of a connection file for an Enterprise Geodatabase. Having a connection file means that it is not necessary to specify the other directives.

Workbench Parameter: *Connection File*

DATASET

Required/Optional: *Required*

The Enterprise Geodatabase dataset from which data is to be written.

Workbench Parameter: *Destination ESRI Geodatabase (ArcSDE) Dataset*

SERVER

Required/Optional: *Optional*

The server machine where the dataset resides.

Workbench Parameter: *Server*

INSTANCE

Required/Optional: *Optional*

The Enterprise Geodatabase instance to connect to.

Workbench Parameter: *Instance Name*

USERID

Required/Optional: *Optional*

User ID of the Enterprise Geodatabase user.

If the userid and password are missing or not set, then the reader will try and connect with AUTHENTICATION_MODE set to OSA (Operating System Authentication).

Workbench Parameter: *User ID*

PASSWORD

Required/Optional: *Optional*

Password for the user account.

If the userid and password are missing or not set, then the reader will try and connect with AUTHENTICATION_MODE set to OSA (Operating System Authentication).

Workbench Parameter: *Password*

COMMIT_TRANSACTIONS_AFTER_WRITE

Required/Optional: *Optional*

Only valid when the versioning type is set to 'transactions'. Specifies whether to commit transactions at the end of each write operation. If 'yes', transactions will be committed as soon as they are complete. If set to 'no', transactions will be committed at the start of the subsequent transaction. This option has no effect on translations taking place within an edit session.

Value: YES | NO

Default Value: NO

Workbench Parameter: *Commit Transactions at end of Write*

Example:

```
GEODATABASE_SDE_COMMIT_TRANSACTIONS_AFTER_WRITE YES
```

Writer Directives – Enterprise Geodatabase Feature Classes

The directives listed in this section are used when connecting to an Enterprise Geodatabase Feature Class.

These directives are not applicable for writing to raster datasets in an Enterprise Geodatabase.

VERSION

Required/Optional: *Optional*

The name of the version to which features should be written (only applicable on multi-versioned databases).

Parameter: *<version>*

Workbench Parameter: *Transactional Version*

Example:

```
GEODATABASE_SDE_VERSION jim.testversion
```

RECONCILE_AND_POST

Required/Optional: *Optional*

This optional directive reconciles all the changes between the version specified by the writer directive VERSION and its parent version, even those edits made outside of the current translation, and then posts the version to its parent. The reconcile and posting is done at the end of the translation when the Geodatabase writer is being shut down. As a result, if an error occurs during the reconcile or post, then only the reconcile and post changes are undone; all the features that were inserted/updated/deleted prior are still saved. The reconcile and post will only be successful if no conflicts are found. Conflicts must be resolved manually using ESRI's ArcGIS. Upon a successful post, the child version will be deleted or left intact depending on the value of the DELETE_CHILD_AFTER_RECONCILE_AND_POST directive.

Since reconciling is done for all features including those inserted/updated/deleted outside the current translation, then this directive can be used in an empty workspace to simply reconcile and post a child version to its parent. Since this directive reconciles and posts the version specified by VERSION, the value for VERSION must not be SDE.DEFAULT because SDE.DEFAULT has no parent version. When reconciling and posting, the TRANSACTION_TYPE directive must be set to VERSIONING.

Parameter: *<reconcile_and_post>*

Values: *YES | NO*

Default Value: *NO*

Workbench Parameter: *Reconcile and Post*

Example:

```
GEODATABASE_SDE_RECONCILE_AND_POST YES
```

DELETE_CHILD_AFTER_RECONCILE_AND_POST

Required/Optional: *Required*

This directive determines whether to delete the child version following a reconcile and post, including the case where the child and parent version are identical. A value of 'YES' will delete the child version, while a value of 'NO' will leave it intact. The default value is 'YES'.

Parameter: *<delete_child_after_reconcile_and_post>*

Values: *YES | NO*

Default Value: *YES*

Example:

In the example below, the child version will not be deleted after the reconcile and post operation completes.

```
GEODATABASE_SDE__DELETE_CHILD_AFTER_RECONCILE_AND_POST NO
```

Writer Directives – Personal and File-based Geodatabase

The directives listed in this section are used when connecting to a Personal or File-based Geodatabase.

DATASET

Required/Optional: *Required*

For Personal Geodatabase, this is the MS Access file to which data is to be written.

For File-based Geodatabase, this is the directory ending in .gdb.

Workbench Parameter: *Destination ESRI Geodatabase File*

COMPRESS_AT_END

Required/Optional: *Optional*

This directive determines whether the dataset should be compressed after the writer has finished writing features to it. This directive applies to both Personal and File Geodatabases.

Values: YES | NO

Default Value: NO

Workbench Parameter: *Compress Database at End*

Example:

```
GEODATABASE_MDB_COMPRESS_AT_END YES
```

OVERWRITE_GEODB

Required/Optional: *Optional*

If set to YES, deletes the existing database. This directive applies to both Personal and File Geodatabases. Note that a File Geodatabase cannot be overwritten if it is open in the FME Viewer or by ArcMap/ArcCatalog.

Values: YES | NO

Default Value: NO

Workbench Parameter: *Overwrite Geodatabase*

Writer Directives – Geodatabase Raster

This section describes the directives the Geodatabase writer module recognizes when writing raster datasets.

COMPRESSION_TYPE

This directive determines the type of compression to use on raster data being mosaicked.

Required/Optional

Required

Values

LZW | JPEG | JPEG2000 | NONE

Default Value

NONE

Mapping File Syntax

GEODATABASE_FILE_RASTER_DATASET_COMPRESSION_TYPE JPEG2000

* Workbench Parameter

Compression Type

COMPRESSION_QUALITY

This directive determines the compression quality, with higher values indicating improved quality.

Required/Optional

Required

Mapping File Syntax

GEODATABASE_FILE_RASTER_DATASET_COMPRESSION_QUALITY 95

Values

Positive integer (default value is 0)

* Workbench Parameter

Compression Quality

PYRAMID_RESAMPLE_TYPE

This directive determines the resampling type to be used when generating reduced resolution pyramids on the destination raster dataset.

Required/Optional

Required

Mapping File Syntax

GEODATABASE_FILE_RASTER_DATASET_PYRAMID_RESAMPLE_TYPE BILINEAR

Values

NEAREST (default) | BILINEAR | CUBIC

Nearest Neighbor is the default value, which provides the fastest output but the poorest quality. Cubic Convolution provides the best quality, but can reduce performance when writing.

* Workbench Parameter

Pyramid Resample Type

PYRAMID_LEVEL

This directive specifies the number of reduced resolution pyramids to build. More pyramids enable better performance when viewing raster data using FME and ESRI products.

Required/Optional

Required

Values

positive integer

Default Value: 0

Mapping File Syntax

```
GEODATABASE_FILE_RASTER_DATASET_PYRAMID_LEVEL 10
```

* Workbench Parameter

Pyramid Level

Writing Subtypes and Domains

When writing subtypes, the user has two choices: use the integer code or use the code's description. If the integer code is used, then the code is set on an attribute of the same name as the subtype field. For example, if the subtype field is called `road_type`, then an attribute called `road_type` must be populated on the FME feature with the appropriate integer code. If the code's description is used instead, then the description must be supplied on a special attribute called `geodb_subtype_name`. The code corresponding to the description will then be looked up and, once found, will be inserted onto the subtype field.

Similarly, when writing domains, either the integer code may be specified or the code's description. If the integer code is used, then the code is set on an attribute of the same name as the domain field. For example, if the domain field is called `road_type`, then an attribute called `road_type` must be populated on the FME feature with the appropriate integer code. If the code's description is used instead, then the description must be supplied on an attribute called `road_type_resolved`. The code corresponding to the description will be looked up and inserted onto the domain field.

The Geodatabase writer goes through the following steps when writing a subtype:

1. Retrieve the subtype (code) attribute from the feature.
 - a. If the attribute is found but does not contain a valid subtype code (i.e. the code is not one of the possible subtypes or the code is not an integer) then an error is returned and the feature is not written.
 - b. If the attribute was not supplied on the feature, or was supplied but set to the empty string (i.e. it was set to ""), then go to step 2.
2. Retrieve the `geodb_subtype_name` attribute.
 - a. If this attribute is found and not set to the empty string, then the writer attempts to look up the code corresponding to the supplied description. If no code is found, then the description used is not valid, resulting in an error being returned and the feature not being written.
 - b. If this attribute is not supplied or was supplied but set to the empty string (i.e., set to ""), and we're inserting a new feature (not updating an existing feature), then the default code gets written to the subtype field.

Geodatabase Table Representation

The Geodatabase writer requires that every Geodatabase table to which a feature is written be defined within the FME mapping file if the table does not yet exist. If the table exists when writing, then only the table name needs to be specified on the DEF line. When reading from the Geodatabase, it is not necessary that the source tables be defined. Geodatabase tables are specified within the FME mapping file using the `<WriterKeyword>_DEF` statement. It is important to note that when using FME to define a simple table with no spatial column, the definition is merely in this form:

```
<writerKeyword>_DEF <tableName> \  
    [geodb_type geodb_table ] \  
    [<attribute name> <attribute type>]* \  
    ;
```

```
[GEODB_OBJECT_ID_NAME <column_name>] \  
[GEODB_OBJECT_ID_ALIAS <column_name>]
```

The more general format of a table definition – in which a spatial column can be defined (a feature class in ESRI terms) – is given here.

```
<WriterKeyword>_DEF <tableName> \  
  [geodb_type <geodb_type>] \  
  [<attribute name> <attribute type>] * \  
  [GEODB_UPDATE_KEY_COLUMNS <list of column names>] \  
  [GEODB_DROP_TABLE < YES | NO >] \  
  [GEODB_TRUNCATE_TABLE < YES | NO >] \  
  [GEODB_OBJECT_ID_NAME <column_name>] \  
  [GEODB_OBJECT_ID_ALIAS <column_name>] \  
  [ \  
    GEODB_SHAPE_NAME <column_name> \  
    GEODB_SHAPE_ALIAS <column_name> \  
    GEODB_FEATURE_DATASET <feature_dataset_name>] \  
    GEODB_GRID{1} <gridsize> \  
    [GEODB_GRID{2} <grid2size>] \  
    [GEODB_GRID{3} <grid3size>] \  
    [GEODB_GRID{n} <gridsize>] \  
    [GEODB_AVG_NUM_POINTS <num_points>] \  
    [GEODB_XORIGIN <minimum_x>] \  
    [GEODB_YORIGIN <minimum_y>] \  
    [GEODB_XYSCALE <scale>] \  
    [GEODB_HAS_Z_VALUES < YES | NO >] \  
    [GEODB_ZORIGIN <minimum_z>] \  
    [GEODB_ZSCALE <scale>] \  
    [GEODB_HAS_MEASURES < YES | NO >] \  
    [GEODB_MEASURES_ORIGIN <measures_origin>] \  
    [GEODB_MEASURES_SCALE <measures_scale>] \  
    [GEODB_ANNO_REFERENCE_SCALE <anno_ref_scale>] \  
  ]
```

Note that the OBJECTID column and SHAPE column precede all user-defined columns in a new table or feature class created by the Geodatabase writer.

<tableName>

This specifies the name of the Geodatabase table being defined by the <WriterKeyword>_DEF statement. The name must conform to the conventions and restrictions of the underlying RDBMS database. For example, the name cannot have a hyphen (-) in it. Table name case is always preserved.

The following example shows the first portion of the definition for a table named roads.

```
GEODATABASE_SDE_DEF roads . . .
```

geodb_type <geodb_type>

When the Geodatabase writer creates a new table, it looks for the geodb_type specified on the DEF line. For a list of all the valid geodb_type's possible, see [Feature Representation](#). If the geodb_type is not valid or is not found on the DEF line, then the geodb_type attribute will be retrieved from the first feature headed for that table. In the unlikely occurrence that the feature does not have a valid geodb_type or any geodb_type, then a warning will be logged, the writer will assume that the geodb_type is geodb_table, and a non-spatial table will be created. As a result, all features of that feature type will have their geometry ignored.

Attribute Definitions

This section of the <WriterKeyword>_DEF statement defines the attributes for a table.

- The <attribute name> specified within the FME mapping file must obey the following rules:
 - Attribute Name case must conform to the conventions and restrictions of the underlying RDBMS database. When writing to a Personal or File Geodatabase, the case is preserved, but when writing to an Enterprise

Geodatabase whether or not the case is preserved depends on the underlying database (i.e., in Oracle, attribute names are made uppercase, whereas in Microsoft SQL Server at ArcSDE 9.2 or later, case is preserved).

- Attribute Names must obey all length and character restrictions of the Geodatabase. There is a limit of 30 characters when writing to Geodatabase.

Note: If a table is being created and one of the attribute names conflicts with a Geodatabase system field (i.e., the object ID or one of the shape fields), then a different name will be selected for the system field and a warning message logged.

- The <attribute name> definition defines the type and has the form <attribute name> <attribute type>

The supported attribute types are listed in the following table.

FME Attribute Type
smallint
smallint (n)
integer
integer(n)
float
float(n,m)
double
double(n,m)
boolean
char(n)
date
subtype
subtype_codes
range_domain
coded_domain

smallint

This type is used to represent 16-bit integer values.

smallint(n)

This type is used to represent small integer values with less than or equal to n digits.

If an invalid width is specified for this data type then FME will correct the value and output a warning saying that it has done so. A valid value for the width lies between 1 and 5 inclusive.

Note: This only applies to Geodatabase SDE. It will behave like a regular smallint in the other Geodatabases.

integer

This type is used to represent 32-bit integer values.

integer(n)

This type is used to represent integer values with less than or equal to n digits.

If an invalid width is specified for this data type then FME will correct the value and output a warning saying that it has done so. A valid value for the width lies between 1 and 10 inclusive.

Note: This only applies to Geodatabase SDE. It will behave like a regular integer in the other Geodatabases.

float

This type is used to represent 32-bit float values.

float(n,m)

This type is used to represent float values with a precision not exceeding n and a scale not exceeding m.

If an invalid width and/or decimal is specified for this data type then FME will correct the value(s) and output a warning saying that it has done so. A valid value for the width lies between 1 and 6 inclusive. A valid value for the decimal lies between 0 and the value of the width inclusive.

Note: This only applies to Geodatabase SDE. It will behave like a regular float in the other Geodatabases.

double

This type is used to represent 64-bit float values.

double(n,m)

This type is used to represent double values with a precision not exceeding n and a scale not exceeding m.

If an invalid width and/or decimal is specified for this data type then FME will correct the value(s) and output a warning saying that it has done so. A valid value for the width lies between 1 and 38 inclusive. A valid value for the decimal lies between 0 and the value of the width inclusive.

Note: This only applies to Geodatabase SDE. It will behave like a regular float in the other Geodatabases.

boolean

This type is used to represent Boolean values. The possible values are true and false.

char(n)

This type is used to represent character values with a length not exceeding n characters. The FME will read from and write to geodatabases using the UTF-16 encoding.

date

This is used to store and retrieve date information within the Geodatabase.

When a date field is read by the Geodatabase, it is placed in the FME feature with the form HHMMSS, YYYYMMDD, or YYYYMMDDHHMMSS. The time portion is specified using the 24-hour clock. When writing, the date attribute must also be in one of these three forms. These forms are compatible with all other FME dates.

Note: When the Geodatabase writer creates a new table, all the fields, except for the object ID field, will be defined as allowing NULL values.

subtypes

Subtypes allow you to define a subclassification of a table based on a field. For instance, a table named road may have a field called condition which can have values good, bad and miserable. In the Geodatabase, the field must be an integer type of some sort in order to be able to create subtypes on it.

For the Geodatabase writer, subtypes can be created when creating a new table using the following syntax on DEF lines

```
<attribute name> subtype(value1:value2:value3:.....valuen)
```

or

```
<attribute name> subtype_codes(code1:val1:code2:val2:...:coden:valn)
```

Note: The argument list (i.e., everything between the parentheses) must be colon-separated and must be encoded using a special XML-like encoding. Workbench automatically encodes the list properly. To encode the argument list manually within a mapping file or FME Objects, see *Substituting Strings in Mapping Files* in the FME Fundamentals on-line help file. You can also contact Safe Software for assistance.

In the first case, descriptions can be supplied as strings, in which case codes are generated by the Geodatabase writer starting at zero. In the second case, the input list consists of pairs of codes and corresponding descriptions.

The first code in the list will be used as the default subtype code. In the first case where only descriptions are specified, the code created for value1 will be used as the default subtype code. For instance, if the DEF line is specified as follows

```
subtype_codes(1:a:3:b:4:c:5:d)
```

then 1 will be used as the default code (which maps to a).

The following restrictions are applied when subtypes are created:

- Each table can have only one subtype.
- All the codes have to be unique and valid integers.
- All the value,description pairs have to be unique.
- You cannot add subtypes to an existing table. If you do, the DEF line definition will be ignored and the table will use the existing subtype, if one exists. If a subtype does exist on the table, then a comparison will be made between the DEF line definition and the existing subtype's definition. A warning message will be logged if they are different.

When writing features, the subtype attribute must contain the code (which is stored as an integer by Geodatabase). To supply the description instead of the code for a feature, use the special attribute `geodb_subtype_name`. If the subtype attribute is not specified, then the writer will look for the `geodb_subtype_name` attribute and will convert the description to its corresponding code. See the section **Writing Subtypes and Domains** for more information.

Example:

```
<writerKeyword>_DEF road \  
  type geodb_polyline \  
  condition subtype_codes(0:good:1:bad:2:miserable) \  
  ..... \  
  .....
```

domains

The domains can be specified at the DEF lines using the following syntax:

```
<attribute name> range_domain(domain_name:field_type:min_val: max_val)  
<attribute name> coded_domain(domain_name:field_type:name_1:value_1: ....  
  name_n:value_n)
```

This syntax is used to create new domains along with the new field. The domains defined as above are added to both the workspace and the field of the table. If the domain name already exists in the workspace, a comparison is made against the existing definition. A warning is issued if they differ, and the existing domain is used.

Note: The argument list (i.e., everything between the parentheses) must be colon-separated and must be encoded using a special XML-like encoding. Workbench automatically encodes the list properly. To encode the argument list manually within a mapping file or FME Objects, see *Substituting Strings in Mapping Files* in the FME Fundamentals on-line help file. You can also contact Safe Software for assistance.

Example 1:

To define a domain with the following code and values for a float field type:

Code	Values
1.2	val1
3.6	val2,val3
4.5	test "quotes" here

you would use the following format:

```
coded_domain(floatCodedDom:float:1.2:val1:3.6:val2<comma>val3:4.5:
<quote>test<space><quote><quote>quotes<quote><quote><space>here<quote>)
```

To use an existing domain, you can use the following short syntax:

```
range_domain(domain_name)
coded_domain(domain_name)
```

An error will be generated if the domain does not exist in the workspace or is not defined elsewhere using the long syntax (i.e., the syntax specified at the beginning of this section) for domains.

Note: If the same new domain is referenced multiple times in the translation, the long syntax form only needs to be specified on one attribute; all other instances can use the short form. It does not matter on which attribute the long form is specified within the mapping file or workspace.

To specify the length of a text field while specifying an existing domain, you can use the following variant of the short syntax:

```
coded_domain(domain_name:char<openparen><textsize><closeparen>)
```

Note: The <openparen> and <closeparen> are the result of applying FME's XML-like way of escaping certain characters. Contact Safe Software if more information is needed, keeping in mind that Workbench automatically performs this encoding when creating new domains and subtypes.

For example,

```
coded_domain(textDomain:char<openparen>50<closeparen>)
```

It is important to note:

- The variant of this short syntax is allowed for text fields only.
- Since ArcGIS allows text fields to have only coded value domains, using this field type with range_domain will result in an error.
- If no text length is specified for a text field, that is, if the syntax used is:

```
coded_domain(domain_name)
```

then FME will insert a default length of 254 characters.

Example 2:

A complete DEF line example is shown below:

```
GEODATABASE_MDB_OUT_DEF Q1_LINE \
  geodb_type geodb_polyline \
  GEODB_DROP_TABLE YES \
  GEODB_TRUNCATE_TABLE NO \
  GEODB_OBJECT_ID_NAME Object_ID \
  GEODB_OBJECT_ID_NAME Object_ID \
  GEODB_OBJECT_ID_ALIAS "object ID" \
  GEODB_SHAPE_NAME Shape \
  GEODB_SHAPE_ALIAS Shape \
  GEODB_CONFIG_KEYWORD DEFAULTS \
  GEODB_FEATURE_DATASET "" \
  GEODB_GRID{1} ${_GEODBOutGrid1} \
  GEODB_AVG_NUM_POINTS "" \
```

```

GEODB_HAS_MEASURES NO \
GEODB_HAS_Z_VALUES $(_GEODBOutDimension) \
GEODB_XORIGIN "" \
GEODB_YORIGIN "" \
GEODB_ZORIGIN "" \
GEODB_XYSCALE "" \
GEODB_ZSCALE "" \
GEODB_ANNO_REFERENCE_SCALE "" \
geodb_oid integer \
igds_class double \
igds_color double \
igds_graphic_group double \
igds_style double \
igds_weight double \
object_ID integer \
Shape_Length double \
property_range_domain(intDomain:integer:0:100) \
testVal_range_domain(realDomain:double:0.5:25.5) \
textVal_coded_domain(textVal:char<openparen>50<closeparen>:a:a11:b:bad) \
floatVal_coded_domain(floatCodedDom:float:1.2:val1:3.6:val2<comma>val3:
4.5:<quote>test<space><quote><quote>quotes<quote><quote><space>here<quote>)

```

Configuration Parameters

There are a number of configuration parameters in the GEODATABASE_<SDE|MDB|FILE>_DEF line that are used to define characteristics of a feature class. They are described in the following table. *The configuration parameters are only needed when defining a new table.* If the table already exists, it is sufficient to have the following line:

```
<writerKeyword>_DEF <tableName>
```

When creating a new table, configuration parameters found on the table definition will override the equivalent writer directives. However, the parameters related to setting grid sizes and x,y,z origins and scales will override the writer directives only if the configuration parameter GEODB_XYSCALE is specified and not equal to zero.

Note: It is important to understand that the values from the settings box are assigned to the writer directives, not to the configuration parameters described in this section. This was done to prevent the problem whereby the values for the writer directives get changed in the workspace/mapping file, but never get used because they are overridden by the corresponding configuration parameters.

Parameter	Contents
geodb_type	A valid geodb_type must be specified in order to determine what type of table to create. Please check the section Feature Representation for valid values of geodb_type. On DEF lines, setting the geodb_type to geodb_arc is equivalent to geodb_polyline and setting it to geodb_ellipse is equivalent to geodb_polygon.
GEODB_UPDATE_KEY_COLUMNS	<p>The list of field names that are used by the writer when it is updating or deleting a feature. The value is a comma-separated list of the fields which are matched against the corresponding FME attributes' values to specify which rows are to be updated using the other attribute values. If a corresponding attribute is not on the FME feature, then the value NULL will be used in the query for that particular column.</p> <p>In general, this should identify a unique feature but can also be used to update/delete multiple features if desired. If an update/delete is being performed and no value is assigned to this parameter, then the writer will use the object ID column to perform the update and the corresponding object ID attribute must be present on the FME feature.</p> <p>The following example sets the update fields to be COUNTRY and CAP-</p>

Parameter	Contents
	ITAL: GEODB_UPDATE_KEY_COLUMNS COUNTRY,CAPITAL
GEODB_DROP_TABLE	<p>Specifies that the writer drop the table before writing, and create a new one. If the table does not exist, it will be created when the data is written.</p> <p>The following example sets the drop table flag to false.</p> <pre>GEODB_DROP_TABLE NO</pre> <p>Default: NO</p> <p>Values: YES NO</p>
GEODB_TRUNCATE_TABLE	<p>Specifies that the writer truncate the table before writing. If the table does not exist, it will be created when the data is written.</p> <p>The following example sets the truncate table flag to false.</p> <pre>GEODB_TRUNCATE_TABLE NO</pre> <p>Default: NO</p> <p>Values: YES NO</p>
GEODB_OBJECT_ID_NAME	<p>The name of the column containing object IDs for the current table. The name of the column must not contain any spaces. If this parameter is not found on the DEF line, then the column will be given the name Object_ID (Universal Translator) or OBJECTID (Workbench). If the value conflicts with a user attribute, then the writer will change the value for this field (by appending a numeric suffix) and log a warning.</p> <p>Note: Due to the way annotation feature classes are created since the inception of ArcGIS 9.0, this parameter has no effect when run on computers with ArcGIS 9.0 or newer.</p> <p>The following example defines the column name to hold object IDs to be OBJECT_IDENT:</p> <pre>GEODB_OBJECT_ID_NAME OBJECT_IDENT</pre>
GEODB_OBJECT_ID_ALIAS	<p>The alias for the object IDs column for the current table. The alias is used in ArcMap (and possibly in other ArcGIS products) when viewing data, the object ID column will be labeled by its alias. If this parameter is not found on the DEF line, then the alias will be given the value Object ID (Universal Translator) or OBJECTID (Workbench).</p> <p>The following example defines the alias for the object ID column name to be Primary ID. Notice the alias name must be surrounded by quotation marks ("").</p> <pre>GEODB_OBJECT_ID_ALIAS "Primary ID"</pre>
GEODB_SHAPE_NAME	<p>The name of the column containing the shape data for features in the current feature class. This applies only to feature classes. The name of the column must not contain any spaces. If this parameter is not found on the DEF line, then the column will be given the name Shape (Universal Translator) or SHAPE (Workbench). If the value, or one of its corresponding LENGTH or AREA fields, conflicts with a user attribute, then the writer will change the value for this field (by appending a numeric</p>

Parameter	Contents
	<p>suffix) and log a warning.</p> <p>Note: Due to the way annotation feature classes are created since the inception of ArcGIS 9.0, this parameter has no effect when run on computers with ArcGIS 9.0 or newer.</p> <p>The following example defines the shape data column name to be Geometry:</p> <p>GEODB_SHAPE_NAME Geometry</p>
<p>GEODB_SHAPE_ALIAS</p>	<p>The alias for the shape data column. When viewing data in ArcMap (and possibly in other ArcGIS products), the shape data column will be labeled by its alias. If this parameter is not found on the DEF line, then the alias will be given the value Shape (Universal Translator) or SHAPE (Workbench).</p> <p>The following example defines the alias for the shape data column name to be "Shape Geometry". (Note that the alias name must be enclosed in quotation marks.)</p> <p>GEODB_SHAPE_ALIAS "Shape Geometry"</p>
<p>GEODB_FEATURE_DATASET</p>	<p>The name of the feature dataset to which a feature class belongs. If this parameter is not specified on the DEF line, then the feature class is created as a standalone feature class. If this parameter is specified, then the feature class will be made part of the specified feature dataset, and if that dataset does not exist then it will be created. If the feature dataset is created by FME, then the values for the false origin and scale are taken from the respective writer directives (that is, the X_ORIGIN, Y_ORIGIN, Z_ORIGIN, XY_SCALE, and Z_SCALE). If a directive is not specified, then the default value for that directive is used. However, when writing to a File-based Geodatabase, these directives do not get used. Instead, default values, based on the coordinate system set, are used for the origin and resolution.</p> <p>The following example specifies that the feature class belongs to a feature dataset named Town:</p> <p>GEODB_FEATURE_DATASET Town</p>
<p>GEODB_GRID{1}</p>	<p>This parameter specifies the size of the spatial index in the coordinate system of the layer. It is only applicable if a feature class is being created and is only used if the GEODB_XYSCALE parameter contains a non-zero value. The value must be a real number greater than zero.</p> <p>When using the Enterprise Geodatabase or File-based Geodatabase writer, if the value is 0 or is not specified, and the value for the GRID_1 directive is 0 then the grid size will be automatically calculated. The File-based Geodatabase writer will also automatically calculate sizes for grids 2 & 3.</p>

Parameter	Contents
	<p>The following example defines a grid size of 200 for the level 1 grid:</p> <pre>GEODB_GRID{1} 200</pre>
GEODB_GRID{2}	<p>This optional parameter defines the level 2 grid size. If it is not desired, then the value should not be specified or should be set to 0. Before the level 2 grid size can be specified, the level 1 grid size must be specified. This parameter must be a real number greater than zero.</p> <p>The following example defines a grid size of 600 for the level 2 grid:</p> <pre>GEODB_GRID{2} 600</pre>
GEODB_GRID{3}	<p>This optional parameter defines the level 3 grid element size. If it is not desired, then the value should not be specified or should be set to 0. Before the level 3 grid size can be specified, the level 2 grid size must be specified. This parameter must be a real number greater than zero.</p> <p>The following example defines a grid size of 1800 for the level 3 grid:</p> <pre>GEODB_GRID{3} 1800</pre>
GEODB_AVG_Num_points	<p>This optional field specifies the estimated average number of points per feature. It is used in the creation of the spatial index for the feature class. If this configuration parameter is not specified, then a default value will be assigned to the feature class, depending on its geometry type. If the geometry is a point the default value will be 1; if the geometry is a multipoint, the default will be 10; if the geometry is a polyline, the default value will be 20; if the geometry is a polygon, the default value will be 40. This parameter must be an integer.</p> <p>For example, if a new feature class called Roads is to be created and we believe that the average number of points for a feature from the Roads feature class will be 3, then on the DEF line for the Roads feature class, we would find the following:</p> <pre>GEODB_AVG_NUM_POINTS 3</pre>
GEODB_CONFIG_KEYWORD	<p>This optional field can be used to specify the configuration parameters of the table to be written. This directive can be used effectively if the database is accessed through ArcSDE. It is ignored when writing to Personal database. For example</p> <pre>GEODB_CONFIG_KEYWORD TEST_CONFIG</pre> <p>The default value is DEFAULTS.</p>
GEODB_XORIGIN	<p>This is the same as the writer directive X_ORIGIN above, except this configuration parameter applies only to the table whose DEF line it is on. This parameter is only used if the GEODB_XYSCALE parameter contains a non-zero value and is only used by standalone feature classes. If a feature class is part of a feature dataset, then this parameter is ignored. The value must be a real number.</p> <p>For example:</p> <pre>GEODB_XORIGIN -53040</pre> <p>Note: This parameter is not used by the File-based Geodatabase writer as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class being created.</p>

Parameter	Contents
GEODB_YORIGIN	<p>This is the same as the writer directive Y_ORIGIN above, except this configuration parameter applies only to the table whose DEF line it is on. This parameter is only used if the GEODB_XYSCALE parameter contains a non-zero value and is only used by standalone feature classes. If a feature class is part of a feature dataset, then this parameter is ignored. The value must be a real number.</p> <p>For example:</p> <pre>GEODB_YORIGIN 1043.89</pre> <p>Note: This parameter is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class being created.</p>
GEODB_XYSCALE	<p>This is the same as the writer directive XY_SCALE above, except this configuration parameter applies only to the table whose DEF line it is on. If this parameter does not appear or is set to 0, all the x,y,z origins and scales and the grid 1 size are taken from the writer directives, even if some of these values are supplied on the DEF line. If this value is set to a non-zero value, then all the DEF line parameters for the x,y,z scales and origins and grid 1 size must be specified. This parameter is only used by standalone feature classes. If a feature class is part of a feature dataset, then this parameter is ignored. The value must be a real number greater than or equal to 0.</p> <p>For example:</p> <pre>GEODB_XYSCALE 1000</pre> <p>Note: This parameter is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class being created.</p>
GEODB_HAS_Z_VALUES	<p>This optional parameter specifies whether or not the features will contain Z values. The only valid values are YES and NO.</p> <p>For example:</p> <pre>GEODB_HAS_Z_VALUES YES</pre> <p>Because Geodatabase does not allow mixed 2D and 3D features in the same feature class, it is best to put a value of YES for this parameter if you have mixed dimensions. The 2D features will be forced to 3D.</p> <p>If you get an error message saying your feature class does not support Z values, you cannot simply add this configuration parameter to your DEF line and perform the translation again. Since this parameter is only used when a feature class is created, rather than an existing feature class being opened, you must either delete the existing feature class or translate to a new Geodatabase.</p> <p>If this configuration parameter is not specified, then a default value of NO will be assumed.</p>
GEODB_ZORIGIN	<p>This is the same as the writer directive Z_ORIGIN above, except this configuration parameter applies only to the table whose DEF line it is on. This parameter is only used if the GEODB_XYSCALE parameter contains a non-zero value and is only used by standalone feature classes. If a feature class is part of a feature dataset, then this parameter is ignored.</p>

Parameter	Contents
	<p>The value must be a real number.</p> <p>For example:</p> <p style="padding-left: 40px;">GEODB_ZORIGIN 0</p> <p>Note: This parameter is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class being created.</p>
<p>GEODB_ZSCALE</p>	<p>This is the same as the writer directive Z_SCALE above, except this configuration parameter applies only to the table whose DEF line it is on. This parameter is only used if the GEODB_XYSCALE parameter contains a non-zero value and is only used by standalone feature classes. If a feature class is part of a feature dataset, then this parameter is ignored. The value must be a real number greater than 0.</p> <p>For example:</p> <p style="padding-left: 40px;">GEODB_ZSCALE 10</p> <p>Note: This parameter is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class being created.</p>
<p>GEODB_HAS_MEASURES</p>	<p>This optional field specifies whether or not the features will contain measures. The only valid values are YES and NO. If this configuration parameter is not specified, then a default value of NO will be assumed.</p> <p>For example:</p> <p style="padding-left: 40px;">GEODB_HAS_MEASURES YES</p>
<p>GEODB_MEASURES_ORIGIN</p>	<p>This is the same as the writer directive MEASURES_ORIGIN above, except this configuration parameter applies only to the table whose DEF line it is on. If this parameter does not appear, then the value for the directive MEASURES_ORIGIN is taken. This parameter is only used by standalone feature classes. If a feature class is part of a feature dataset, then this parameter is ignored. This parameter must be a real number.</p> <p>For example:</p> <p style="padding-left: 40px;">GEODB_MEASURES_ORIGIN -232</p> <p>Note: This parameter is not used by the File-based Geodatabase writer, as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class being created.</p>
<p>GEODB_MEASURES_SCALE</p>	<p>This is the same as the writer directive MEASURES_SCALE above, except this configuration parameter applies only to the table whose DEF line it is on. If this parameter does not appear, then the value for the directive MEASURES_SCALE is taken. This parameter is only used by standalone feature classes. If a feature class is part of a feature dataset, then this parameter is ignored. This parameter must be a real number greater than zero.</p> <p>For example:</p> <p style="padding-left: 40px;">GEODB_MEASURES_SCALE 5489.6</p> <p>Note: This parameter is not used by the File-based Geodatabase writer,</p>

Parameter	Contents
	<p>as default values are used for the domain and resolution. The default values used are dependent on the coordinate system of the feature class being created.</p>
<p>GEODB_ANNO_REFERENCE_SCALE</p>	<p>This optional field specifies what reference scale to use when creating an annotation feature class. The reference scale determines the scale at which the text's size, on the screen, is the size indicated on each annotation feature. When the scale is larger in value than the reference scale, then the text appears smaller than that indicated on the annotation feature and vice versa. If no value is specified for this field, then FME uses first annotation feature for the annotation feature class. If the feature contains the attribute geodb_text_ref_scale then the value for the attribute is used as the reference scale. If the attribute doesn't exist, then the default value for the attribute is used, which is 1.</p> <p>For example:</p> <p>GEODB_ANNO_REFERENCE_SCALE 12000</p>
<p>GEODB_COMPRESSION_TYPE</p>	<p>This is the same as the writer directive COMPRESSION_TYPE above, except this configuration parameter applies only to the table whose DEF line it is on. If this parameter does not appear, then the value for the directive COMPRESSION_TYPE is taken.</p> <p>For example:</p> <p>GEODB_COMPRESSION_TYPE LZ77</p> <p>Note: This parameter is only applicable when writing to raster datasets.</p>
<p>GEODB_COMPRESSION_QUALITY</p>	<p>This is the same as the writer directive COMPRESSION_QUALITY above, except this configuration parameter applies only to the table whose DEF line it is on. If this parameter does not appear, then the value for the directive COMPRESSION_QUALITY is taken.</p> <p>For example:</p> <p>GEODB_COMPRESSION_QUALITY LZ77</p> <p>Note: This parameter is only applicable when writing to raster datasets.</p>
<p>GEODB_PYRAMID_RESAMPLE_TYPE</p>	<p>This is the same as the writer directive PYRAMID_RESAMPLE_TYPE above, except this configuration parameter applies only to the table whose DEF line it is on. If this parameter does not appear, then the value for the directive PYRAMID_RESAMPLE_TYPE is taken.</p> <p>For example:</p> <p>GEODB_PYRAMID_RESAMPLE_TYPE BILINEAR</p> <p>Note: This parameter is only applicable when writing to raster datasets.</p>
<p>GEODB_PYRAMID_LEVEL</p>	<p>This is the same as the writer directive PYRAMID_LEVEL above, except this configuration parameter applies only to the table whose DEF line it is on. If this parameter does not appear, then the value for the directive PYRAMID_LEVEL is taken.</p> <p>For example:</p> <p>GEODB_PYRAMID_LEVEL 5</p> <p>Note: This parameter is only applicable when writing to raster datasets.</p>

Creating Dimension Feature Classes

Currently, there is no way to specify any dimension-specific settings when using the Geodatabase writer to create a dimension feature class. As a result, default settings are used. The default dimension style created by the writer is equivalent to the default one ArcCatalog creates. Likewise, the reference scale and map units are set to the default values used by ArcCatalog, 1 and decimal degrees, respectively. If the dimension feature class is created within a feature dataset, it will inherit the units of the feature dataset rather than being set to decimal degrees.

If the default settings are not sufficient, the dimension feature class should be created before the translation using ArcCatalog. ArcCatalog makes it easy to import dimension styles from existing feature classes and then change them as required.

Creating Geometric Networks

Currently, FME cannot be used to create geometric networks, or the feature classes participating in them. These should be created before the translation using ArcCatalog. FME can then be used to either populate these existing feature classes with point and line data, or alternately to create simple point and line feature classes, with the geometric network created as a post-processing step.

Creating Relationship Classes

Currently, FME cannot be used to create relationship classes. These should be created before the translation using ArcCatalog. FME can then be used to populate these existing relationship classes from source origin and destination features that are related to one another.

Improving the Speed of Translations Using the Geodatabase Writer

You can speed up translations involving the Geodatabase writer by lengthening the interval between committing transactions. Committing transactions is an expensive operation and therefore it is recommended that you make the transaction interval as big as possible (or alternatively, if transactions are not needed, then you can turn them off). In speed tests performed at Safe Software Inc., changing the transaction interval from 500 (the default) to 1000 resulted in a specific translation being 2.5% faster. Changing the transaction interval to 5000 resulted in the same translation running 5.5% faster. Turning off transactions resulted in an improvement of either 12% or 19%. The performance advantages of changing the transaction interval or of turning transactions off will differ between various datasets.

Another way in which the speed of writing features can be increased is by creating all the feature datasets, feature classes, and non-spatial tables ahead of time so that the Geodatabase writer only needs to open them, rather than create them.

Tips for Using the Geodatabase Writer

- When writing to a Geodatabase, those fields that are not assigned values (i.e., no attribute exists on the feature for that field) will be assigned a NULL value if the field allows NULL values.
- When writing to Geodatabase, if your translation fails with the ArcObjects message number of -2147216072 (FDO_E_SE_DB_IO_ERROR), there may be several reasons for this, such as:
 - The database that you are writing to may be out of space.
 - One of the values that you are trying to insert is too large for the underlying database to handle.
 - One of the values is too large for the data type specified. Try changing the data types of your columns.
 - One of your column names is invalid, possibly due to characters that are not considered capitalized. Try renaming the faulty column.

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see [About Feature Attributes](#)), this format adds the format-specific attributes described in this section.

The Geodatabase modules make use of the following special attribute names.

Attribute Name	Contents
geodb_type	<p>The type of geometric entity stored within the feature. The valid values are listed below:</p> <ul style="list-style-type: none"> geodb_annotation geodb_arc geodb_attributed_relationship geodb_complex_edge geodb_complex_junction (read-only) geodb_dimension geodb_ellipse geodb_metadata geodb_multipatch geodb_multipoint geodb_point geodb_polygon geodb_polyline geodb_raster geodb_relationship geodb_simple_edge geodb_simple_junction geodb_table
geodb_measures	<p>This is present for features that have measures when reading. To write measures, you simply build this list with one value for each vertex in the feature being written. This is a comma-separated list of floating values that correspond to the vertex measures. The first value is for the first vertex, second for the second, and so on.</p>
geodb_feature_is_simple (Reader only)	<p>Indicates whether or not the geometry is simple. Only present on spatial features and when ArcGIS 9 is installed.</p>
geodb_subtype_name	<p>When reading, if RESOLVE_SUBTYPE_NAMES is set to YES, then the value corresponding to the subtype code is stored in this attribute. When writing to a table with subtypes and no integer value is supplied for the subtype field, then specifying this attribute with an actual value (i.e. not "") will trigger a look-up for the code corresponding to the value supplied in this attribute. If the code is found, it will be written to the subtype field; otherwise the feature will fail to be written. If the value specified was the empty string (i.e., "") then the default value will get used.</p>

Attribute Name	Contents
<attribute-name>_resolved	When reading, if RESOLVE_DOMAINS is set to YES, then the description corresponding to the domain code is stored in this attribute. When writing to a field associated with a coded value domain, specifying this attribute instead of <attribute-name> will trigger a look-up for the corresponding code. If the code is found, it will be written to <attribute-name>.

Features read from, or written to, the Geodatabase also have an attribute for each column in the database table.

Tables

geodb_type: geodb_table

Features with this value consist of no coordinates. This value is used by both the reader and the writer.

Points

geodb_type: geodb_point

Features with this value are point features. This value is used by both the reader and the writer.

Polylines

geodb_type: geodb_polyline

Features with this value are features or multi-part features consisting of one or more linear features (that are linked together). This type of linear feature is allowed to touch or cross over itself. This value is used by both the reader and the writer.

Multipoints

geodb_type: geodb_multipoint

Features with this value are multi-part features consisting of points. This value is used by both the reader and the writer.

Note: If a multipoint feature is written to an existing point feature class, then the feature will be split up and each point written out as a separate feature. Each new feature will have the same attribution as the original feature; the only difference will be the geometry. If a multipoint feature is written to a point feature class that has not yet been created, then a multipoint feature class will be created instead of a point feature class.

Arcs

geodb_type: geodb_arc

Features with this value contain either a circular arc or an elliptical arc. Arc features are like ellipse features, except two additional angles control the portion of the ellipse boundary which is drawn.

Tip: The Function @Arc() can be used to convert an arc to a line. This is useful for representing arcs in systems that do not support them directly.

Note: During reading, if an arc is encountered that has become a line, FME will create a line feature out of it rather than creating an arc feature and its geodb_type will be set to geodb_polyline instead of geodb_arc

Attribute Name	Contents
geodb_primary_axis	The length of the semi-major axis in ground units. If the arc is circular, this will be the same as the geodb_

Attribute Name	Contents
	secondary_axis.
geodb_secondary_axis	The length of the semi-minor axis in ground units. If the arc is circular, this will be the same as the geodb_primary_axis.
geodb_start_angle	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of start_angle. Default: 0
geodb_sweep_angle	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of sweep_angle.
geodb_rotation	The rotation of the major axis. The rotation is measured in degrees counter clockwise up from horizontal. Default: 0
geodb_arc_start_x Available only with classic geometry.	The x coordinate of the start point of the arc. This attribute is only present on classic geometry. On enhanced geometry, this information is stored within the geometry itself.
geodb_arc_start_y Available only with classic geometry.	The y coordinate of the start point of the arc. On enhanced geometry, this information is stored within the geometry itself.
geodb_arc_start_z Available only with classic geometry.	The z coordinate of the start point of the arc. Only applicable when dealing with 3D arcs. On enhanced geometry, this information is stored within the geometry itself.
geodb_arc_end_x Available only with classic geometry.	The x coordinate of the end point of the arc. On enhanced geometry, this information is stored within the geometry itself.
geodb_arc_end_y Available only with classic geometry.	The y coordinate of the end point of the arc. On enhanced geometry, this information is stored within the geometry itself.
geodb_arc_end_z Available only with classic geometry.	The z coordinate of the end point of the arc. Only applicable when dealing with 3D arcs. On enhanced geometry, this information is stored within the geometry itself.

Attribute Name	Contents
geodb_original_arc_direction Available only with classic geometry.	Always used in conjunction with the start and end points of the arc. This attribute indicates whether the arc goes clockwise beginning from the start point, or counterclockwise. Default: counterclockwise

Ellipses

geodb_type: geodb_ellipse

Ellipse features are point features used to represent both circles and ellipses. The point serves as the centre of the ellipse.

Tip: The Function @Arc() can be used to convert an ellipse to a polygon. This is useful for representing ellipses in systems that do not support them directly.

Attribute Name	Content
geodb_primary_axis	The length of the semi-major axis in ground units.
geodb_secondary_axis	The length of the semi-minor axis in ground units. Default: the value of geodb_primary_axis
geodb_rotation	The rotation of the major axis. The rotation is measured in degrees counterclockwise up from horizontal. Default: 0

Polygon

geodb_type: geodb_polygon

Features with this value are features or multi-part features consisting of polygons and/or donut polygons.

Multipatch

geodb_type: geodb_multipatch

Features with this value consist of a 3D geometry, used to represent the outer surface of features which occupy a discrete area or volume in three-dimensional space. Geodatabases directly support 3D polygonal faces, triangle fans, triangle patches and triangle strips. By definition, the surfaces which compose a multipatch do not need to be connected. This value is used by both the reader and the writer.

Multipatches support appearances, but only one appearance per surface is supported. For two-sided surfaces, the writer will favor non-default appearances with textures, with the front side taking precedence over the back side. If this behavior is not desired, the writer can be forced to choose the front side by using the AppearanceRemover transformer in FME Workbench to remove all back appearances.

For writing, all 3D geometry types are supported. Any types of 3D geometry which are not directly supported in a geodatabase (e.g., solids) are decomposed into a set of 3D polygonal faces prior to writing.

Annotation

geodb_type: geodb_annotation

Feature-linked Annotation

Annotations are separate features but can be linked to other features through feature-linked annotations. Feature-linking occurs when there is a relationship between an annotation feature class and some other feature class. The attribute `geodb_linked_feature_id` controls which annotations are linked to which features.

Note: Feature-linked annotations can be read and written using FME if the necessary feature classes and relationships are created and set up before the translation. Currently, it is only possible to read or insert feature-linked annotations, not update or delete them.

If the feature to be linked to by the annotation has not yet been written, then it is possible for the Geodatabase writer to write the feature, retrieve the object ID of the new feature, and then write the annotation feature linking to it, supplying the correct value for `geodb_linked_feature_id`. This is accomplished by supplying attributes (Geodatabase-specific annotation attributes and user-defined attributes) from the annotation feature on the non-annotation feature; of the annotation feature attributes, only `geodb_text_string` must be specified. In addition, the following attributes must also be specified:

1. `geodb_text_feat_class_name` – specifies the annotation feature class to which the annotation will be written
2. `geodb_text_x_coord` & `geodb_text_y_coord` – specifies the location of the annotation

The result is that the one FME feature contains enough information to write two features: one annotation feature and one non-annotation feature.

To specify multiple feature-linked annotations, the list attribute `geodb_text{i}` must be used to group together each annotation's attributes. There are 4 mandatory annotation attributes that must be present for each annotation in the list:

1. `geodb_text{i}.geodb_text_string` - the text string to write
2. `geodb_text{i}.geodb_text_feat_class_name` - the name of the destination annotation feature class
3. `geodb_text{i}.geodb_text_x_coord` - the x coordinate of the annotation's location
4. `geodb_text{i}.geodb_text_y_coord` - the y coordinate of the annotation's location

Other than these 4 attributes, as many or as few annotation attributes can be used. Keep in mind that default values will be used for attributes that are not specified. A list of all available annotation attributes is given at the end of this section on annotations.

For example, to insert 3 annotation features, each belonging to a different anno feature class, for the one 'streets' feature written, a feature like this would be needed:

```
feature type: streets
fme_geometry = fme_line
fme_type = fme_line
geodb_type = geodb_polyline
user_defined_field_1 = value
user_defined_field_2 = value
user_defined_field_3 = value
...
geodb_text{0}.geodb_text_feat_class_name = street_names
geodb_text{0}.geodb_text_x_coord = 7504799.45082186
geodb_text{0}.geodb_text_y_coord = 731099.587626632
geodb_text{0}.geodb_text_angle = 10
geodb_text{0}.geodb_text_line_spacing = 0
geodb_text{0}.geodb_text_ref_scale = 100
geodb_text{0}.geodb_text_scale = true
geodb_text{0}.geodb_text_size = 8.2
geodb_text{0}.geodb_text_string = "displayed street name"
geodb_text{0}.user_field_1_for_street_names = value
geodb_text{0}.user_field_2_for_street_names = value
...
```

```

geodb_text{1}.geodb_font_bold = false
geodb_text{1}.geodb_font_charset = 0
geodb_text{1}.geodb_font_italic = false
geodb_text{1}.geodb_font_name = Arial
geodb_text{1}.geodb_font_size = 20(
geodb_text{1}.geodb_font_strikethrough = false
geodb_text{1}.geodb_font_underline = false
geodb_text{1}.geodb_font_weight = 400
geodb_text{1}.geodb_text_feat_class_name = alternate_street_names
geodb_text{1}.geodb_text_x_coord = 7504783.11300916
geodb_text{1}.geodb_text_y_coord = 731109.158628889
geodb_text{1}.geodb_text_angle = 56.8
geodb_text{1}.geodb_text_size = 4.3
geodb_text{1}.geodb_text_string = "alternate street name"
geodb_text{1}.user_field_1_for_alternate_street_names = value
...
geodb_text{2}.geodb_text_feat_class_name = old_street_names
geodb_text{2}.geodb_text_string = "old street name"
geodb_text{2}.geodb_text_x_coord = 7504788.43294883
geodb_text{2}.geodb_text_y_coord = 731105.044247817
geodb_text{2}.user_field_1_for_old_street_names = value
...
Geometry Type: Line (2)

Number of Coordinates: 4 -- Coordinate Dimension: 2 -- Coordinate System: _FME_0'

(7504779.48166667,731111.522380952)
(7504797.35380952,731098.524285714)
(7504813.13690476,731104.327142857)
(7504805.59357143,731109.201190476)

```

Lastly, the TRANSACTION_TYPE directive must be set to EDIT_SESSION or VERSIONING whenever writing feature-linked annotations.

Annotation Attributes

The following attributes are used to store the annotation information within an FME annotation feature. In ArcGIS 9.1, the schema of annotation feature classes changed and a considerable number of additional fields were added. These new fields contain information about the annotation such as its font, size, angle, offset, leading, etc. When writing, these fields should **never** be set directly; instead the attributes in the table below must be used to control the properties of the annotation. After the translation, the fields will display the desired values because the attributes below correspond to some of the fields.

Attribute Name	Contents
geodb_text_string	The annotation string. It is returned as a UTF-16 encoded string by the reader. The writer converts the value supplied for this attribute into UTF-16.
geodb_text_size	The size of the text in user units. This size gets converted to points, and the text will be displayed at this size when viewed at the reference scale. If this attribute is not supplied, then a default text size of 10 points is used and no conversion is done. Default: 10.0 points

Attribute Name	Contents
geodb_text_feat_class_name	The name of the destination annotation feature class. It is only used when writing feature-linked annotations.
geodb_text_x_coord	The x coordinate of the annotation's location. It is only used when writing feature-linked annotations.
geodb_text_y_coord	The y coordinate of the annotation's location. It is only used when writing feature-linked annotations.
geodb_linked_feature_id	<p>The ID of the feature to which the annotation is linked. Only applicable when writing feature-linked annotation and when the feature being linked to has already been written. In most cases the non-annotation feature will not yet have been written and so the feature ID will not be known. In this case, the Geodatabase writer can provide the correct value for this attribute. See the section Feature-Linked Annotation for more information.</p> <p>Default: -1</p>
geodb_anno_class_id	<p>The ID of the Annotation Class to use when writing the annotation. The ID is an integer: to see what ID an Annotation Class maps to, view the Subtypes tab of the Feature Class Properties dialog in ESRI ArcCatalog® for the annotation feature class. If an invalid ID or -1 was specified then the annotation will be created using an inline text symbol; otherwise the annotation will reference an existing text symbol. Referencing existing text symbols decreases table size and may improve performance.</p> <p>When using an Annotation Class, certain annotation properties can be overridden and others cannot. The following attributes cannot override the properties of the Annotation Class:</p> <ul style="list-style-type: none"> - geodb_font_strikethrough - geodb_font_weight - geodb_font_charset - geodb_text_scale - geodb_text_break_char - geodb_text_clip - geodb_right_to_left <p>The following attributes can override the properties of the Annotation Class:</p> <ul style="list-style-type: none"> - geodb_text_size

Attribute Name	Contents
	<ul style="list-style-type: none"> - geodb_text_angle - geodb_font_name - geodb_font_size - geodb_font_italic - geodb_font_underline - geodb_font_bold - geodb_text_color - geodb_color - geodb_text_x_offset - geodb_text_y_offset - geodb_h_align - geodb_v_align - geodb_text_line_spacing <p>Default: -1</p>
geodb_symbol_id	<p>The ID of Symbol to use when writing the annotation. The ID is an integer: to see what ID a Symbol maps to, view the Annotation or Subtypes tabs of the Feature Class Properties dialog in ESRI ArcCatalog® for the annotation feature class. If an invalid ID or -1 was specified then the annotation will be created using an inline text symbol; otherwise the annotation will reference an existing text symbol.</p> <p>If geodb_anno_class_id is also specified, geodb_symbol_id takes precedence. See geodb_anno_class_id for details about overriding annotation properties.</p> <p>Default: -1</p>
geodb_font_name	<p>The name of the font used to display the text string.</p> <p>Default: Arial</p>
geodb_font_size	<p>The size of the font used to display the text string.</p> <p>Default: 10</p>
geodb_font_italic	<p>Indicates whether the string should be Italicized text. Allowable values are Yes and No.</p> <p>Default: No</p>
geodb_font_underline	<p>Indicates whether the string should be underlined text. Allowable values are Yes and No.</p> <p>Default: No</p>
geodb_font_bold	<p>Indicates whether the string should be boldface text. Allowable values are Yes and No.</p> <p>Default: No</p>

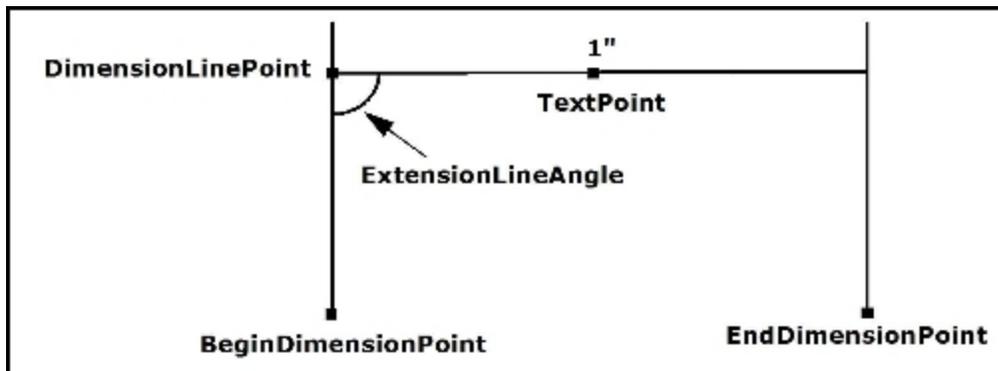
Attribute Name	Contents
geodb_font_strikethrough	<p>Indicates whether the string should be "strike-through" text. Allowable values are Yes and No.</p> <p>Default: No</p>
geodb_font_weight	<p>Indicates the weight of the font being used to display the string. The value must be an integer greater than or equal to zero.</p> <p>Default: 400</p>
geodb_font_charset	<p>Indicates the character set being used to display the string. The value must be the integer value associated with a specific character set. For example, the ANSI character set is given the value 0, the default character set is given the value 1, and the symbol character set is given the value 2. Some additional character sets, and their values, are:</p> <p>BALTIC_CHARSET 186 CHINESEBIG5_CHARSET 136 EASTEUROPE_CHARSET 238 GB2312_CHARSET 134 GREEK_CHARSET 161 HANGUL_CHARSET 129 MAC_CHARSET 77 OEM_CHARSET 255 RUSSIAN_CHARSET 204 SHIFTJIS_CHARSET 128 TURKISH_CHARSET 162</p> <p>Default: 0 (ANSI character set)</p>
geodb_color	<p>The color of the text defined as an RGB string, with each value separated by a comma. Each value must be an integer between 0 and 255 (inclusive). Note that the Geodatabase writer has an internal attribute for color (geodb_text_color) and so the default for this attribute is only used when no value is supplied for the geodb_text_color attribute.</p> <p>Default: 0,0,0 (black)</p>
geodb_text_angle	<p>The rotation of the annotation measured from the horizontal in a counterclockwise direction. It is measured in degrees.</p> <p>Default: 0</p>

Attribute Name	Contents
geodb_text_ref_scale	<p>The reference scale at which the text's size, on the screen, is the size indicated by <code>geodb_text_size/fme_text_size</code>. When the scale is larger in value than the reference scale, the text appears smaller than that indicated by <code>geodb_text_size/fme_text_size</code>, and vice versa.</p> <p>Default: 1</p>
geodb_text_scale	<p>Indicates whether the text scales with the map.</p> <p>Default: TRUE</p>
geodb_text_break_char	<p>The ASCII value of the character that should be interpreted as the line end.</p> <p>Default: 10 (the line feed character)</p>
geodb_text_clip	<p>Indicates whether the text string will be clipped in order to fit into an envelope geometry.</p> <p>Default: No</p>
geodb_text_x_offset	<p>The text offset in the x direction, measured in points.</p> <p>Default: 0</p>
geodb_text_y_offset	<p>The text offset in the y direction, measured in points.</p> <p>Default: 0</p>
geodb_text_leader_line	<p>The geometry of the leader line associated with the annotation, if present. It will be stored in OGC WKT format.</p>
geodb_text_leader_line_anchor_point	<p>The geometry of the leader line anchor point associated with the annotation, if present. It will be stored in OGC WKT format.</p>
geodb_h_align	<p>The alignment of text horizontally if the text spans multiple lines.</p> <p>Options: left, right, center, full</p> <p>Default: left</p>
geodb_v_align	<p>The vertical alignment of text.</p> <p>Options: baseline, bottom, center, top</p> <p>Default: bottom</p>
geodb_right_to_left	<p>If TRUE, then this indicates that the text is written from right to left. If FALSE, then this indicates that the text is written from left to right.</p> <p>Default: FALSE</p>
geodb_text_char_spacing	<p>The amount of character spacing, measured as a percentage of the original character's length. A value of 0</p>

Attribute Name	Contents
	<p>indicates that the standard amount of character spacing, as set by ESRI, will be used. A value greater than 0 increases the amount of character spacing, whereas a value less than 0 decreases the amount of character spacing.</p> <p>Default: 0</p>
geodb_text_character_width	<p>The width added to each character, beyond what is defined by its character box in its font. Character width is a percentage of the original character.</p> <p>Default: 0</p>
geodb_text_line_spacing	<p>The amount of line spacing, measured in font points. The value must be a real number. A value of 0 indicates that the standard amount of line spacing, as set by ESRI, will be used. A value greater than 0 increases the amount of line spacing, whereas a value less than zero decreases the amount of line spacing. If the value is small enough, the order of lines will get reversed (that is, the first line becomes the last line, the second line becomes the second last line, and so on).</p> <p>Default: 0</p>

Dimensions

geodb_type: geodb_dimension



Dimension features are defined by the following attributes:

Attribute Name	Contents
geodb_dim_style_id	<p>The numeric ID describing which style this dimension uses. When writing dimensions this attribute must be supplied on the feature and be assigned an ID for an existing style, otherwise an error will occur.</p>

Attribute Name	Contents
	<p>Note: When translating from one Geodatabase dimension feature class to another, make sure that the destination feature class contains all the dimension styles used by the input dimension feature class. This may mean that the destination dimension feature class has to be created before the translation using ArcGIS.</p>
geodb_dim_length	The length of the dimension. A read-only attribute.
geodb_dim_custom_length	<p>A length specified to be displayed instead of the actual length in geodb_dim_length. This value only gets used if geodb_dim_using_custom_length is set to true.</p> <p>Default: 0</p>
geodb_dim_using_custom_length	<p>Specifies whether or not to use the custom length specified instead of the actual length. Permitted values are true and false.</p> <p>Default: false</p>
geodb_dim_type	<p>Specifies whether the dimension is linear or aligned. Please reference ESRI documentation on dimensions for specific definitions. The numbers specified follow the ESRI enumeration esriDimensionType and the values are:</p> <p>0 = aligned 1 = linear</p> <p>Default: 0</p>
geodb_dim_line_display	<p>Specifies which dimension parts appear on the dimension line (that is, if they point inward or outward, etc.). The integer values for this parameter follow the ESRI enumeration esriDimensionDisplay and the values are:</p> <p>0 = Displays both dimension parts. 1 = Displays the beginning dimension part. 2 = Displays the ending dimension part. 3 = Does not display any dimension part.</p> <p>Setting this attribute overrides the value set by the dimension style.</p>
geodb_dim_extn_line_display	<p>Specifies which dimension parts appear on the extension line. The valid values are the same as for geodb_dim_line_display. Setting this attribute overrides the value set by the dimension</p>

Attribute Name	Contents
	style.
geodb_dim_marker_display	Specifies how arrows are displayed for the dimension. The values are the same as that of geodb_dim_line_display, except that they apply to markers (arrows) instead of dimension parts. Setting this attribute overrides the value set by the dimension style.
geodb_dim_text_angle	The angle of the text displayed, in radians. From ESRI's documentation: "The TextAngle property will only affect the dimension if the dimension's style's text alignment property is True in which case the text is always parallel to the dimension line." Default: 0
geodb_dim_extn_line_angle	The angle (in degrees) between the dimension line and the extension line. Default: 90
geodb_dim_begin_dimension_x	The X value for the Begin Dimension Point.
geodb_dim_begin_dimension_y	The Y value for the Begin Dimension Point.
geodb_dim_begin_dimension_z	The Z value for the Begin Dimension Point.
geodb_dim_end_dimension_x	The X value for the End Dimension Point.
geodb_dim_end_dimension_y	The Y value for the End Dimension Point.
geodb_dim_end_dimension_z	The Z value for the End Dimension Point.
geodb_dim_line_x	The X value for the Dimension Line Point. The Dimension Line Point determines the height of the dimension line above the baseline. To create a two-point dimension, the Dimension Line Point must be the same as the Begin Dimension Point.
geodb_dim_line_y	The Y value for the Dimension Line Point. The Dimension Line Point determines the height of the dimension line above the baseline. To create a two-point dimension, the Dimension Line Point must be the same as the Begin Dimension Point.
geodb_dim_line_z	The Z value for the Dimension Line Point. The Dimension Line Point determines the height of the dimension line above the baseline. To create a two-point dimension, the Dimension Line Point must be the same as the Begin Dimension Point.

Attribute Name	Contents
geodb_dim_text_x	The X value for the Text Point. If the x,y,z values for the text point are all zero then the default text position is used. Default: 0
geodb_dim_text_y	The Y value for the Text Point. If the x,y,z values for the text point are all zero, then the default text position is used. Default: 0
geodb_dim_text_z	The Z value for the Text Point. If the x,y,z values for the text point are all zero, then the default text position is used. Default: 0

Simple Junctions

geodb_type: geodb_simple_junction

This type is supported by both the reader and the writer. However, simple junction feature classes must be created in ArcCatalog prior to running the translation. Simple junction features are defined by the following attributes:

Attribute Name	Contents
geodb_edge_feature_count	The number of edge features associated with the junction. Present only on features being read.
geodb_element_id	The logical network element ID of the junction. Present only on features being read.
geodb_ancillary_role	The network ancillary role of the junction. Possible values are: none, source, and sink.

Simple Edges

geodb_type: geodb_simple_edge

This type is supported by both the reader and the writer. However, simple edge feature classes must be created in ArcCatalog prior to running the translation. Simple edge features are defined by the following attributes:

Attribute Name	Contents
geodb_element_id	The logical network element ID of the junction. Present only on features being read.
geodb_from_junction_element_id	The junction element ID that corresponds to the from endpoint. Present only on features being read.
geodb_to_junction_element_id	The junction element ID that corresponds to the to endpoint. Present only on features being read.

Complex Junctions

geodb_type: geodb_complex_junction

This type is deprecated, and only supported by the Reader. Complex junction features are defined by the following attributes:

Attribute Name	Contents
geodb_junction_element_count	The number of junctions associated with the feature
geodb_edge_feature_count{}	The number of edge features associated with the indexed connection point
geodb_topological_configuration	The configuration of the feature. Possible values are: chain, loop, star, and mesh.
geodb_ancillary_role	The network ancillary role of the junction. Possible values are: none, source, and sink.
geodb_edge_element_count	The number of edge elements associated with the feature.

Complex Edges

geodb_type: geodb_complex_edge

This type is supported by both the reader and the writer. However, complex edge feature classes must be created in ArcCatalog prior to running the translation. The attributes present on an FME feature depend on the value for the reader directive SPLIT_COMPLEX_EDGES. If the value is NO, the following attributes will be present:

Attribute Name	Contents
geodb_edge_element_count	The number of edge elements associated with the feature. Present only on features being read.
geodb_from_junction_element_id	The junction element ID that corresponds to the <i>from endpoint</i> . Present only on features being read..
geodb_junction_feature_count	The number of connected junction features. Present only on features being read.
geodb_to_junction_element_id	The junction element ID that corresponds to the <i>to endpoint</i> . Present only on features being read.

If the value is YES, these attributes will be present:

Attribute Name	Contents
geodb_element_id	The element ID of the logical edge element. Present only on features being read.
geodb_element_index	An attribute created and assigned by FME. It is used to order the edge elements within a complex feature. The index begins at zero, not one. Present only on features

Attribute Name	Contents
	being read.
geodb_from_junction_element_id	The junction element ID that corresponds to the from endpoint. Note: This is the <i>from endpoint</i> of the edge element, not the edge feature. Present only on features being read.
geodb_to_junction_element_id	The junction element ID that corresponds to the to endpoint. Note: This is the <i>to endpoint</i> of the edge element, not the edge feature. Present only on features being read.

Relationships

geodb_type: geodb_relationship and geodb_attributed_relationship

Relationship features contain information about a single relationship between an origin and destination feature. They can be both read and written using FME: attributed relationships can be inserted, updated and deleted, while non-attributed relationships can only be inserted and deleted. Relationships are not rows in a table or feature class like other features, but rather implied through the primary and foreign key values of an origin and destination feature. Attributed relationships have intermediate tables associated with them, which can be updated by providing an RID (relationship id) as a key field, much as an OBJECTID must be provided when updating a table or feature class.

Relationship classes cannot be created through FME, and must be set up through ArcCatalog prior to running the translation.

When reading, the following attributes are stored on the feature, and are required for writing relationships; they must be supplied, if the feature was not read from Geodatabase:

Attribute Name	Contents
geodb_rel_origin_oid	The OBJECTID of the related origin feature.
geodb_rel_destination_oid	The OBJECTID of the related destination feature.
geodb_type	geodb_relationship (for non-attributed relationships) geodb_attributed_relationship (for attributed relationships)

The following attributes are stored on all related origin or destination features, and are required for writing relationships; they must be supplied if a feature was not read from Geodatabase:

Attribute Name	Contents
geodb_feature_has_relationships	Whether the feature participates in a relationship as an origin or destination. 'true', and 't' (all case-insensitive) are accepted for features participating in
geodb_id	The temporary OBJECTID of the feature.

In particular, it's possible to write to origin and destination feature classes, as well as associated relationship classes, all in one pass. This is automatic when reading from one Geodatabase and writing to another, but can be achieved in other cases with additional care. This applies to attributed and non-attributed relationships. It applies to relationships that reference the OBJECTID field in the origin and destination tables, as well as relationships that reference other fields in the origin and destination tables.

When writing origin or destination features that will be referenced by relationship features written during the same pass, they must have:

`geodb_oid = <temporary local object id>`

`geodb_feature_has_relationships = "yes"`

The `<temporary local object id>` does not become the ultimate OBJECTID in ArcGIS. However, it can be used to reference origin and destination features when writing relationship features in a single pass.

When writing relationship features, they must have:

`geodb_rel_origin_oid = <temporary local object id> OR <true object id>`

`geodb_rel_destination_oid = <temporary local object id> OR <true object id>`

`geodb_type = "geodb_relationship"` (for non-attributed relationships), or

`"geodb_attributed_relationship"` (for attributed relationships)

If there is an overlap between true object IDs and temporary local object IDs, FME preferentially assumes the ID is a temporary local one. This allows the user to ignore the true object IDs if desired.

The destination feature type properties for relationship classes should have

`"Allowed Geometries" = "geodb_relationship" or "geodb_attributed_relationship"`

as appropriate. (This explanation is for Workbench; mapping file authors should set `geodb_type` to one of the above on the DEF line. See Geodatabase Table Representation for more details.)

Note that when writing relationship features with FME, you must always provide object IDs, *not* the values of the actual origin and destination key fields if they are different than OBJECTID.

Metadata

`geodb_type: geodb_metadata`

Metadata features contain metadata about the feature type. Metadata can be read and written. The metadata used is of the same form as when using ESRI's ArcCatalog to export table metadata to (plain) XML.

When reading, the following attributes are supplied on the feature, where applicable:

Attribute Name	Contents
<code>fme_contains_spatial_column</code>	yes or no, depending on whether the feature type is a non-spatial table, attributed relationship, or a feature class
<code>fme_dimension</code>	2 or 3, depending on the dimension of the feature class
<code>fme_feature_identifier</code>	The name of the object ID field
<code>fme_geometry{0}</code>	The geometry of the feature class. This will be set to <code>fme_no_geom</code> for non-spatial tables and attributed relationships
<code>fme_num_entries</code> (Personal Geodb only)	The total number of features in the table
<code>geodb_metadata_string</code>	The geodatabase metadata in XML

If the feature type represents a feature class, the geometry of the metadata feature returned is a polygon, representing the extents of the feature class and the coordinate system of the feature class also gets set on the feature.

When writing, the `geodb_type` of the feature must be `geodb_metadata`; however, the `geodb_type` of the destination feature type must not be `geodb_metadata`, but rather the type of the table itself.

The metadata within `geodb_metadata_string` will overwrite any previous metadata that exists in the table. If multiple metadata features are written to a single table, then the last metadata feature will be used. Viewing the metadata

within ArcCatalog after the translation will automatically update certain fields, such as table name & record count, if they were set incorrectly in geodb_metadata_string. However, if you use FME to read back the metadata before viewing the results in ArcCatalog then the incorrect fields will not have been corrected. None of the other attributes supplied on an FME feature when reading metadata will get used when writing metadata.

Writing metadata features does not increase the number of features in a table.

Raster

Features with this value consist of a raster geometry, and are used to represent a two-dimensional grid of values. FME supports the writing and reading of both single- and multi-banded numeric rasters, and three-banded color rasters.

Attribute Name	Contents
geodb_raster_compression_type	The type of compression algorithm used to store data in the raster dataset.
geodb_raster_compression_quality	The compression quality.
geodb_raster_pyramid_resample_type	The resampling method used when building reduced resolution pyramids on the raster dataset.
geodb_raster_pyramid_level	The number of reduced resolution pyramids built.

ESRI Geodatabase (XML) Reader

Format Notes: To use FME's ESRI XML Geodatabase Reader, you must also install ESRI's ArcGIS® 9. It is not available with ArcGIS 8.

Since the XML Geodatabase Reader is based on the same technology as FME's ESRI Geodatabase Reader, the two formats are essentially the same. Please see the Reader section of the *ESRI Geodatabase Reader/Writer* for information on usage. Only the Geodatabase Reader directives listed in that chapter are currently supported.

The XML Geodatabase Reader allows FME to retrieve data from ESRI's XML Workspace Document.

Overview

The XML Geodatabase Reader can read both binary and text XML Workspace Documents.

XML Geodatabase Quick Facts

Format Type Identifier	GEODATABASE_XML
Reader/Writer	Reader
Licensing Level	ESRI Edition
Dependencies	ArcGIS 9
Dataset Type	File
Feature Type	Table/Feature Class name
Typical File Extensions	.xml or .ZIP or .Z
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Always
Schema Required	No
Transaction Support	N/A
Enhanced Geometry	Yes
Geometry Type	geodb_type
Encoding Support	Yes

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	yes	point	yes
circles	yes	polygon	yes
circular arc	yes	raster	yes

Geometry Support				
Geometry	Supported?		Geometry	Supported?
donut polygon	yes		solid	no
elliptical arc	yes		surface	yes
ellipses	yes		text	yes
line	yes		z values	yes
none	yes			

ESRI Shape Reader/Writer

The ESRI® Shape Reader and Writer module allows FME to read and writer ESRI's Shape format. The Shape format is the native format of ESRI's ArcView product and has been made public by ESRI.

Overview

An ESRI shapefile consists of a main file, an index file, and a dBASE table. The main file is a direct access, variable-record-length file in which each record describes a shape with a list of its vertices. In the index file, each record contains the offset of the corresponding main file record from the beginning of the main file. The dBASE table contains feature attributes with one record per feature. The one-to-one relationship between geometry and attributes is based on record number. Attribute records in the dBASE file must be in the same order as records in the main file.

Shapefiles store both geometry and attributes for features. No topological information however is carried in a shapefile. A single logical shapefile consists of three physical files, each with one of the following file name extensions:

File Name Extension	Contents
.shp	Geometric data
.shx	Index to the geometric data
.dbf	Attributes for the geometric data
.sbn and .sbx	Spatial index for the geometric data. These two files will not exist unless you generate them with an ESRI product.

These extensions are added to the base name of the shapefile, creating separate physical files that must all reside in the same directory.

Point, multipoint, polyline, polygon, and multipatch geometric data can be stored in **.shp** files. However, a single **.shp** file can contain only one type of geometry. Each entity in a **.shp** file has a corresponding entry in the **.shx** index file and a corresponding row of attributes in the associated **.dbf** file. The order of the entries in each of these files is synchronized. For example, the third geometric entity in the **.shp** file is pointed to by the third entry in the **.shx** index file and has the attributes held in the third row of the **.dbf** file.

In the case of multipoint data, there is only one **.dbf** row for each set of points held in the file. This is in contrast with a point file where there is one **.dbf** row for each point. Polyline files contain linear features or aggregates of linear features, each having a single attribute entry. Polygon files contain polygons or groups of disjoint or overlapped, in the case of holes, polygons each having a single attribute entity.

Tips:

Aggregate linear features and aggregate polygonal features may be created using the `AggregateFactory`. They may be broken into their component pieces for output to formats that do not support aggregation using the `DeaggregateFactory`.

If a polygon containing holes is written to a Shapefile, any adjacent holes will be merged into a single hole before the polygon is output.

The number and type of attributes associated with each entity is user-definable however, there must be at least one field in the **.dbf** file. As well, all features in the same shapefile will have the same number and type of attributes.

Note: Any single DBF (attribute) file can have a maximum file size of 2 GB, a limit imposed by the dBase III specification. Files larger than 2 GB may be readable, but not officially supported. Files larger than 2 GB are not writable, and will produce an error message.

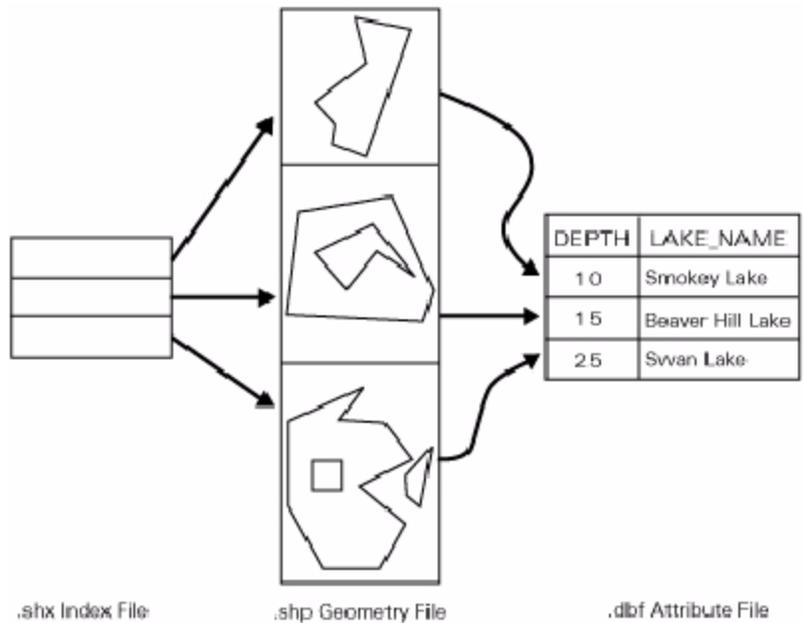
Shapefiles may hold both two- and three-dimensional geometry, as well as an optional measure value on each vertex. However, all features within a single shapefile will have the same dimensionality. Note that while older ESRI products may only support two-dimensional shapefiles, FME can read and write both two- and three-dimensional shapefiles. FME can also handle measure data associated with features.

Note: Measures are currently not supported when reading or the shape_multipatch geometry type.

In previous FME releases, when a feature with measures went through certain transformers, the measures would sometimes become out of sync with the feature that they were attached to. The addition of enhanced geometry support ensures that this will no longer happen, as long as the workspace/mapping file has been set to use enhanced geometry.

FME considers a Shape dataset to be a collection of shapefiles in a single directory. The geometry type and attribute definitions of each shapefile must be defined in the mapping file before being read or written.

The following diagram shows a Shape polygon file with three geometric entities in it. The index file has three entries, each of which refer to the vector data defining each polygon. Notice the second polygon contains a hole and the third polygon is an aggregate of two disjoint polygons, one of which contains a hole. Each geometric entity in turn corresponds with one record in the attribute table.



ESRI Shape Quick Facts

Format Type Identifier	SHAPE
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	Directory or File
Feature Type	File base name
Typical File Extensions	.shp (.shx, .dbf)
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Optional
Generic Color Support	No
Spatial Index	Optional
Schema Required	Yes
Transaction Support	No
Enhanced Geometry	Yes
Encoding Support	Yes
Geometry Type	SHAPE_GEOMETRY

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	yes
ellipses	no		text	no
line	yes		z values	yes
none	yes			

Reader Overview

The Shape reader produces FME features for all feature data held in shapefiles residing in a given directory. The Shape reader first scans the directory given for the shapefiles which have been defined in the mapping file. For each shapefile it finds, it checks to see if that file is requested by looking at the list of IDs specified in the mapping file. If a match is made or no IDs were specified in the mapping file, the shapefile is opened to be read. The Shape reader extracts features one at a time from the file and passes them on to the rest of the FME for further processing. When the file is exhausted, the Shape reader starts on the next file in the directory.

Reader Directives

The suffixes shown are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the Shape reader is `SHAPE`.

DATASET

Required/Optional: *Required*

The value for this directive is the directory containing the shapefiles to be read, or a single shapefile. A typical mapping file fragment specifying an input Shape dataset looks like:

```
SHAPE_DATASET /usr/data/shape/92i080
```

Workbench Parameter: *Source ESRI Shape File(s)*

DEF

Required/Optional: *Optional*

The definition specifies only the base name of the file, the type of geometry it contains, and names and types of all attributes. The syntax of a Shape `DEF` line is:

```
<ReaderKeyword>_DEF <baseName>
SHAPE_GEOMETRY shape_point|
shape_multipoint|
shape_polyline|
shape_polygon|
shape_null|
shape_pointm|
shape_polylinem|
shape_polygonm|
shape_pointz|
shape_polylinez|
shape_polygonz|
shape_multipart|
[<attrName> <attrType>]+
```

Note: In older versions of FME, `shape_polyline` was called `shape_arc`. This has been changed to avoid confusion with mathematical arcs. However, FME still accepts `shape_arc` in place of `shape_polyline` to accommodate backwards compatibility.

Shapefiles with geometry of `shape_point`, `shape_multi_point`, `shape_polyline`, and `shape_polygon` contain two-dimensional features. A geometry of `shape_null` is used for shape files that contain no geometry. If the type of geometry has an `m` at the end, then each two-dimensional coordinate of a feature may optionally have an associated measure value. If the type of geometry has a `z` at the end or is `shape_multipart`, the shapefile contains three-dimensional features, and each coordinate may optionally have an associated measure value.

The file name of each of the physical shapefiles is constructed by adding their extension to the base name. The `SHAPE_GEOMETRY` clause specifies the geometry type for the entire file.

It is also possible to store features having no defined geometry. These features have their `SHAPE_GEOMETRY` attribute set to `shape_null`. These `shape_null` features may be stored or read from any type of shapefile.

Tip: When creating Shapefiles, no attributes need to be specified on the `SHAPE_DEF` line. When no attributes are defined on a Shapefile being written, FME automatically generates an `_ID` attribute for the Shapefile. This is useful if the Shapefile is to be imported into ArcInfo. If the Shapefile contained polygons, an `AREA` attribute is also generated. In both cases, the values of these attributes will be `NULL` for all features.

Shapefiles require at least one attribute be defined. The attribute definition given must match the definition of the file being read. If it does not, translation is halted and the true definition of the shapefile's attributes is logged to the log

file. All shapefile attribute names must be uppercase and must not exceed 10 characters in length. The following table shows the attribute types that are supported.

Field Type	Description
char(<width>)	Character fields store fixed-length strings. The width parameter controls the maximum characters that can be stored by the field. When a character field is written, it is right-padded with blanks, or truncated, to fit the width. When a character field is retrieved, any padding blank characters are stripped away.
date	Date fields store dates as character strings with the format YYYYMMDD.
logical	Logical fields store TRUE/FALSE data. Data read to or written from such fields must always have a value of either true or false.
number(<width>,<decimals>)	Number fields store single and double precision floating point values. The width parameter is the total number of characters allocated to the field, including the decimal point. The decimals parameter controls the precision of the data and is the number of digits to the right of the decimal.

The following mapping file fragment defines two shapefiles: one containing polygonal features possibly disjoint and with holes, and the other containing linear features:

```
SHAPE_DEF landcover SHAPE_GEOMETRY shape_polygon \
    AREA    number(12,3) \
    TYPE    char(11)\
    PERIMETER    number(12,3)
SHAPE_DEF roads SHAPE_GEOMETRY shape_arc \
    NUMOFLANES number(2,0) \
    TYPE    char(5) \
    UNDERCNST logical \
    DIVIDED logical \
    TRVLDIR char(6)
```

Workbench Parameter: <WorkbenchParameter>

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined shapefiles read. If no **IDs** are specified, then all defined and available shapefiles are read. If more shapefiles were in the directory, they are ignored. The syntax of the **IDs** directive is:

```
<ReaderKeyword>_IDS <baseName1>\
<baseName2> ...\
<baseNameN>
```

The base names must match those used in **DEF** lines.

The example below selects only the **roads** shapefile for input during a translation:

SHAPE_IDS roads

Workbench Parameter: *Feature Types to Read*

MEASURES_AS_Z

Required/Optional: *Optional*

This optional specification controls how measures data associated with geometric data is treated. If the value is **yes**, measures data is treated as elevations.

Workbench Parameter: *Treat Measures as Elevation*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

✳ Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

This specification will only be used by the reader for datasets that have an associated spatial index (.sbn and .sbx file).

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>

✳ Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

DISSOLVE HOLES

Required/Optional: *Optional*

This optional specification controls whether the SHAPE reader dissolves adjacent holes in polygons read from shapefiles. If the value is set to **yes** or is not set, then the SHAPE reader will dissolve adjacent holes.

Workbench Parameter: *Dissolve Adjacent Holes*

REPORT BAD GEOMETRY

Required/Optional: *Optional*

This optional specification controls whether the SHAPE reader reports geometric anomalies in input shapefiles.

By default, the SHAPE reader will perform the following operations to ensure the validity of input features: close unclosed polygons, remove duplicate points, remove empty elements, dissolve holes (if **DISSOLVE_HOLES** is set to **YES** or is not set).

If **REPORT_BAD_GEOMETRY** is set to **YES**, then the `shape_geometry_error{}` list attribute will be set on input features, and will contain error messages as geometric anomalies are detected and/or fixed. The error messages are of the following format:

- Closed Polygon at (x,y)
- Duplicated Point at (x,y)
- Removed Empty Element #n near (x,y)
- Removed Duplicate Point at (x,y)
- Invalid Polygon/Donut Orientation near (x,y)
- Dissolved Holes

Workbench Parameter: *Report Geometry Anomalies*

ENCODING

Required/Optional: *Optional*

This optional specification controls which character encoding is used to interpret text attributes from the shapefile. If the value is not set, then the character encoding will be automatically detected from the source shapefile. If the value is set, it will take precedence over the automatically detected character encoding.

This directive is useful when the character encoding information stored in the shapefile is missing or incorrect.

Workbench Parameter: *Character Encoding*

Example:

<ReaderKeyword>_ENCODING <character encoding>

Parameter	Description
<character encoding>	<p>The character encoding to use when interpreting text attributes. Must be set to any of the following values:</p> <p>ANSI - this means use the "current OS language"</p> <p>BIG5</p> <p>EUC</p> <p>HKBIG5</p> <p>ISO</p> <p>OEM</p> <p>SJIS</p> <p>UTF-8</p> <p>CP437</p> <p>CP708</p> <p>CP720</p> <p>CP737</p> <p>CP775</p> <p>CP850</p> <p>CP852</p> <p>CP855</p> <p>CP857</p> <p>CP860</p> <p>CP861</p> <p>CP862</p> <p>CP863</p> <p>CP864</p> <p>CP865</p> <p>CP866</p> <p>CP869</p> <p>CP874</p> <p>CP932</p> <p>CP936</p> <p>CP950</p> <p>CP1250</p> <p>CP1251</p> <p>CP1252</p> <p>CP1253</p> <p>CP1254</p> <p>CP1255</p> <p>CP1256</p> <p>CP1257</p> <p>CP1258</p> <p>ISO8859-1</p>

Parameter	Description
	ISO8859-2
	ISO8859-3
	ISO8859-4
	ISO8859-5
	ISO8859-6
	ISO8859-7
	ISO8859-8
	ISO8859-9
	ISO8859-11
	ISO8859-13
	ISO8859-15
	WINDOWS-874

UPPER_CASE_ATTR_NAMES

Required/Optional: *Optional*

This option specifies whether the reader should upper case attribute names. If no, it will allow mixed case, otherwise attribute names will be uppercased. The default value is no; however, for backwards compatibility, when this keyword is not present, a value of yes will be used.

This keyword is used when generating workspaces & mapping files. As a result, it is not editable within workbench after the workspace has been generated.

Workbench Parameter: *NOT APPLICABLE*

TRIM_PRECEDING_SPACES

Required/Optional: *Optional*

This option specifies whether the reader should trim preceding spaces of attribute values. If the option is set to YES, then preceding spaces in attribute values will be discarded. If the option is set to NO, then preceding spaces will be left intact. The default value is YES.

Workbench Parameter: *Trim Preceding Spaces*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

*** Workbench Parameter**

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

*** Workbench Parameter**

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

*** Workbench Parameter**

Additional Attributes to Expose

Writer Overview

The Shape writer creates and writes feature data to shapefiles in the directory specified by the **DATASET** directive. As with the reader, the directory must exist before the translation occurs. Any old shapefiles in the directory are overwritten with the new feature data. As features are routed to the Shape writer by the FME, it determines the file they are to be written to and outputs them according to the type of the file. Many shapefiles can be written during a single FME session.

Writing to shapefiles of type polygonz or polygonm

Writing to shapefiles of type **polygonz** or **polygonm** will result in a measures column, whether measures exist or not. If they don't, and the Universal Viewer displays in Classic mode, then geometry shows <1.#QNAN> for the M dimension.

Classic Geometry Handling

shape_measures NaN,NaN,NaN,NaN

1: (4526424.7720003976, 5690159.0588858956, 110.87090000000001)

Enhanced Geometry Handling

0: (4526424.7720003976,5690159.0588858956,110.87090000000001)<1.#QNAN>

Writer Directives

The Shape writer processes the **DATASET**, **DEF**, and **MEASURES_AS_Z** directives as described in the *Reader Directives* subsection. It does not make use of the **IDs** or **SEARCH_ENVELOPE** directives.

The **ENCODING** directive is used to specify which character encoding should be used when writing text attributes into shapefiles. If the value of this directive is not set, the current OS language is used. The syntax of the **ENCODING** writer directive is the same as the **ENCODING** reader directive, as described in the *Reader Directives* section.

UPPER_CASE_ATTR_NAMES

Required/Optional: *Optional*

This option specifies whether the writer should change attribute names to uppercase text. If set to NO, mixed case attribute names will be allowed. The default value is YES.

This directive is used when generating workspaces and mapping files. As a result, it is not editable within Workbench after the workspace has been generated.

Workbench Parameter: *NOT APPLICABLE*

SURFACE_AND_SOLID_STORAGE

Required/Optional: *Optional*

This option specifies whether the writer should write input 3D surfaces and solids as shape_multipatch to preserve their original structure or as shape_polygonz to break them down into polygon components. The default value is multipatch.

This directive is used when generating workspaces and mapping files. As a result, it is not editable within Workbench after the workspace has been generated.

Workbench Parameter: *NOT APPLICABLE*

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see [About Feature Attributes](#)), this format adds the format-specific attributes described in this section.

Shape features consist of geometry, a special predefined attribute, and a set of user-defined attributes. All shape features have one predefined attribute, **SHAPE_GEOMETRY**, which identifies the type of the features geometry. Geometry types can be two-dimensional (2D), 2D plus elevations, 2D plus measures, 2D plus elevations and measures or 3D:

Attribute Name	Contents
SHAPE_GEOMETRY	<p>The type of the geometry read from the shapefile. This attribute will contain one of:</p> <ul style="list-style-type: none"> shape_point shape_multipoint shape_polyline shape_polygon shape_null shape_pointm shape_multipointm shape_polylinem shape_polygonm shape_pointz shape_multipointz shape_polylinez shape_polygonz shape_multipart <p>Default: No default</p>
shape_measures	<p>This is present for features that have measures when reading only if the reader directive MEASURES_AS_Z is not specified or is set to no. To write measures using this attribute, make sure the writer directive MEASURES_AS_Z is not specified or is set to no, and simply build this list with one value for each vertex in the feature being written. This is a comma-separated list of floating values which correspond to the vertex measures. The first value is for the first vertex, second for the second, and so on. Not-a-Number values are represented by the string "NaN".</p> <p>However, this will not be present on features if the FME_GEOMETRY_HANDLING directive is set to YES.</p>

A Shapefile defines a set of features that share the same geometry type and the same list of user-defined attributes. In other words, all features belonging to the same shapefile have the same value for the **SHAPE_GEOMETRY** attribute and the same list of user-defined attributes. The values of the user-defined attributes can vary from feature to feature within the same Shapefile. The geometry type and the names of the user-defined attributes for an individual

shapefile are specified in the DEF line for that Shapefile. The feature type of a Shape feature is the same as the base-Name specified in the DEF line.

When reading Shape features, the **SHAPE_GEOMETRY** attribute will correspond to the geometry type specified in the **DEF** line for that for the shapefile. When writing Shape features, the **SHAPE_GEOMETRY** attribute is not required and will be ignored if it is present because the geometry type is taken from the **DEF** line for the shapefile. If the feature being written out cannot be converted into the geometry type specified on the **DEF** line, this feature will not be written out and a warning message will be printed in the logfile. (An example of this would be trying to write an area feature into a point geometry file.)

There is one exception where the geometry type indicated on the **DEF** line may not be the type of file that is actually created. If the **DEF** line indicates that a point file is to be created, but the first actual feature written to that file is instead a multipoint, a multipoint file will be created instead. (The same will be true for **pointz/multipointz** as well as **pointm/multipointm** files.)

As of ESRI ArcGIS Desktop 9.3 Shape files of type **shape_null** are no longer valid. Any **DEF** lines with **SHAPE_GEOMETRY** set to **shape_null** will be output instead as **shape_point**.

When reading a polyline feature with multiple parts, the FME representation consists of an aggregate of lines. Similarly, when reading a polygon feature with multiple parts, the FME representation consists of an aggregate of polygons. Conversely, when writing aggregates of lines or polygons, the FME will output multi-part polyline and polygon Shape features.

When providing 3D data to the Shape writer the polygonz or multipatch geometry types can either split or preserve the 3D geometries. By default 3D surfaces and solids are mapped to multipatches to preserve their representation as single objects. If 3D geometries are instead provided to polygonz destination feature types the surfaces and solids will be converted to individual component polygons.

Facet XDR Reader/Writer

The Facet XDR Reader/Writer allows FME to read and write Facet XDR files. The Facet XDR Format (Facet) is a binary format used by tools produced by Facet Decision Systems, Inc. Facet datasets may be in either ASCII or binary (XDR) format. Currently, FME supports only the binary version.

Overview

Facet data sets store objects that are very flexible in nature. Facet XDR files are self-describing and contain two parts: a signature defining the interpretation of the file's structure and a second part containing either coordinates, attribute data, or methods depending on the signature. Facet structures can mix simple data, other structures, and methods nested to any depth. Therefore, Facet may hold either two-dimensional (2D) or three-dimensional (3D) geometric data.

Facet files store both feature geometry and attributions. A logical Facet file consists of one physical file, with the .xdr file name extension.

The extension **.xdr** is added to the basename of the Facet file when written.

Facet XDR Quick Facts

Format Type Identifier	FACET
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	<ul style="list-style-type: none">• File for Reader• Directory for Writer
Feature Type	File base name
Typical File Extensions	.xdr
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	facet_fme_type
Encoding Support	No

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	point	yes

Geometry Support				
Geometry	Supported?		Geometry	Supported?
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	yes
line	yes		z values	yes
none	yes			

Reader Overview

The Facet reader opens the input file and immediately starts reading features, returning them to the rest of FME for processing. The reader doesn't have any requirement for explicit instruction on how to decode Facet files, as it automatically identifies the type of file from the signature it contains.

The feature returned by the Facet reader has its feature type set to its file basename.

FME automatically recognizes nine different Facet file structures and imports the coordinates and attributes in a special way. If the Facet file does not conform to any of the following nine known formats, FME will interpret the file as a "database" type, which retains all information it contains but does not perform any special conversion on any fields. The table below lists the nine special Facet file formats.

File Format	Contents and Interpretation
text	Contains text features without attributes.
text and attributes	Contains text features with custom attributes.
geometry	Contains single precision geometric information; for example, line, multipoint, polygon.
double geometry	Contains double precision geometric information; for example, line, multipoint, polygon.
geometry and attributes	Contains single precision geometric information; for example, line, multipoint, polygon, followed by custom attributes for each feature.
double geometry and attributes	Contains double precision geometric information; for example, line, multipoint, polygon, followed by custom attributes for each feature.
interleaved geometry and attributes	Contains single precision geometric information; for example, line, multipoint, polygon, interleaved with custom attributes for each feature.
interleaved double geometry and attributes	Contains double precision geometric information; for example, line, multipoint, polygon interleaved with custom attributes for each feature.
database	Contains attribute information only with no coordinates.

Reader Directives

The suffixes listed are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the Facet reader is `FACET`.

DATASET

Required/Optional: *Required*

The value for this keyword is the file name of the Facet XDR file to be read.

Example:

```
FACET_DATASET /usr/data/Canada/roads.xdr
```

Workbench Parameter: *Source Facet XDR File(s)*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The `COORDINATE_SYSTEM` directive, which specifies the coordinate system associated with the data to be read, must always be set if the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` to the reader `COORDINATE_SYSTEM` prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The Facet writer creates and writes feature data to a directory specified by the DATASET keyword. Unlike the reader, this keyword refers to a directory, not a file name. This directory will be created if it does not exist before the translation occurs. Many Facet files may be written to in a single FME session (one translation).

The feature type on each Facet **DEF** line specifies the basename of the output Facet XDR file.

Each Facet **DEF** line specifies a single Facet output file. The attributes listed on the **DEF** line appear within the output Facet file. The special **DEF** line keyword FACET_GEOMETRY assists the FME in determining the format of the output Facet file. The table below outlines how the FME decides what special format the output Facet file will take.

Output File Format	Conditions
text	The FACET_GEOMETRY on the DEF line is assigned the value facet_text . The DEF line specifies no attributes.
text and attributes	The FACET_GEOMETRY on the DEF line is assigned the value facet_text .

Output File Format	Conditions
	The DEF line specifies at least one attribute.
geometry	The COORD_PRECISION keyword is set to Single. The FACET_GEOMETRY on the DEF line is assigned the value facet_line, facet_polygon, or facet_multipoint. The DEF line specifies no attributes.
double geometry	The COORD_PRECISION keyword is set to Double. The FACET_GEOMETRY on the DEF line is assigned the value facet_line, facet_polygon, or facet_multipoint. The DEF line specifies no attributes.
interleaved geometry and attributes	The COORD_PRECISION keyword is set to Single. The FACET_GEOMETRY on the DEF line is assigned the value facet_line, facet_polygon, or facet_multipoint. The DEF line specifies at least one attribute.
interleaved double geometry and attributes	The COORD_PRECISION keyword is set to Double. The FACET_GEOMETRY on the DEF line is assigned the value facet_line, facet_polygon, or facet_multipoint. The DEF line specifies at least one attribute.
database	The FACET_GEOMETRY on the DEF line is not assigned any value, or is assigned a value other than facet_text, facet_line, facet_polygon, or facet_multipoint. The DEF line specifies at least one attribute.

Writer Directives

The following table lists the keywords processed by the Facet Writer. The suffixes shown are prefixed by the current <WriterKeyword> in a mapping file. By default, the <WriterKeyword> for the Facet writer is FACET.

DATASET

Required/Optional: *Required*

The value for this directive is the file name of the Facet XDR file to be written.

Example:

```
FACET_DATASET /usr/data/Canada/roads_output.xdr
```

Workbench Parameter: *Destination Facet XDR Directory*

COORD_PRECISION

Required/Optional: *Optional*

Specifies the precision by which the coordinates will be stored.

Values: *Single* | *Double*

Default Value: *Single*

Workbench Parameter: *Coordinate precision*

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Facet features may consist of geometry and attributes. When reading Facet files, several special attributes hold the data from the file. When writing Facet files, the values in these attributes are written out to the file. If the feature does not have these special attributes, appropriate default values will be used.

Special supported Facet format types include: text, line, polygon, multipoint, point, or database. Note that multipoint data may contain a single point. Also note that database features contain attributes but no geometry.

All Facet features produced by the FME reader contain the `facet_fme_type` attribute, which identifies the geometric type. Depending on the geometric type, the feature contains additional attributes specific to the geometric type. These are described in subsequent sections.

Attribute Name	Contents
<code>facet_fme_type</code>	The Facet geometric type of this entity. Values: <code>facet_multipoint</code> <code>facet_line</code> <code>facet_polyogn</code> <code>facet_text</code> <code>facet_database</code>

Points

facet_fme_type: `facet_multipoint`

A Facet multipoint feature specifies single or multiple 2D or 3D coordinates. There are no special attributes with this type of feature.

Lines

facet_fme_type: `facet_line`

Facet line features contains 2D or 3D linear geometry. There are no special attributes with this type of feature.

Polygons

facet_fme_type: `facet_polygon`

Facet polygon features contains 2D or 3D geometry. Polygons may be either simple or may contain holes, thereby being donuts. There are no special attributes with this type of feature.

Text

facet_fme_type: `facet_text`

Facet text features contain 2D or 3D coordinates and a text string, along with the text alignment, rotation, and size.

Attribute Name	Contents
facet_text_string	<p>The text string.</p> <p>Range: any length character string</p> <p>Default: NULL string</p>
facet_text_alignment	<p>A numeric code indicating the feature's alignment. Several settings are listed below. These settings may be ANDED together to form a single numeric code:</p> <p>TextAlignLeft 1 TextAlignRight 2 TextAlignCenter 4 TextAlignBase 16 TextAlignHalf 32 TextAlignCap 64 TextAlignTop 128 TextAlignBottom 256</p> <p>Range: 0 - 511</p> <p>Default: 0</p>
facet_rotation	<p>The rotation of the text, measured in degrees counter-clockwise from the horizon.</p> <p>Range: any real number</p> <p>Default: 0.0</p>
facet_text_size	<p>The size of the text.</p> <p>Range: any real number</p> <p>Default: 1.0</p>

Database

facet_fme_type: facet_database

Facet database features contain no coordinates. These features may contain any number of custom attributes. There are no special attributes with this type of feature.

FME Feature Store Reader/Writer

This section describes how FME reads and writes FME Feature Store (FFS) files.

Overview

The FFS Reader and Writer modules allow FME to read and write FFS files. This format is a memory dump of FME features, and is the same as the format used by the RecorderFactory. See the RecorderFactory in the *FME Functions and Factories* manual for more details on this format.

Note: If you do not have a current version of FME and you are using the FFS format for data exchange or storage, you may receive this error message:

No geometry mapping entry found for 'fme_raster' in metafile 'C:\Program Files\FME_1378\metafile\FFS.fmf'. Program Terminating.

FFS files that are created with recent FME builds cannot be read by some very early versions of Workbench (build 1378 and earlier).

A spatial index may also be created and saved with the feature store which the FFS reader uses to quickly extract only those features within a specified area.

Because the format is a memory dump of FME features, it can hold anything that FME features carry. This makes the format attractive as a holding spot for data that should persist between FME runs.

A logical FFS dataset consists of one or more files in the same directory with the extension `.ffs`. This extension is added to the base name of the FFS files. It is also possible to use the FFS reader and writer to read and write only a single file.

When writing a large amount of data to a single FFS file, file size limits may be encountered. If this occurs, the data is automatically split into multiple files of acceptable sizes. Reading in the first FFS file will automatically read in all files that were produced when the file was split.

FFS files contain all custom coordinate system definitions used on the features it contains, if any.

Rasters stored to an FFS file will have their data written to a corresponding `.frs` (FME Raster Store) file. One FRS file may hold the data for multiple raster features. FRS files hold up to 2GB of raster data; if the file surpasses this size, the data is automatically split across multiple files. If an FFS file containing raster features is opened and the corresponding FRS file cannot be opened, then the raster features will be restored as polygons with attributes indicating the properties of the raster (e.g., number of rows/columns, spacing, etc.).

FME Feature Store Quick Facts

Format Type Identifier	FFS
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	File
Feature Type	FME Feature Type names
Typical File Extensions	.ffs
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Optional
Schema Required	No
Transaction Support	No
Enhanced Geometry	Yes
Encoding Support	Yes
Geometry Type	fme_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	yes		polygon	yes
circular arc	yes		raster	yes
donut polygon	yes		solid	yes
elliptical arc	yes		surface	yes
ellipses	yes		text	yes
line	yes		z values	yes
none	yes			

Reader Overview

The FFS reader first reads the input file and passes the resultant features on to the rest of FME for further processing. If the FFS files have associated spatial indexes, then a spatial query can be used to limit the features returned.

Note that compressed FFS files may take time to generate, since the whole file is scanned for schema features.

Reader Directives

The suffixes listed are prefixed by the current <ReaderKeyword> in a mapping file.

By default, the <ReaderKeyword> for the FFS reader is *FFS*.

DATASET

The value for this directive is the directory containing the FFS files to be read or a single FFS file.

Required/Optional

Required

Mapping File Syntax

A typical mapping file fragment specifying an input FFS dataset looks like:

```
FFS_DATASET /usr/data/ffs/92i080
```

The dataset may also be an actual FFS file. In such a case, that file is read, and the IDs and DEF lines must not be present.

```
FFS_DATASET /usr/data/data/92i080.ffs
```

* Workbench Parameter

Source FME Feature Store (FFS) File(s)

DEF

This specification is used to define FFS files read. The syntax of the DEF directive is:

```
<ReaderKeyword>_DEF <baseName>  
[<attrName> <attrType>]+
```

Note that this directive is not used when the dataset is a file.

Required/Optional

Optional

Mapping File Syntax

The example below defines a roads FFS file for input during a translation:

```
FFS_IDS roads
```

The following table shows the attribute types supported.

Field Type	Description
char(<width>)	Character fields store fixed length strings. The width parameter controls the maximum number of characters that can be stored by the field. No padding is required for strings shorter than this width.
varchar(<width>)	Variable character fields store variable length strings. The width parameter controls the maximum number of characters that can be stored

Field Type	Description
	by the field. These are often used to optimize storage.
buffer	Buffers store unbounded length character or byte strings.
date	Date fields store dates as character strings with the format YYYYMMDD.
datetime	Datetime fields store dates as character strings with the format YYYYMMDDHHMMSS.FFF
time	Time fields store times as character strings with the format HHMMSS.FFF
number(<width>, <decimals>)	Number fields store single and double precision floating point values. The width parameter is the total number of characters allocated to the field, including the decimal point. The decimals parameter controls the precision of the data and is the number of digits to the right of the decimal.
real64	Float fields store 64 bit floating point values. There is no ability to specify the precision and width of the field.
real32	Float fields store 32 bit floating point values. There is no ability to specify the precision and width of the field.
int16	Int16 fields store 16 bit signed integers and therefore have a range of -32767 to 32767.
int32	Int32 fields store 32 bit signed integers and therefore have a range of -2147483648 to 2147483647.
logical	Logical fields store TRUE/FALSE data. Data read or written from and to such fields must always have a value of either true or false. Another name for this field type is boolean.

IDs

This specification is used to limit the available and defined FFS files read. The syntax of the IDs directive is:

```
<ReaderKeyword>_IDs <baseName1> \
                    <baseName2> ... \
                    <baseNameN>
```

The base names must match those used in DEF lines.

Required/Optional

Optional

Mapping File Syntax

The example below selects only the *roads* FFS file for input during a translation:

```
FFS_IDS roads
```

Note that this directive is not used when the dataset was a file. Also note that if IDs are specified, only those files whose IDs were listed will be read. If no IDs and no DEF lines were present, then all the files in the directory will be read.

* Workbench Parameter

Feature Types to Read

PASSPHRASE

This specification is used to decrypt the source dataset.

This passphrase must exactly match the passphrase that was used to encrypt the dataset when it was created.

Mapping File Syntax

```
<ReaderKeyword>_PASSPHRASE <character string>
```

This must only be used if the FFS files being read were encrypted when they were created.

Required/Optional

Required only when input is encrypted

* Workbench Parameter

Password

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

Note: This directive can only be used if the FFS files being read were created with spatial indexes.

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

SEARCH_CLOSEST_POINT

This specification is used to restrict the returned features to the closest feature in each FFS file to some search point. This directive can only be used if the FFS files being read were created with spatial indexes.

The *mm* parameter indicates the maximum distance a feature can be away from a point before it will be returned. Distances are calculated to the boundaries of area-based features.

Required/Optional

Optional

Mapping File Syntax

The syntax of this directive is:

```
<ReaderKeyword>_SEARCH_CLOSEST_POINT <x> <y> <maxdist>
```

* Workbench Parameter

Closest Search Point

ENFORCE_SECONDARY_FILE_NAMES

The FFS writer has the capability to split one output file into multiple segments and store each segment in a "spillover file" (<filename>_1.ffs, <filename>_2.ffs, etc...). See **MAX_FILE_SIZE**.

However, someone could also have three separate FFS files with the same names as the spillover files. This directive is used to differentiate between the two situations.

Required/Optional

Optional

Values

YES (default) | NO

If the files are spillover files, then this directive should be set to YES. If the files are individual files, then this directive should be set to NO. If the files are individual files, but the directive is set to YES, the features in the spillover files may be read duplicate times.

Mapping File Syntax

```
<ReaderKeyword>_ENFORCE_SECONDARY_FILE_NAMES YES
```

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The FFS writer dumps the memory representation of each FME feature to a disk file, optionally creating a spatial index for the features.

While early versions of the writer wrote to a directory, the current version of the writer will merge all features into a single FFS file and preserve the feature type internally.

The former occurs when the dataset is a directory and `_DEF` lines are present, whereas the latter occurs when no `DEF` lines are present in the mapping file.

Writer Directives

The suffixes shown are prefixed by the current <WriterKeyword> in a mapping file. By default, the <WriterKeyword> for the FFS writer is FFS.

DATASET

The value for this directive is the destination file name for the FFS files.

Required/Optional

Required

Mapping File Syntax

A typical mapping file fragment specifying an output FFS dataset:

```
<writerKeyword>_DATASET /usr/data/ffs/new.ffc
```

* Workbench Parameter

Destination FME Feature Store (FFS) File

DEF

The definition lists only the feature type of those features to be written to the file.

Required/Optional

Optional

Mapping File Syntax

The syntax of the DEF directive is:

```
<writerKeyword>_DEF <baseName>  
[<attrName> <attrType>]+
```

This fragment defines a feature type in the FFS file for all features whose feature type was roads:

```
<writerKeyword>_DEF roads
```

INDEXED

This flag indicates whether or not a spatial index should be created and saved along with the output FFS file.

The spatial index has the same base name as the FFS file, but it will have an .fsi extension. Spatial indexes are needed if the FFS file is later used as the source for spatial queries by the FFS reader.

Required/Optional

Optional

Mapping File Syntax

This fragment specifies that a spatial index should be created:

```
<writerKeyword>_INDEXED yes
```

* Workbench Parameter

Create Spatial Index

STRICT_SCHEMA

This flag indicates whether or not features should have all user attributes not listed on the DEF line removed before they are saved.

Strictly adhering to a schema can, in some cases, greatly reduce file size by removing unnecessary attributes.

Values

yes | no (default)

If yes, the unlisted attributes are stripped, forcing the features to strictly conform to the schema specified on the DEF line.

Required/Optional

Optional

Mapping File Syntax

This fragment specifies that all features strictly adhere to the defined schema:

```
<writerKeyword>_STRICT_SCHEMA yes
```

*** Workbench Parameter**

Enforce Strict Schema

MAX_FILE_SIZE

This directive (which is not generally used) limits the size of each FFS file.

If a file exceeds the specified number of bytes, it will be closed and a new file with a numeric suffix starting at 1 will be created. A single spatial index is created for the group of files.

Required/Optional

Optional

Mapping File Syntax

```
<writerKeyword>_MAX_FILE_SIZE 1000000
```

*** Workbench Parameter**

Maximum FFS File Size, in Bytes

PASSPHRASE

This specification is used to encrypt the output dataset for additional security. This exact passphrase must be used to decrypt this dataset when it is read in again.

If this directive is not used when writing the output dataset, it is not necessary to specify it when reading it in again.

Required/Optional

Optional

Mapping File Syntax

```
<writerKeyword>_PASSPHRASE <character string>
```

* Workbench Parameter

Password

COMPRESSION

A lower COMPRESSION_LEVEL value will result in faster operation for both reading and writing, while a higher compression level will result in smaller file sizes.

Required/Optional

Optional

Values

0 to 9

Default Value: 0

Mapping File Syntax

```
<writerKeyword>_COMPRESSION_LEVEL 0
```

* Workbench Parameter

Compression Level

BYTE_ORDER

The directive BYTE_ORDER indicates whether the resulting file should be optimized for either LITTLE_ENDIAN or BIG_ENDIAN machines.

For example, the architecture of machines running Microsoft Windows is little endian, while the Solaris architecture is big endian.

Required/Optional

Optional

Values

Default Value: NATIVE

The BYTE_ORDER of NATIVE means the file should be optimized for the type of machine on which it is currently running.

Note that all files created can be read back on machines of either byte order; the only issue is that reading back files optimized for the opposite byte order will take slightly longer.

Mapping File Syntax

```
<writerKeyword>_BYTE_ORDER LITTLE_ENDIAN
```

* Workbench Parameter

Byte Order

FILE_DEST_DATASET

This writer directive allows the FFS writer to write to single files instead of directories (early versions of the FFS writer wrote only to directories).

This directive is supplied by default in all newly generated workspaces.

Note: If you are using this writer through the FMEObjects SDK, the old behavior will still apply and you will need to use this directive to enable the new behavior. If a directory output is still required, you might consider using *Fan-out Dataset* in Workbench.

Required/Optional

Optional

Values

Default Value: *YES*

Mapping File Syntax

```
<writerKeyword>_FILE_DEST_DATASET YES
```

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

The FFS reader and writer just make memory dumps of FME features, therefore no special attributes apply. All attributes on the feature are read and written.

Genasys GenaMap Reader

Format Notes:

This format is not supported by FME Base Edition.

The GenaMap Reader module enables FME to read GenaMap Type 4, Type 5, and Type 10 maps. Type 4 maps are two-dimensional (2D) Vector maps containing point, line, and area features, Type 5 maps are the Type 4 (3D) equivalent, and Type 10 maps are Text maps containing graphical annotation information. This section assumes familiarity with these formats.

Overview

GenaMap is a Geographic Information System (GIS) with comprehensive functionality for entering, editing, displaying, analyzing and reporting map data.

Information stored by the GenaMap system is organized on the basis of individual maps. Each map has a type number associated with it to indicate its map type. The following tables indicate the map types that the FME GenaMap reader module recognizes.

Type 4(2D), Type 5(3D) – Point, Line, and Area Maps

A logical Type 4 or Type 5 map consists of several physical files having the following file name extensions:

File Name Extension	Contents
.FH	The Map Header contains the metadata of the vector map.
.FF	The Feature File contains information on how to form tagged point, tagged line, and area features.
.FL	The Chain File contains information on which edges are chained to form the features.
.FE	The Edge File contains references to all edges of the map.
.FN	The Node File contains all nodes of the map.
.FC	The Coordinate File contains the coordinate records that make up all edges of the map.
.FT	The Tag File contains the tag values for tagged vector features.
.FQ	The Tag/Queue File is built from destroyed features.
.FR	The Edge Minimum Bounding Rectangle's (MBR's) File contains the minimum bounding rectangle for each available edge in the map.

Type 3 – Attribute Table

Note: Type 3 maps are accessed when the input Type 4 or Type 5 maps have their attribute hookup status set.

A logical Type 3 map consists of several physical files with the following file name extensions:

File Name Extension	Contents
.FA	The Attribute Header file contains the available attribute types and descriptions for the attribute table.
.FD	The Attribute Data file contains the actual values for attribute table.

Type 10 – Text Maps

A logical Type 10 map consists of several physical files with these file name extensions:

File Name Extension	Contents
.FH	The Map Header contains the metadata for the text map.
.FF	The Feature File contains the text feature records. The text feature records contain indexes to the .FT , .TG , and .TS files.
.FT	The Tag File contains the tag values for tagged text features.
.TG	The Text Graphics file contains information—such as text rotation, justification, and so on—that describes the text string.
.TS	The Text String file contains the actual text string of the text.

GenaMap Quick Facts

Format Type Identifier	GENAMAP
Reader/Writer	Reader
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	Map name
Typical File Extensions	.fh (. fa, .fc, .fd, .fe, .ff, .fl, .fq, .fr, .ft, .tg, .ts)
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	No
Transaction Support	No
Geometry Type	genamap_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	yes
line	yes		z values	yes
none	no			

Reader Overview

The GenaMap reader scans the map header file (.FH file) and uses it to determine if the data contains an arc/node (Type 4, 5) map or a text map.

When reading Type 4 or Type 5 maps, the GenaMap reader extracts all tagged features. If the tagging status of the input map is not complete, the reader will also extract the untagged features.

Extraction of user attribution is also supported for the input Type 4 and Type 5 maps. When the attribute hookup status of the Type 4 map is set, the features are extracted with their associated attributes. The GenaMap reader looks for the associated attributes in the attribute table, located in the **ZF03** directory, referred to in the map header.

When reading Type 10 maps, the GenaMap reader extracts all text features that are contained in the **.FF** file.

Reader Directives

The suffixes listed are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the GenaMap reader is **GENAMAP**.

DATASET

Required/Optional: *Required*

The value for this directive is the GenaMap **.FH** file.

Example:

A typical mapping file fragment specifying an input GenaMap Type 4 dataset directory looks like:

```
GENAMAP_DATASET /usr/data/genamap/ZF04/MAP/MAP.FH
```

Workbench Parameter: *Source Genasys GenaMap File(s)*

FORCE_ATTR_HOOKUP

Required/Optional: *Optional*

Forces the reader to attempt translation of Type 3 maps when reading Type 4 or Type 5 maps even if the arc/node map header states that the attribute hookup was not complete.

Valid values: *YES | NO*

Example:

The following forces the reader to attempt reading the Type 3 maps, effectively ignoring the attribute hookup status flag for the arc/node maps:

```
GENAMAP_FORCE_ATTR_HOOKUP yes
```

SCALE_OF_TRUE_DISPLAY

Required/Optional: *Optional*

This directive allows the user to override the scale of true display found in the GenaMap map header. This may be useful for a more accurate control of the resulting text feature's text size.

Valid values: *positive integers*

Example:

```
GENAMAP_SCALE_OF_TRUE_DISPLAY 1000
```

ALIGN_TEXT_COORDS_TO_LOWER_LEFT

Required/Optional: *Optional*

This directive allows the reader to align all text features into a lower-left justification. The original text justification may be found in the text feature's **genamap_original_justification** attribute.

Valid values: *YES | NO*

If the automatic transformation for all text features into lower-left justification is not desired, then this keyword should be set to **NO**.

Default: *YES*

Example:

The GenaMap justification in this case may also be found in the text feature's `genamap_original_justification` attribute.

```
GENAMAP_ALIGN_TEXT_COORDS_TO_LOWER_LEFT NO
```

DATA_IN_BIG_ENDIAN

Required/Optional: *Optional*

This directive allows the reader to specify that the data is in big endian or little endian.

Valid values: *YES* | *NO*

Default: *YES*

Example:

The following specifies that the data to be read in is little endian:

```
GENAMAP_DATA_IN_BIG_ENDIAN NO
```

Workbench Parameter: *Data in Big Endian*

TEXT_TRANSFORM_AFFINE_COEFF

Required/Optional: *Optional*

Allows the user to specify the coefficients for a 2D affine transformation that is to be performed on a GenaMap text feature coordinate. Affine transformations include translations, rotations, scalings, and reflections. One of the uses of this directive is to allow shifting the position of the text coordinate in any direction on the x or y axis.

Values:

The range of values for this directive are: “*a b c d e f*” where *a, b, c, d, e,* and *f* must be real numbers, white-space-separated, and enclosed within double quotation marks. *a, b, c, d, e,* and *f* are coefficients for the equations:

$$\begin{aligned}x' &= ax + by + c \\y' &= dx + ey + f\end{aligned}$$

Default: *NO*

The following will shift all of the text coordinates read 5 units down the y-axis:

```
GENAMAP_TEXT_TRANSFORM_AFFINE_COEFF "1 0 0 0 1 -5.0"
```

It is possible to apply an affine transformation on the coordinate of a text selectively according to the value of its GenaMap alignment. The following nine directives may be used in conjunction with, but will override the affine transformation of, the generic `TEXT_TRANSFORM_AFFINE_COEFF` directive:

- `TEXT_TRANSFORM_COEFF_LL` - Applies the specified affine transformation to GenaMap text features having lower-left justifications.
- `TEXT_TRANSFORM_COEFF_LC` - Applies the specified affine transformation to GenaMap text features having lower-center justifications.
- `TEXT_TRANSFORM_COEFF_LR` - Applies the specified affine transformation to GenaMap text features having lower-right justifications.
- `TEXT_TRANSFORM_COEFF_CL` - Applies the specified affine transformation to GenaMap text features having center-left justifications.
- `TEXT_TRANSFORM_COEFF_CM` - Applies the specified affine transformation to GenaMap text features having center-middle justifications.
- `TEXT_TRANSFORM_COEFF_CR` - Applies the specified affine transformation to GenaMap text features having center-right justifications.

- **TEXT_TRANSFORM_COEFF_UL** - Applies the specified affine transformation to GenaMap text features having upper-left justifications.
- **TEXT_TRANSFORM_COEFF_UC** - Applies the specified affine transformation to GenaMap text features having upper-center justifications.
- **TEXT_TRANSFORM_COEFF_UR** - Applies the specified affine transformation to GenaMap text features having upper-right justifications.

For example, the following directives shift lower-left justified text 5 units down the y-axis, while text with other justifications are shifted 10 units up the y-axis:

```
GENAMAP_TEXT_TRANSFORM_AFFINE_COEFF_LL "1 0 0 0 1 -5.0"
GENAMAP_TEXT_TRANSFORM_AFFINE_COEFF "1 0 0 0 1 10.0"
```

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

✳ Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

✳ Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

✳ Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

✳ Workbench Parameter

Additional Attributes to Expose

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Each feature returned by the GenaMap reader has its feature type set to the name of the input map. All GenaMap FME features contain the `genamap_type` attribute that identifies the geometric type.

Attribute Name	Contents
<code>genamap_type</code>	The type of geometric entity stored within the feature. The valid values are listed below: <code>genamap_point</code> <code>genamap_line</code> <code>genamap_area</code> <code>genamap_text</code>

General Attributes

All GenaMap FME features contain the following attribute:

Attribute Name	Contents
genamap_tag	The primary GenaMap attribute.

Type 4 and Type 5 maps carry the additional attribute:

Attribute Name	Contents
genamap_symbology_id	The GenaMap symbolization pointer value.

Points

genamap_type: genamap_point

GenaMap point features specify point features defined by a single x,y or x,y,z coordinate.

Lines

genamap_type: genamap_line

GenaMap line features specify linear features defined by a sequence of x,y or x,y,z coordinates.

Polygons

genamap_type: genamap_polygon

GenaMap polygon features specify area (polygonal) features. The polygon may contain holes.

Text

genamap_type: genamap_text

The GenaMap text features are extracted from GenaMap Type 10 maps. Each text feature has a single x and y coordinate. The text coordinate point is located at the lower left of the text string. The original GenaMap origin point and justification are also stored in the text features as the attributes genamap_original_x, genamap_original_y, and genamap_original_justification. Text features have the following special attributes associated with them.

Attribute Name	Contents
genamap_text_string	The text string. Range: Any character string
genamap_rotation	The rotation of the text measured in degrees counter-clockwise from horizontal. Range: 0...360
genamap_width	The width of each text in ground units. Range: Any real number ≥ 0
genamap_height	The height of each text in ground units. Range: Any real number ≥ 0
genamap_original_justification	The original GenaMap alignment of the text based on the original GenaMap text origin point. Range: upper_left upper_center upper_right

Attribute Name	Contents
	center_left center_middle center_right lower_left lower_center lower_right
genamap_original_x	The original GenaMap text origin point x coordinate.
genamap_original_y	The original GenaMap text origin point y coordinate.
genamap_char_rotation	The rotation of each character in the text string measured in degrees counterclockwise from the horizontal. Range: 0...360
genamap_slant	The slant for each character in the text string. A negative angle gives a clockwise slant; a positive angle gives a counterclockwise slant. Range: -90...90
genamap_font	The GenaMap lettering style. Range: Any valid GenaMap system font. integer ≥ 0
genamap_color	The GenaMap color of the text. Range: GenaMap color index. integer ≥ 0

Geographic Data Management System (GDMS) Reader

The Geographic Data Management System (GDMS) Reader allows FME to read files in the GDMS format.

All three **VERTICES**, **CROSSREF**, and **TEXTDATA** input file types are supported for import.

Overview

GDMS is a Wang-based mapping system used by municipalities around the world. GDMS is first generation mapping system technology, now owned by ESRI and called the Spatial Database Engine (SDE). GDMS supports several geometry types, annotation, and only limited attribution.

The GDMS File reader module provides FME with access to the three GDMS file formats known as **VERTICES**, **CROSSREF**, and **TEXTDATA**.

The **VERTICES** and **CROSSREF** files should be provided as a matching pair, where the **VERTICES** file holds the geometry of features and the **CROSSREF** file holds attribute information. Points, line, polygons, and donuts are contained in these files.

The **TEXTDATA** file holds both geometry and attributes for text annotation features. These features include text, poly-line or polygon, circle, and symbol elements.

GDMS data files are binary and hold two-dimensional (2D) features.

While GDMS data sets consist of the three separate files, as described above, the precise format of these files is specified using **DEF** lines within the mapping file. There is no default extension that the FME recognizes as a GDMS input file.

FME does not automatically translate GDMS files, as some modification to the DEF lines is required. FME does, however, automatically generate a mapping file that can be used as a good starting point for customized GDMS translations.

GDMS Quick Facts

Format Type Identifier	GDMS
Reader/Writer	Reader
Licensing Level	Base
Dependencies	None
Dataset Type	Directory
Feature Type	Geometry-based name
Typical File Extensions	N/A
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Not applicable
Transaction Support	No
Geometry Type	gdms_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	yes		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	yes
line	yes		z values	no
none	no			

Reader Overview

First the GDMS reader parses the **DEF** lines and keywords to determine the location of the input data sets, as well as their precise format. The FME then proceeds to the following two steps:

1. If no **VERTICES** or **CROSSREF** files are specified or found, this step is ignored.

The GDMS reader opens the **VERTICES** and **CROSSREF** input files and immediately starts reading from both files, using the **Map Layer** and **Unique ID** to attach attributes from the **CROSSREF** file to the

VERTICES appropriate features. The GDMS reader then returns these features to the rest of the FME for processing.

It is assumed that the **VERTICES** and **CROSSREF** files are both sorted by **Map Layer** and **Unique ID** before these files are passed to the FME. Note that features within the **VERTICES** may have zero, one, or more corresponding **CROSSREF** entries.

2. If no **TEXTDATA** file is specified or found, this step is ignored.

The GDMS reader opens the **TEXTDATA** input file and immediately starts reading features, returning them to the rest of the FME for processing.

Each returned feature has its feature type set to the geometric type of the feature, as follows: **gdms_point**, **gdms_line**, **gdms_polygon**, **gdms_text_symbol**, **gdms_text_line**, **gdms_text_polygon**, **gdms_annotation**.

Reader Directives

The suffixes listed are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the GDMS reader is **GDMS**.

DATASET

Required/Optional: *Required*

This keyword's value is the directory containing the GDMS files to be read.

Example:

```
GDMS_DATASET C:\Data\GDMS\Input
```

Workbench Parameter: *Source Geographic Data Management System (GDMS) Directory*

SYSTEM_UNITS

Required/Optional: *Optional*

This setting determines the scaling factor of coordinates within the GDMS files. All coordinates are multiplied by this value. The default value is **1.0**.

Example:

```
GDMS_SYSTEM_UNITS 1000.0
```

X_OFFSET

This setting determines the X offset of coordinates within the GDMS files. This value is added to all X coordinates. The default value is **0**.

Example:

```
GDMS_X_OFFSET 300.0
```

Y_OFFSET

This setting determines the Y offset of coordinates within the GDMS files. This value is added to all Y coordinates. The default value is **0**.

Example:

```
GDMS_Y_OFFSET 300.0
```

DEF

Required/Optional: *Required*

The precise format of all three GDMS input file types must be specified on separate **DEF** lines before these files are read. These definition lines also provide the file names of each input file. The full path name of each file is determined by using the directory location from the **DATASET** keyword and the file name from the **DEF** line.

The syntax of the **DEF** lines depends on which file and record type it specifies. The first symbol of the **DEF** line indicates the further syntax and use of the rest of the **DEF** line as follows:

GDMS_DEF <definition type> ...

Each definition type is given in the following table. Refer to *Example Mapping File from GDMS to Shape* for a list of attribute names and definitions for each definition type.

Definition Type	Use and Syntax
VERTICES_RECORD_1	This DEF line specifies the VERTICES file location and the format of the VERTICES record type 1. The full syntax of this DEF line is: <pre><ReaderKeyword>_DEF VERTICES_RECORD_1 \ GDMS_VERTICES_FILENAME <filename> \ [<attrName> <fieldType>]*</pre>
VERTICES_RECORD_N	This DEF line specifies the format of the VERTICES record type 2. The full syntax of this DEF line is: <pre><ReaderKeyword>_DEF VERTICES_RECORD_N \ [<attrName> <fieldType>]*</pre>
CROSSREF_RECORD	This DEF line specifies the CROSSREF file location and the format of the CROSSREF record. The full syntax of this DEF line is: <pre><ReaderKeyword>_DEF CROSSREF_RECORD \ GDMS_CROSSREF_FILENAME <filename> \ [<attrName> <fieldType>]*</pre>
TEXT_LAYER_HEADER	This DEF line specifies the TEXTDATA file location and the format of the TEXTDATA header record. The size of the header record is also explicitly given. The full syntax of this DEF line is: <pre><ReaderKeyword>_DEF TEXT_LAYER_HEADER \ GDMS_TEXT_LAYER_FILENAME <filename> \ GDMS_RECORD_SIZE <number of bytes> \ [<attrName> <fieldType>]*</pre>
TEXT_LAYER_TEXT	This DEF line specifies the format of the Text element record within the TEXTDATA file. The size of the record is also explicitly given. The full syntax of this DEF line is: <pre><ReaderKeyword>_DEF TEXT_LAYER_TEXT \ GDMS_RECORD_SIZE <number of bytes> \ [<attrName> <fieldType>]*</pre>
TEXT_LAYER_POLY	This DEF line specifies the format of the Polyline/Polygon element record within the TEXTDATA file. The size of the record is also explicitly given. The full syntax of this DEF line is:

Definition Type	Use and Syntax
	<pre><ReaderKeyword>_DEF TEXT_LAYER_POLY \ GDMS_RECORD_SIZE <number of bytes> \ [<attrName> <fieldType>]*</pre>
TEXT_LAYER_CIRCLE	<p>This DEF line specifies the format of the Circle element record within the TEXTDATA file. The size of the record is also explicitly given. The full syntax of this DEF line is:</p> <pre><ReaderKeyword>_DEF TEXT_LAYER_CIRCLE \ GDMS_RECORD_SIZE <number of bytes> \ [<attrName> <fieldType>]*</pre>
TEXT_LAYER_SYMBOL	<p>This DEF line specifies the format of the Symbol element record within the TEXTDATA file. The size of the record is also explicitly given. The full syntax of this DEF line is:</p> <pre><ReaderKeyword>_DEF TEXT_LAYER_SYMBOL \ GDMS_RECORD_SIZE <number of bytes> \ [<attrName> <fieldType>]*</pre>

<attrName> — The attribute names may be specified as anything, but note that for several of the **DEF** linetypes some reserved attribute names are expected somewhere within the line.

<fieldType> — The field types specify the exact length and byte location where each attribute is found within the record being defined. The interpretation of the primitive date type for this region of the record is also indicated by the field type. The following table gives the possible field types.

Tip: These types are the same as the ones available to the CAT type in the Relational Table Reader.

Field Type	Description
Integer(<width>, <position>)	<p>Integer fields hold integer values stored in ASCII format.</p> <p>The width parameter is the total number of bytes allocated to the field.</p> <p>The position parameter is the starting byte of the field in the GDMS record. The bytes are numbered starting from 1.</p>
Real(<width>, <position>)	<p>Real fields hold floating point values stored in ASCII format.</p> <p>The width parameter is the total number of bytes allocated to the field, including the decimal point.</p> <p>The position parameter is the starting byte of the field in the GDMS record. The bytes are numbered starting from 1.</p>
String(<width>, <position>)	<p>String fields hold fixed length strings.</p> <p>The width parameter is the number of bytes that the field holds. When a character</p>

Field Type	Description
	field is retrieved, any padding blank bytes are stripped. The position parameter is the starting byte of the field in the GDMS record. The bytes are numbered starting from 1.
BigEndian(<width>, <position>)	BigEndian fields hold integer values stored in big-endian binary format. The width parameter is the total number of bytes allocated to the field. The only valid width values are: 1 to interpret the field as an 8 bit integer 2 to interpret the field as a 16 bit integer 4 to interpret the field as a 32 bit integer The position parameter is the starting byte of the field in the GDMS record. The bytes are numbered starting from 1.

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]

* Workbench Parameter

Clip To Envelope

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

GDMS features consist of both geometry and attributes. The attributes on the features depend on whether the feature is a result of input **VERTICES** and **CROSSREF** files, or whether it is from a **TEXTDATA** file. The **gdms_feature_type** identifies the entity's geometric type and has these ranges:

- gdms_line |
- gdms_text_line |
- gdms_point |
- gdms_text_symbol |
- gdms_polygon |
- gdms_text_polygon |
- gdms_annotation

GDMS Feature Types

The **DEF** lines specify many attributes for each GDMS record. All features derived from these records have all of these user-defined attributes. The only exceptions are the coordinate point and flag attributes. These attributes are used to construct the geometry of the feature and are not kept as attributes.

Refer to **DEF** for a listing of GDMS attributes.

Lines

gdms_type: gdms_line

GDMS line features represent linear features in 2D. Line features do not have any specific attributes.

gdms_type: gdms_text_line

GDMS line features represent linear features in 2D. Line features do not have any specific attributes.

Points

gdms_type: gdms_point

GDMS point features represent point features in 2D. Point features do not have any specific attributes.

gdms_type: gdms_text_symbol

GDMS text symbols represent point features in 2D and have the following specific attributes:

- gdms_angle
- gdms_length
- gdms_size
- gdms_symbol_number
- gdms_text_layer_number

Polygons

gdms_type: gdms_polygon

GDMS polygon features represent polygon features in 2D. Polygon features do not have any specific attributes.

gdms_type: gdms_text_polygon

GDMS text polygon features represent polygon features in 2D. Polygon features do not have any specific attributes.

Annotations

gdms_type: gdms_annotation

GDMS annotations represent point features in 2D and have the following specific attributes:

- gdms_text
- gdms_angle
- gdms_height
- gdms_text_layer_number

Example Mapping File from GDMS to Shape

The example below shows an FME mapping file used to translate some features from the GDMS format into ESRI Shape format. The mapping file defines the data set location, and gives the correlation lines between GDMS features and Shape.

```
#=====
#=====
#
# This mapping file was automatically generated by the FME
# on 08/18/98 14:53:26 for lossless translation between GDMS and SHAPE.
#
# You may edit this mapping file to customize its operation. Comments are
# placed throughout to assist you.
#
# Modification History:
#
# Name Date Description
#=====
#
#
#=====
```

```
#####  
# The following line defines the title presented to the user when this  
# mapping file is run through the FME GUI. You may modify this  
# if a more meaningful title would be appropriate.
```

GUI TITLE GDMS to SHAPE Translation

```
#####  
# The following line names the log file to which useful statistics about  
# the translation will be written. This line can be uncommented and  
# updated if you do wish to keep these statistics.
```

LOG_FILENAME translation.log

```
#####  
# The following line instructs the FME to log any features that do not  
# match any of the source feature patterns listed further down in  
# this file. If you are modifying this mapping file, this will be  
# useful to describe exactly which features you are losing  
# during translation, if the statistics indicate that features are  
# not being correlated or grouped. Uncorrelated features do not  
# match any source specification; ungrouped features do not have  
# any corresponding _DEF line.
```

FME_DEBUG UNGROUPED UNCORRELATED

```
#####  
# The following two lines define the type of reader and writer to be  
# used for this translation. If you want to translate your data  
# back into its original format, you may make a copy of this file  
# and switch the reader and writer types. If you rerun the FME, you  
# will get your original data back again (together with any modifications  
# you made in the meantime). Note that several formats are NOT  
# bi-directional (for example, GIF can only be used as a WRITER)  
# so a reverse translation may not always be possible.
```

READER_TYPE GDMS

WRITER_TYPE SHAPE

```
#####  
# The following GUI line prompts for a directory to be used as the  
# source of the GDMS files.  
# The user input is stored in a macro, which is then used to define  
# the data set to be read.
```

GUI DIRNAME SourceDataset Original GDMS File Directory:

```
#####  
# The DEF lines below are an outline for the VERTICES and CROSSREF file  
# structures.  
# Notice that most keywords and attribute names must be maintained  
# while their type and/or precise location may be freely altered.
```

GDMS_DEF VERTICES_RECORD_1 \
GDMS_VERTICES_FILENAME VERTICES \

gdms_map_layer BigEndianInt(2,1) \
gdms_ID BigEndianInt(4,3) \
gdms_record_number BigEndianInt(2,7) \
gdms_entity_type BigEndianInt(1,9) \
gdms_num_coords BigEndianInt(2,10) \
gdms_max_x BigEndianInt(4,12) \
gdms_min_x BigEndianInt(4,16) \
gdms_max_y BigEndianInt(4,20) \
gdms_min_y BigEndianInt(4,24) \
gdms_def_anno_x BigEndianInt(4,28) \
gdms_def_anno_y BigEndianInt(4,32) \
gdms_def_anno_angle BigEndianInt(2,36) \
gdms_def_symbol_num BigEndianInt(2,38) \
gdms_point_flag_1 BigEndianInt(1,54) \
gdms_point_flag_2 BigEndianInt(1,55) \
gdms_point_flag_3 BigEndianInt(1,56) \
gdms_point_flag_4 BigEndianInt(1,57) \
gdms_point_flag_5 BigEndianInt(1,58) \
gdms_point_flag_6 BigEndianInt(1,59) \
gdms_point_flag_7 BigEndianInt(1,60) \
gdms_point_flag_8 BigEndianInt(1,61) \
gdms_point_flag_9 BigEndianInt(1,62) \
gdms_point_flag_10 BigEndianInt(1,63) \
gdms_point_flag_11 BigEndianInt(1,64) \
gdms_point_flag_12 BigEndianInt(1,65) \
gdms_point_flag_13 BigEndianInt(1,66) \
gdms_point_flag_14 BigEndianInt(1,67) \
gdms_point_flag_15 BigEndianInt(1,68) \
gdms_point_x_1 BigEndianInt(4,69) \
gdms_point_y_1 BigEndianInt(4,73) \
gdms_point_x_2 BigEndianInt(4,77) \
gdms_point_y_2 BigEndianInt(4,81) \
gdms_point_x_3 BigEndianInt(4,85) \
gdms_point_y_3 BigEndianInt(4,89) \
gdms_point_x_4 BigEndianInt(4,93) \
gdms_point_y_4 BigEndianInt(4,97) \
gdms_point_x_5 BigEndianInt(4,101) \
gdms_point_y_5 BigEndianInt(4,105) \
gdms_point_x_6 BigEndianInt(4,109) \
gdms_point_y_6 BigEndianInt(4,113) \
gdms_point_x_7 BigEndianInt(4,117) \
gdms_point_y_7 BigEndianInt(4,121) \
gdms_point_x_8 BigEndianInt(4,125) \
gdms_point_y_8 BigEndianInt(4,129) \
gdms_point_x_9 BigEndianInt(4,133) \
gdms_point_y_9 BigEndianInt(4,137) \
gdms_point_x_10 BigEndianInt(4,141) \
gdms_point_y_10 BigEndianInt(4,145) \
gdms_point_x_11 BigEndianInt(4,149) \
gdms_point_y_11 BigEndianInt(4,153) \
gdms_point_x_12 BigEndianInt(4,157) \
gdms_point_y_12 BigEndianInt(4,161) \
gdms_point_x_13 BigEndianInt(4,165) \
gdms_point_y_13 BigEndianInt(4,169) \
gdms_point_x_14 BigEndianInt(4,173) \
gdms_point_y_14 BigEndianInt(4,177) \
gdms_point_x_15 BigEndianInt(4,181) \

gdms_point_y_15 BigEndianInt(4,185)

GDMS_DEF VERTICES_RECORD_N\
gdms_map_layer BigEndianInt(2,1) \
gdms_ID BigEndianInt(4,3) \
gdms_record_number BigEndianInt(2,7) \
gdms_point_flag_1 BigEndianInt(1,9) \
gdms_point_flag_2 BigEndianInt(1,10) \
gdms_point_flag_3 BigEndianInt(1,11) \
gdms_point_flag_4 BigEndianInt(1,12) \
gdms_point_flag_5 BigEndianInt(1,13) \
gdms_point_flag_6 BigEndianInt(1,14) \
gdms_point_flag_7 BigEndianInt(1,15) \
gdms_point_flag_8 BigEndianInt(1,16) \
gdms_point_flag_9 BigEndianInt(1,17) \
gdms_point_flag_10 BigEndianInt(1,18) \
gdms_point_flag_11 BigEndianInt(1,19) \
gdms_point_flag_12 BigEndianInt(1,20) \
gdms_point_flag_13 BigEndianInt(1,21) \
gdms_point_flag_14 BigEndianInt(1,22) \
gdms_point_flag_15 BigEndianInt(1,23) \
gdms_point_flag_16 BigEndianInt(1,24) \
gdms_point_flag_17 BigEndianInt(1,25) \
gdms_point_flag_18 BigEndianInt(1,26) \
gdms_point_flag_19 BigEndianInt(1,27) \
gdms_point_flag_20 BigEndianInt(1,28) \
gdms_point_x_1 BigEndianInt(4,29) \
gdms_point_y_1 BigEndianInt(4,33) \
gdms_point_x_2 BigEndianInt(4,37) \
gdms_point_y_2 BigEndianInt(4,41) \
gdms_point_x_3 BigEndianInt(4,45) \
gdms_point_y_3 BigEndianInt(4,49) \
gdms_point_x_4 BigEndianInt(4,53) \
gdms_point_y_4 BigEndianInt(4,57) \
gdms_point_x_5 BigEndianInt(4,61) \
gdms_point_y_5 BigEndianInt(4,65) \
gdms_point_x_6 BigEndianInt(4,69) \
gdms_point_y_6 BigEndianInt(4,73) \
gdms_point_x_7 BigEndianInt(4,77) \
gdms_point_y_7 BigEndianInt(4,81) \
gdms_point_x_8 BigEndianInt(4,85) \
gdms_point_y_8 BigEndianInt(4,89) \
gdms_point_x_9 BigEndianInt(4,93) \
gdms_point_y_9 BigEndianInt(4,97) \
gdms_point_x_10 BigEndianInt(4,101) \
gdms_point_y_10 BigEndianInt(4,105) \
gdms_point_x_11 BigEndianInt(4,109) \
gdms_point_y_11 BigEndianInt(4,113) \
gdms_point_x_12 BigEndianInt(4,117) \
gdms_point_y_12 BigEndianInt(4,121) \
gdms_point_x_13 BigEndianInt(4,125) \
gdms_point_y_13 BigEndianInt(4,129) \
gdms_point_x_14 BigEndianInt(4,133) \
gdms_point_y_14 BigEndianInt(4,137) \
gdms_point_x_15 BigEndianInt(4,141) \
gdms_point_y_15 BigEndianInt(4,145) \
gdms_point_x_16 BigEndianInt(4,149) \

```
gdms_point_y_16 BigEndianInt(4,153) \  
gdms_point_x_17 BigEndianInt(4,157) \  
gdms_point_y_17 BigEndianInt(4,161) \  
gdms_point_x_18 BigEndianInt(4,165) \  
gdms_point_y_18 BigEndianInt(4,169) \  
gdms_point_x_19 BigEndianInt(4,173) \  
gdms_point_y_19 BigEndianInt(4,177) \  
gdms_point_x_20 BigEndianInt(4,181) \  
gdms_point_y_20 BigEndianInt(4,185)
```

```
GDMS_DEF CROSSREF_RECORD \  
GDMS_CROSSREF_FILENAME CROSSREF \  
gdms_map_layer BigEndianInt(2,1) \  
gdms_ID BigEndianInt(4,3) \  
gdms_attr_number BigEndianInt(1,7) \  
gdms_map_layer_2 BigEndianInt(2,8) \  
gdms_attr_number_2 BigEndianInt(1,10) \  
gdms_attr_value String(48,11) \  
gdms_hex_date_yymmdd String(3,59)
```

```
#=====\  
# The DEF lines below are an outline for the TEXTDATA file structure.  
# Notice that most keywords and attribute names must be maintained  
# while their type and/or precise location may be freely altered.
```

```
GDMS_DEF TEXT_LAYER_HEADER \  
GDMS_TEXT_LAYER_FILENAME TEXTDATA \  
GDMS_RECORD_SIZE 44 \  
gdms_text_layer_number BigEndianInt(2,1) \  
gdms_grid_x String(3,3) \  
gdms_grid_y String(3,6) \  
gdms_ID BigEndianInt(4,9) \  
gdms_record_number BigEndianInt(2,13) \  
gdms_unique_flag BigEndianInt(1,15) \  
gdms_max_x BigEndianInt(4,17) \  
gdms_min_x BigEndianInt(4,21) \  
gdms_max_y BigEndianInt(4,25) \  
gdms_min_y BigEndianInt(4,29) \  
gdms_ID_point_x BigEndianInt(4,33) \  
gdms_ID_point_y BigEndianInt(4,37) \  
gdms_length BigEndianInt(2,41) \  
gdms_record_type BigEndianInt(2,43)
```

```
GDMS_DEF TEXT_LAYER_TEXT \  
GDMS_RECORD_SIZE 44 \  
gdms_element_type BigEndianInt(4,1) \  
gdms_alligned_length BigEndianInt(2,5) \  
gdms_true_length BigEndianInt(2,7) \  
gdms_height BigEndianInt(4,9) \  
gdms_width BigEndianInt(4,13) \  
gdms_spacing BigEndianInt(4,17) \  
gdms_angle BigEndianInt(2,21) \  
gdms_annotation_x BigEndianInt(4,37) \  
gdms_annotation_y BigEndianInt(4,41)
```

```
GDMS_DEF TEXT_LAYER_POLY \  

```

```
GDMS_RECORD_SIZE 12\  
gdms_element_type BigEndianInt(4,1) \  
gdms_point_count BigEndianInt(2,5) \  
gdms_fill_flag BigEndianInt(1,7)
```

```
GDMS_DEF TEXT_LAYER_CIRCLE\  
GDMS_RECORD_SIZE 20\  
gdms_element_type BigEndianInt(4,1) \  
gdms_center_x BigEndianInt(4,5) \  
gdms_center_y BigEndianInt(4,9) \  
gdms_radius BigEndianInt(4,13) \  
gdms_fill_flag BigEndianInt(1,17)
```

```
GDMS_DEF TEXT_LAYER_SYMBOL\  
GDMS_RECORD_SIZE 20\  
gdms_element_type BigEndianInt(4,1) \  
gdms_symbol_x BigEndianInt(4,5) \  
gdms_symbol_y BigEndianInt(4,9) \  
gdms_symbol_number BigEndianInt(2,13) \  
gdms_angle BigEndianInt(2,15) \  
gdms_size BigEndianInt(4,17)
```

```
=====
```

The lines below are used for scaling and shifting the input GDMS data set.
All features will have their coordinates scaled and shifted as outlined
below, but their attributes (such as a circle radius or text size) will
not be affected.

```
GDMS_SYSTEM_UNITS 100.0  
GDMS_X_OFFSET 0  
GDMS_Y_OFFSET 0
```

```
GDMS_DATASET "$(SourceDataset)"
```

```
=====
```

The following GUI line prompts for a directory to be used as the
the destination for the ESRI SHAPE files.
The user input is stored in a macro, which is then used to define
the data set to be written.

```
GUI DIRNAME DestDataset Destination Shape File Directory:
```

```
SHAPE_DATASET "$(DestDataset)"
```

```
=====
```

The main body of the mapping file starts here. Each of the
_DEF lines describes the data model of the particular feature
type, and the correlation lines describe how the feature is
transformed from the source type to the destination type.
You may edit the following lines to add or remove attributes, change
attribute definitions, or invoke other FME functions as the
features are translated.

```
=====
```

```
SHAPE_DEF gdms_points\  
=====
```

SHAPE_GEOMETRY shape_point \
1 char(48) \
2 char(48) \
3 char(48) \
4 char(48) \
5 char(48) \
6 char(48) \
7 char(48) \
8 char(48) \
9 char(48) \
MAP_LAYER number(5,0) \
GDMS_ID number(5,0) \
NUMBER number(5,0) \
TYPE number(5,0) \
NUM_COORDS number(5,0) \
GDMS_MAX_X number(11,0) \
GDMS_MIN_X number(11,0) \
GDMS_MAX_Y number(11,0) \
GDMS_MIN_Y number(11,0) \
DEF_ANNO_X number(11,0) \
DEF_ANNO_Y number(11,0) \
ANNO_ANGLE number(11,0) \
SYMBOL_NUM number(11,0)

GDMS gdms_point \
gdms_type gdms_point \
gdms_attribute_1 %gdms_attribute_1 \
gdms_attribute_2 %gdms_attribute_2 \
gdms_attribute_3 %gdms_attribute_3 \
gdms_attribute_4 %gdms_attribute_4 \
gdms_attribute_5 %gdms_attribute_5 \
gdms_attribute_6 %gdms_attribute_6 \
gdms_attribute_7 %gdms_attribute_7 \
gdms_attribute_8 %gdms_attribute_8 \
gdms_attribute_9 %gdms_attribute_9 \
gdms_map_layer %gdms_map_layer \
gdms_ID %gdms_id \
gdms_record_number %gdms_record_number \
gdms_entity_type %gdms_entity_type \
gdms_num_coords %gdms_num_coords \
gdms_max_x %gdms_max_x \
gdms_min_x %gdms_min_x \
gdms_max_y %gdms_max_y \
gdms_min_y %gdms_min_y \
gdms_def_anno_x %gdms_def_anno_x \
gdms_def_anno_y %gdms_def_anno_y \
gdms_def_anno_angle %gdms_def_anno_angle \
gdms_def_symbol_num %gdms_def_symbol_num

SHAPE gdms_points \
1 %gdms_attribute_1 \
2 %gdms_attribute_2 \
3 %gdms_attribute_3 \
4 %gdms_attribute_4 \
5 %gdms_attribute_5 \
6 %gdms_attribute_6 \
7 %gdms_attribute_7 \

8 %gdms_attribute_8 \
9 %gdms_attribute_9 \
MAP_LAYER %gdms_map_layer \
GDMS_ID %gdms_id \
NUMBER %gdms_record_number \
TYPE %gdms_entity_type \
NUM_COORDS %gdms_num_coords \
GDMS_MAX_X %gdms_max_x \
GDMS_MIN_X %gdms_min_x \
GDMS_MAX_Y %gdms_max_y \
GDMS_MIN_Y %gdms_min_y \
DEF_ANNO_X %gdms_def_anno_x \
DEF_ANNO_Y %gdms_def_anno_y \
ANNO_ANGLE %gdms_def_anno_angle \
SYMBOL_NUM %gdms_def_symbol_num

#=====

SHAPE_DEF gdms_lines \
SHAPE_GEOMETRY shape_polyline \
1 char(48) \
2 char(48) \
3 char(48) \
4 char(48) \
5 char(48) \
6 char(48) \
7 char(48) \
8 char(48) \
9 char(48) \
MAP_LAYER number(5,0) \
GDMS_ID number(5,0) \
NUMBER number(5,0) \
TYPE number(5,0) \
NUM_COORDS number(5,0) \
GDMS_MAX_X number(11,0) \
GDMS_MIN_X number(11,0) \
GDMS_MAX_Y number(11,0) \
GDMS_MIN_Y number(11,0) \
DEF_ANNO_X number(11,0) \
DEF_ANNO_Y number(11,0) \
ANNO_ANGLE number(11,0) \
SYMBOL_NUM number(11,0)

GDMS gdms_line \
gdms_type gdms_line \
gdms_attribute_1 %gdms_attribute_1 \
gdms_attribute_2 %gdms_attribute_2 \
gdms_attribute_3 %gdms_attribute_3 \
gdms_attribute_4 %gdms_attribute_4 \
gdms_attribute_5 %gdms_attribute_5 \
gdms_attribute_6 %gdms_attribute_6 \
gdms_attribute_7 %gdms_attribute_7 \
gdms_attribute_8 %gdms_attribute_8 \
gdms_attribute_9 %gdms_attribute_9 \
gdms_map_layer %gdms_map_layer \
gdms_ID %gdms_id \
gdms_record_number %gdms_record_number \

gdms_entity_type %gdms_entity_type\
gdms_num_coords %gdms_num_coords\
gdms_max_x %gdms_max_x\
gdms_min_x %gdms_min_x\
gdms_max_y %gdms_max_y\
gdms_min_y %gdms_min_y\
gdms_def_anno_x %gdms_def_anno_x\
gdms_def_anno_y %gdms_def_anno_y\
gdms_def_anno_angle %gdms_def_anno_angle\
gdms_def_symbol_num %gdms_def_symbol_num

SHAPE gdms_lines\
1 %gdms_attribute_1\
2 %gdms_attribute_2\
3 %gdms_attribute_3\
4 %gdms_attribute_4\
5 %gdms_attribute_5\
6 %gdms_attribute_6\
7 %gdms_attribute_7\
8 %gdms_attribute_8\
9 %gdms_attribute_9\
MAP_LAYER %gdms_map_layer\
GDMS_ID %gdms_id\
NUMBER %gdms_record_number\
TYPE %gdms_entity_type\
NUM_COORDS %gdms_num_coords\
GDMS_MAX_X %gdms_max_x\
GDMS_MIN_X %gdms_min_x\
GDMS_MAX_Y %gdms_max_y\
GDMS_MIN_Y %gdms_min_y\
DEF_ANNO_X %gdms_def_anno_x\
DEF_ANNO_Y %gdms_def_anno_y\
ANNO_ANGLE %gdms_def_anno_angle\
SYMBOL_NUM %gdms_def_symbol_num

#=====

SHAPE_DEF gdms_annotations\
SHAPE_GEOMETRY shape_point\
TEXT_ANGLE number(14,6)\
TEXT_SIZE number(14,6)\
TEXTSTRING char(254)\
GDMS_ID number(10,0)\
ID_POINT_X number(10,0)\
ID_POINT_Y number(10,0)\
LENGTH0 number(5,0)\
TYPE0 number(1,0)\
GRID_X char(3)\
GRID_Y char(3)\
LENGTH number(10,0)\
GDMS_MAX_X number(10,0)\
GDMS_MAX_Y number(10,0)\
GDMS_MIN_X number(10,0)\
GDMS_MIN_Y number(10,0)\
NUMBER number(5,0)\
TYPE1 number(5,0)\
SPACING number(10,0)

NUMBER1 number(5,0) \
LENGTH1 number(5,0) \
FLAG number(1,0) \
GDMS_WIDTH number(10,0) \
GDMS_TEXT char(254) \
GDMS_ANGLE number(5,0) \
HEIGHT number(10,0)

GDMSgdms_annotation \
gdms_ID %gdms_id \
gdms_ID_point_x %gdms_id_point_x \
gdms_ID_point_y %gdms_id_point_y \
gdms_aligned_length %gdms_aligned_length \
gdms_element_type %gdms_element_type \
gdms_grid_x %gdms_grid_x \
gdms_grid_y %gdms_grid_y \
gdms_length %gdms_length \
gdms_max_x %gdms_max_x \
gdms_max_y %gdms_max_y \
gdms_min_x %gdms_min_x \
gdms_min_y %gdms_min_y \
gdms_record_number %gdms_record_number \
gdms_record_type %gdms_record_type \
gdms_spacing %gdms_spacing \
gdms_text_layer_number %gdms_text_layer_number \
gdms_true_length %gdms_true_length \
gdms_unique_flag %gdms_unique_flag \
gdms_width %gdms_width \
gdms_text %gdms_text \
gdms_angle %gdms_angle \
gdms_height %gdms_height

SHAPE gdms_annotations \
GDMS_ID %gdms_id \
ID_POINT_X %gdms_id_point_x \
ID_POINT_Y %gdms_id_point_y \
LENGTH0 %gdms_aligned_length \
TYPE0 %gdms_element_type \
GRID_X %gdms_grid_x \
GRID_Y %gdms_grid_y \
LENGTH %gdms_length \
GDMS_MAX_X %gdms_max_x \
GDMS_MAX_Y %gdms_max_y \
GDMS_MIN_X %gdms_min_x \
GDMS_MIN_Y %gdms_min_y \
NUMBER %gdms_record_number \
TYPE1 %gdms_record_type \
SPACING %gdms_spacing \
NUMBER1 %gdms_text_layer_number \
LENGTH1 %gdms_true_length \
FLAG %gdms_unique_flag \
GDMS_WIDTH %gdms_width \
TEXTSTRING %gdms_text \
TEXT_ANGLE %gdms_angle \
TEXT_SIZE %gdms_height

#=====

SHAPE_DEF gdms_polygons \
SHAPE_GEOMETRY shape_polygon \
GDMS_ID number(10,0) \
ID_POINT_X number(10,0) \
ID_POINT_Y number(10,0) \
TYPE0 number(1,0) \
FILL_FLAG number(1,0) \
GRID_X char(3) \
GRID_Y char(3) \
LENGTH number(10,0) \
GDMS_MAX_X number(10,0) \
GDMS_MAX_Y number(10,0) \
GDMS_MIN_X number(10,0) \
GDMS_MIN_Y number(10,0) \
COUNT number(5,0) \
NUMBER number(5,0) \
TYPE1 number(5,0) \
NUMBER1 number(5,0) \
FLAG number(1,0)

GDMS gdms_text_polygon \
gdms_ID %gdms_id \
gdms_ID_point_x %gdms_id_point_x \
gdms_ID_point_y %gdms_id_point_y \
gdms_element_type %gdms_element_type \
gdms_fill_flag %gdms_fill_flag \
gdms_grid_x %gdms_grid_x \
gdms_grid_y %gdms_grid_y \
gdms_length %gdms_length \
gdms_max_x %gdms_max_x \
gdms_max_y %gdms_max_y \
gdms_min_x %gdms_min_x \
gdms_min_y %gdms_min_y \
gdms_point_count %gdms_point_count \
gdms_record_number %gdms_record_number \
gdms_record_type %gdms_record_type \
gdms_text_layer_number %gdms_text_layer_number \
gdms_unique_flag %gdms_unique_flag

SHAPE gdms_polygons \
GDMS_ID %gdms_id \
ID_POINT_X %gdms_id_point_x \
ID_POINT_Y %gdms_id_point_y \
TYPE0 %gdms_element_type \
FILL_FLAG %gdms_fill_flag \
GRID_X %gdms_grid_x \
GRID_Y %gdms_grid_y \
LENGTH %gdms_length \
GDMS_MAX_X %gdms_max_x \
GDMS_MAX_Y %gdms_max_y \
GDMS_MIN_X %gdms_min_x \
GDMS_MIN_Y %gdms_min_y \
COUNT %gdms_point_count \
NUMBER %gdms_record_number \
TYPE1 %gdms_record_type \
NUMBER1 %gdms_text_layer_number \

FLAG %gdms_unique_flag

GeoJSON (Geographic JavaScript Object Notation) Reader/Writer

Format Notes: This format is not supported by FME Base Edition.

GeoJSON is a standard for encoding spatial data in JSON structured text. FME currently supports the GeoJSON Format Specification Revision 1.0, which is available at <http://geojson.org>.

Overview

GeoJSON encodes both geometry and feature information into objects. It also provides support for geometry and feature collections.

GeoJSON represents geometry with a single JSON object. The type of geometry is identified by the value of the *type* key, which must be present in a GeoJSON object. Possible values of the *type* key are *Point*, *LineString*, *Polygon*, *MultiPoint*, *MultiLineString*, and *MultiPolygon*. The geometry coordinates are stored in the *coordinates* key of the geometry object.

```
{
  "type": "LineString",
  "coordinates": [ [100.0, 0.0], [101.0, 1.0] ]
}
```

An aggregate geometry is represented by a GeoJSON object which has a *type* key with value *GeometryCollection*. A *GeometryCollection* object must contain a *geometries* key whose value is an array containing GeoJSON geometry objects.

```
{
  "type": "GeometryCollection",
  "geometries": [
    {
      "type": "LineString",
      "coordinates": [ [100.0, 0.0], [101.0, 1.0] ]
    },
    {
      "type": "MultiPoint",
      "coordinates": [ [100.0, 0.0], [101.0, 1.0] ]
    }
  ]
}
```

A feature is represented by a GeoJSON object which has a *type* key with value *Feature*. A *Feature* object may contain a *geometry* key whose value is a GeoJSON geometry object or a GeoJSON *GeometryCollection* object. A *Feature* object may also contain a *properties* key whose value is an object containing attribute names and values.

```
{
  "type": "Feature",
  "geometry": {
    "type": "MultiPoint",
    "coordinates": [
      [102.0, 2.0],
      [103.0, 3.0]
    ]
  }
}
```

A collection of features is represented by a GeoJSON object which has a *type* key with value *FeatureCollection*. A *FeatureCollection* must contain a *features* key whose value is any array containing GeoJSON *Feature* objects.

```

{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "MultiPoint",
        "coordinates": [
          [102.0, 2.0],
          [103.0, 3.0]
        ]
      }
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "GeometryCollection",
        "geometries": [
          {
            "type": "LineString",
            "coordinates": [ [100.0, 0.0], [101.0, 1.0] ]
          },
          {
            "type": "MultiPoint",
            "coordinates": [ [100.0, 0.0], [101.0, 1.0] ]
          }
        ]
      }
    }
  ]
}

```

Coordinate systems are supported in GeoJSON through the use of a *crs* key. If a GeoJSON object has the *crs* key, it is assumed to represent the coordinate reference system of the included features or geometries. The value of the *crs* key must be an object containing both a *type* and a *properties* key. If the *crs* key is absent, the projection is assumed to be LL84. If the *type* key is set to 'name', then the feature(s) will be tagged with an OGC CRS URN. If the *type* key is set to 'link', it means that the *href* key is specified as a URL or a file path that contains the CRS info. In this case, the *href* parameter will be stored in the *json_crs_url* attribute, with its corresponding *type* in the *json_crs_url_type* attribute.

Named CRS

```

{
  "type": "Feature",
  "crs": {
    "type": "name",
    "properties": { "name": "urn:ogc:def:crs:OGC:1.3:CRS84" }
  },
  "geometry": {
    "type": "Point",
    "coordinates": [100.0, 0.0]
  }
}

```

Linked CRS

```

{
  "type": "Feature",
  "crs": {
    "type": "link",
    "properties": {
      "href": "http://spatialreference.org/ref/epsg/2001/proj4/",
      "type": "proj4"
    }
  },
  "geometry": {

```

```

    "type": "Point",
    "coordinates": [100.0, 0.0]
  }
}

```

GeoJSON Quick Facts

Format Type Identifier	GeoJSON
Reader/Writer	Reader/Writer
Licensing Level	Professional
Dependencies	None
Dataset Type	File/URL
Feature Type	Varies: schema is dependent on the source dataset
Typical File Extensions	.json
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Never
Schema Required	No
Transaction Support	No
Geometry Type	json_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	yes		z values	no
none	yes			

Reader Overview

The GeoJSON reader is capable of reading several different GeoJSON structures. If the base JSON element is a GeoJSON geometry object, then the reader will return a single FME feature with the given geometry. If the base JSON element is a GeoJSON *GeometryCollection* object, then the reader returns a single FME feature with an aggregate geometry. In both cases, the FME feature type will be *GeoJSON*.

If the base JSON element is a GeoJSON *Feature* object, then the GeoJSON reader will return a single FME feature. The feature geometry will be taken from the *geometry* key of the *Feature* object, and the feature attributes will be taken from the *properties* key of the *Feature* object. If the base JSON element is a GeoJSON *FeatureCollection* object, then the GeoJSON reader will return an FME feature for each element of the *features* array of the *FeatureCollection* object. In both cases, the FME feature type for each feature will be *GeoJSON*.

If the base JSON element is an array, then any GeoJSON objects in the array are converted into FME features as described above.

If the base JSON element is an object, but not a GeoJSON object, then any value which is a GeoJSON object is converted into FME features as described above, with the exception that the FME feature type is the key name of the corresponding GeoJSON object.

Coordinate Systems

The GeoJSON reader currently supports coordinate systems in EPSG, OGC URN, or URL format as described in the overview.

Reader Directives

The suffixes shown are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the GeoJSON reader is `GEOJSON`.

DATASET

Required/Optional: Required

The location of the GeoJSON file to be read. This can be the path to a local or network file, or a URL.

Examples:

```
GEOJSON_DATASET c:\json_sample.json
GEOJSON_DATASET \\path\to\network\file.json
GEOJSON_DATASET http://geojson.org/sample
```

* Workbench Parameter

Source GeoJSON File or URL

DELETE_DOWNLOAD_FILE

Required/Optional: Optional

If the value of this directive is 'Yes', then when the reader has finished reading downloaded GeoJSON text, it will delete the file that the text was downloaded to. The default value is 'Yes'. The value of this directive is only meaningful if the dataset is a URL.

Example:

```
GEOJSON_DELETE_DOWNLOAD_FILE No
```

Workbench Parameter: *Delete downloaded file*

PROXY_URL

Required/Optional: Optional

Specifies a proxy server that the reader will be use when accessing a URL dataset. The port number of the proxy server can be set in the URL, or by using the `PROXY_PORT` directive.

Example:

```
GEOJSON_PROXY_URL www.someproxy.net
GEOJSON_PROXY_URL www.someproxy.net:8080
```

Workbench Parameter: *Http Proxy URL*

PROXY_PORT

Required/Optional: Optional

Specifies the port number of the proxy server indicated by the `PROXY_URL` directive. This directive should only be used if the port number was not indicated in the `PROXY_URL` directive. This directive is ignored if the `PROXY_URL` directive has no value.

Example:

```
GEOJSON_PROXY_PORT 8080
```

Workbench Parameter: *Http Proxy Port*

PROXY_USERNAME

Required/Optional: Optional

Specifies the username to use when accessing a password protected proxy server. This directive is ignored if any of the `PROXY_URL`, `PROXY_PASSWORD` or `PROXY_AUTH_METHOD` directives have no value.

Example:

```
GEOJSON_PROXY_USERNAME someusername
```

Workbench Parameter: *Http Proxy Username*

PROXY_PASSWORD

Required/Optional: Optional

Specifies the password to use when accessing a password protected proxy server. This directive is ignored if any of the `PROXY_URL`, `PROXY_USERNAME` or `PROXY_AUTH_METHOD` directives have no value.

Example:

```
GEOJSON_PROXY_PASSWORD password1234
```

Workbench Parameter: *Http Proxy Password*

PROXY_AUTH_METHOD

Required/Optional: Optional

Specifies the authentication method to use when accessing a password protected proxy server. This directive is ignored if any of the `PROXY_URL`, `PROXY_USERNAME` or `PROXY_PASSWORD` directives have no value. Acceptable values for this directive are 'Basic' or 'Digest'.

Example:

```
GEOJSON_PROXY_AUTH_METHOD Basic
```

Workbench Parameter: *Http Proxy Authentication Method*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

*** Workbench Parameter**

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

`<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>`

*** Workbench Parameter**

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

`<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]`

*** Workbench Parameter**

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The GeoJSON writer writes out a single object, in which each key is an FME feature type, and the value of each key is a GeoJSON *FeatureCollection* object which contains the features of the given type.

Coordinate Systems

The GeoJSON writer currently supports coordinate systems in EPSG, OGC URN, or URL format as described in the overview. To write a URL coordinate system, ensure that the `json_crs_url` and `json_crs_url_type` attributes are complete and that no coordinate system is set on the feature. If a coordinate system is specified which violates the GeoJSON specifications, the coordinate system will be reprojected to LL84. If no coordinate system is available, no coordinate system will be output. However, it is important to note that the specifications state that GeoJSON objects not specifically tagged with a coordinate reference system are assumed to be the LL84. If multiple coordinate systems exist among features, the first feature will be used to determine the coordinate system for the feature collection.

Geometry

FME feature geometry is written out in a GeoJSON geometry object as the value of the *geometry* key in a *Feature* type GeoJSON object. Because GeoJSON only supports linear geometry, arcs will be stroked to lines, and ellipses will be stroked to polygons. Also, paths are simplified to a single line, and an FME feature with text geometry only has its location written; the text value is ignored.

The value of the *geometry* key for an FME feature with aggregate geometry will be a *GeometryCollection* object, whose *geometries* key will have an array of GeoJSON geometry objects as its value.

Writer Directives

The suffixes shown are prefixed by the current `<WriterKeyword>` in a mapping file. By default, the `<WriterKeyword>` for the GeoJSON writer is `GEOJSON`.

DATASET

Required/Optional: Required

The file to which the GeoJSON writer should write to. If the file does not exist it will be created.

Example:

```
GEOJSON_DATASET c:\geojson_file.json
```

Workbench Parameter: *Destination GeoJSON File*

WRITE_NULL_ATTRIBUTE_VALUES

Required/Optional: Optional

This directive specifies whether or not the object containing an FME feature's attributes should contain a key for attributes for which the feature has no value. Possible values for this directive are Yes and No. If the value is No, then the attributes object will only contain keys for which the FME feature has an attribute value. If the value of the directive is Yes, then the output JSON objects will contain keys for every attribute in the feature type schema, and keys for which an FME feature has no attribute value will have a *null* JSON value. The default value of this directive is No.

Example:

```
GEOJSON_WRITE_NULL_ATTRIBUTE_VALUES Yes
```

Workbench Parameter: *Write 'null' for attributes with no value*

STRICT_SPEC

Required/Optional: Optional

This determines whether output will adhere strictly to the GeoJSON grammar. An array will be used as the outermost element in order to represent multiple layers; a single layer will not be contained by an array. Possible values for this directive are YES and NO.

Example:

```
GEOJSON STRICT_SPEC YES
```

*** Workbench Parameter**

Fully conform to GeoJSON grammar

WRITER_CHARSET

Required/Optional: Optional

The character set encoding in which the GeoJSON text will be written. Possible values for this directive are UTF-8, UTF-16, UTF-16BE, UTF16-LE, UTF-32, UTF-32BE and UTF-32LE. If no character set is specified, the GeoJSON text will be written in the UTF-8 character set.

Example:

```
GEOJSON_WRITER_CHARSET UTF-16
```

Workbench Parameter: *Output Character Set*

WRITE_BOM

Required/Optional: Optional

The value of this directive specifies whether or not the GeoJSON writer should preface the JSON text with a byte order marker to indicate the endianness of the Unicode text. Possible values for this directive are Yes and No. The default value is No.

Example:

```
GEOJSON_WRITE_BOM Yes
```

Workbench Parameter: *Byte Order Marker*

JSONP_FUNC_NAME

Required/Optional: Optional

The value of this directive specifies the JSONP JavaScript function name that the user wants to wrap the GeoJSON file with. JSONP (JSON with Padding) is developed as a standard for grabbing JSON from external domains, which works well with AJAX calls.

The default value is null. If no value is set or the default is set, then the GeoJSON writer will output a GeoJSON file without the JSONP padding.

Example:

```
JSONP_FUNC_NAME getFeatures
```

*** Workbench Parameter**

JSONP Function Name

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Geometry

The geometry of GeoJSON features may be identified by the `json_type` attribute. The valid values for this attribute are:

<code>json_type</code>	Description
<code>json_no_geom</code>	FME Feature with no geometry.
<code>json_point</code>	Point feature.
<code>json_line</code>	Linear feature.
<code>json_polygon</code>	Simple polygon or donut feature.
<code>json_collection</code>	Feature with multiple geometries.

No Geometry

json_type: `json_no_geom`

Features with their `json_type` attribute set to `json_no_geom` do not contain any geometry data.

Points

json_type: `json_point`

Features with their `json_type` set to `json_point` are single coordinate features or an aggregate of single points.

Lines

json_type: `json_line`

Features with their `json_type` set to `json_line` are polyline features or an aggregate of polylines.

Areas

json_type: `json_polygon`

Features with their `json_type` set to `json_polygon` are polygon features which may or may not have interior boundaries, or an aggregate of such polygons.

Aggregates

json_type: `json_collection`

Features with their `json_type` set to `json_collection` are a heterogeneous collection of multiple geometries.

GeoRSS/RSS Feed Reader/Writer

Format Notes:

This format is not supported by FME Base Edition.

XML feeds are a popular method of publishing information to a set of subscribers. Using GeoRSS, an XML feed can be extended to include spatial data. The GeoRSS reader/ writer plug-in enables FME to read and write XML feeds and their spatial data extensions.

Overview

An XML feed can be in one of several different formats, with the most common formats being RSS and Atom. Both of these formats have a similar structure in that the feed contains metadata and a collection of entries. The specifications for the current versions of these formats can be found at <http://www.rssboard.org/rss-specification> and <http://tools.ietf.org/html/rfc4287> respectively.

Currently the GeoRSS reader supports RSS versions 0.91, 0.92 and 2.0, as well as Atom 0.3 and 1.0. The GeoRSS writer can output feeds in RSS 2.0 or Atom 1.0.

The GeoRSS specification defines a way to add spatial information to an XML feed. The GeoRSS reader and writer both support each of the three methods used to include spatial information: W3C Geo, GeoRSS Simple, and GML. Specifications for each of these methods can be found at <http://www.georss.org>.

GeoRSS Quick Facts

Format Type Identifier	GeoRSS
Reader/Writer	Reader/Writer
Licensing Level	Professional
Dependencies	None
Dataset Type	File/URL
Feature Type	Feed, Entry
Typical File Extensions	.atom .rss .xml
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	georss_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	yes		z values	yes
none	yes			

Reader Overview

The GeoRSS reader works by mapping an XML feed and its entries into FME features. A single FME feature is created for the feed meta-data, and a FME feature is created for each entry in the feed. Because each feed format has a similar structure, the same schema is used for every feed that the reader processes, regardless of the feed format and version. The reader can handle an XML feed from a local or network file, or a remote URL accessible via http or ftp. The reader can access these URL's directly, or it can be routed through a proxy server.

Normal Mode and Update Mode

The GeoRSS reader can be run in two modes: normal and update.

In normal mode, the reader will always return a feature for every feed entry that it processes.

In update mode, which can be enabled by a reader directive, the reader will only return entry features if they are new or updated. This is accomplished in two ways.

When invoked on a URL in update mode, the reader will save certain key HTTP headers from the response. When the reader is run again with the same URL, it will return these headers to the server, which will use them to determine if the feed has changed since the reader last requested it. If the feed has changed, the server will return it and the reader will proceed as normal. If the feed has not changed, the server will return an HTTP 304 status code, indicating that the feed has not changed, and the reader will quit without returning any features. Note that this functionality is only possible when the dataset is a remote URL.

When processing features in update mode, the reader will save the ID and modification date of each feed entry. In an RSS feed, which does not define an entry modification date, only the entry ID (taken from the `<guid>` element) will be saved. In an Atom feed, both the ID (taken from the `<id>` element) and the modification date (taken from the `<updated>` element) will be saved. If the reader is run again with the same dataset, a feed entry will be skipped if an identical modification date has already been saved for the the entry's ID. Note that this functionality will work for both file and URL datasets.

To provide additional control over update mode, a directive (`FEEDSTORE_ID`) can be set which allows the update mode to treat different datasets as identical for the purposes of determining updates. For instance, a user might wish to treat an RSS feed and a locally downloaded cache of a feed as identical for the purposes of determining updates.

The feed headers and entry modification information is stored in a separate database for each feed, which will be displayed in the log when the reader is run. Deleting this file will force the reader to treat a feed as if it had never been read before. The reader also keeps a database which maps feed locations (URLs or file paths) to database files. Deleting this file forces the reader to treat all feeds as if they had never been read before. The location of this file is configurable with a reader directive. The default file is `georssfmefeeds.sqlite` in the FME temp directory.

Geometry

The GeoRSS reader supports each of the three methods for extending an XML feed with spatial data. The reader also supports feeds and entries with aggregate geometries, even though these are not explicitly included in the GeoRSS

specification. This includes feeds and entries with multiple instances of the same data format, or combinations of the three spatial data formats, as in the following examples:

Example 1:

```
<feed>
  <entry>
    <georss:point>72.0 43.0</georss:point>
    <georss:point>23 -36</georss:point>
  </entry>
</feed>
```

Example 2:

```
<feed>
  <entry>
    <georss:point>72.0 43.0</georss:point>
    <georss:where>
      <gml:LineString>
        <gml:posList>5.0 5.0 6.0 6.0 4.3 -5.5</gml:posList>
      </gml:LineString>
    </georss:where>
  </entry>
</feed>
```

Coordinate Systems

The EPSG:4326 coordinate system is used for all features that contain W3C Geo or GeoRSS Simple geometry extensions. The GML geometry extension allows a different coordinate system to be set for each feature, with EPSG:4326 being the default.

If a feature contains only GML geometry extensions, then the feature's coordinate system will be set from the first extension. The lowest level non-default coordinate system from the first extension will be used, or if there are no coordinate systems specified, the default coordinate system will be used. The following examples illustrate the coordinate system logic used by the GeoRSS reader.

Example 1:

This example uses the default EPSG:4326 coordinate system, because the entry contains a GeoRSS Simple geometry extension. The EPSG:1234 coordinate system definition on the GML geometry extension is ignored.

```
<feed>
  <entry>
    <georss:point>72.0 43.0</georss:point>
    <georss:where>
      <gml:Point srsName="EPSG:1234">
        <gml:pos>72.0 43.0</gml:pos>
      </gml:Point>
    </georss:where>
  </entry>
</feed>
```

Example 2:

This example uses the default EPSG:4326 coordinate system, because the first extension does not define a coordinate system. The coordinate system defined on the second extension is ignored.

```
<feed>
  <entry>
    <georss:where>
      <gml:Point>
        <gml:pos>72.0 43.0</gml:pos>
      </gml:Point>
      <gml:Point srsName="EPSG:1234">
        <gml:pos>72.0 43.0</gml:pos>
      </gml:Point>
    </georss:where>
  </entry>
</feed>
```

```

    <gml:Point>
  </georss:where>
</entry>
</feed>

```

Example 3:

This example uses the EPSG:4321 coordinate system because it is the lowest level non-default coordinate system on the first extension. The EPSG:1234 and EPSG:6789 coordinate system definitions are ignored.

```

<feed>
  <entry>
    <georss:where>
      <gml:Point srsName="1234">
        <gml:pos srsName="4321">72.0 43.0</gml:pos>
      <gml:Point>
        <gml:Point srsName="6789">
          <gml:pos>72.0 43.0</gml:pos>
        <gml:Point>
      </georss:where>
    </entry>
  </feed>

```

Handling of Unknown XML Elements

The GeoRSS reader will not ignore any XML elements that it encounters. If the reader encounters XML that it cannot use to populate the predefined GeoRSS feature schema, it will simply add the XML to the feature as a new attribute.

The new attribute will be named based on the prefix and name of the unknown element. If the xml element has a prefix, the new attribute will be named **prefix_name**. If the element has no prefix, the new attribute name will be **_name**.

If the XML element has no attributes and only text content, the value of the new feature attribute will be the text content of the XML element. If the XML element contains XML attributes or non-text child elements, then the entire XML element will be the value of the new attribute.

Reader Directives

The suffixes shown are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the GeoRSS reader is **GEORSS**.

DATASET

Required/Optional: Required

The location of the XML file containing the feed. This can be the path to a local or network file, or a URL.

Examples:

```

GEORSS_DATASET c:\atomsample.xml
GEORSS_DATASET \\path\to\network\file.xml
GEORSS_DATASET http://earthquake.usgs.gov/eqcenter/recenteqsww/catalogs/eqs7day-M5.xml
GEORSS_DATASET ftp://192.168.80.2/tests/geometry/gml.atom.geomtest.xml

```

Workbench Parameter: *XML Feed*

URL_PARAMETERS

Required/Optional: Optional

The value of this directive is only used if the reader is accessing a dataset which is a URL. The value of the directive should be a space-separated list of space-separated name-value pairs. The name-value pairs will be added to the dataset URL. A value must be provided for each parameter name. An empty string ("") can be used to provide an empty parameter value.

Example:

```
GEORSS_DATASET http://localhost/trafficdata.xml
GEORSS_URL_PARAMETERS province bc city vancouver
```

This example will result in the reader accessing the following url:

```
http://localhost/trafficdata.xml?province=bc&city=vancouver
```

Workbench Parameter: *Additional URL Parameters*

READER_MODE

Required/Optional: Optional

Specifies which mode the reader should run in. This directive must be set to UPDATE to make the reader run in update mode. If it is set to any other value, or if the directive is not present, the reader will run in normal mode.

Example:

```
GEORSS_READER_MODE UPDATE
```

Workbench Parameter: *Reader Mode*

FEEDSTORE_DB_DIR

Required/Optional: Optional

This directive specifies the filesystem directory which contains the databases storing information about the feeds that the GeoRSS reader has processed. This directive is only read from the mapping file if the reader is running in update mode. If no value is set, the FME temp directory will be used.

Example:

```
GEORSS_FEEDSTORE_DB_DIR c:\georss_feeds
```

Workbench Parameter: *Feed Database Location*

FEEDSTORE_MAX_ENTRY_AGE

Required/Optional: Optional

This directive specifies the age in days at which entries will be deleted from a feed's database. When the reader is run in update mode, the reader will delete any entries from the database for this feed which are older than the specified value. This ensures the feed database does not become arbitrarily large. If the value of this directive is not specified, or is 0, no entries will be deleted.

Example:

```
GEORSS_FEEDSTORE_MAX_ENTRY_AGE 30
```

Workbench Parameter: *Max FeedStore Entry Age*

FEEDSTORE_ID

Required/Optional: Optional

This directive allows the user to specify which feeds should be treated as coming from the same source. It only has an effect if the reader is run in update mode. When this directive is set, the input file/URL is treated as identical to any other previous read input which has the same feedstore identifier. If no value for this directive is set, it is the same as setting it equal to the DATASET directive.

Example:

```
GEORSS_FEEDSTORE_ID http://www.safe.com/company/news/rss/rss.xml
```

Workbench Parameter: *FeedStore key id*

PROXY_URL

Required/Optional: Optional

Specifies a proxy server that the reader will be use when accessing a URL dataset. The port number of the proxy server can be set in the URL, or by using the **PROXY_PORT** directive.

Example:

```
GEORSS_PROXY_URL www.someproxy.net
GEORSS_PROXY_URL www.someproxy.net:8080
```

Workbench Parameter: *Http Proxy URL*

PROXY_PORT

Required/Optional: Optional

Specifies the port number of the proxy server indicated by the **PROXY_URL** directive. This directive should only be used if the port number was not indicated in the **PROXY_URL** directive. This directive is ignored if the **PROXY_URL** directive has no value.

Example:

```
GEORSS_PROXY_PORT 8080
```

Workbench Parameter: *Http Proxy Port*

PROXY_USERNAME

Required/Optional: Optional

Specifies the username to use when accessing a password protected proxy server. This directive is ignored if any of the **PROXY_URL**, **PROXY_PASSWORD** or **PROXY_AUTH_METHOD** directives have no value.

Example:

```
GEORSS_PROXY_USERNAME someusername
```

Workbench Parameter: *Proxy Username*

PROXY_PASSWORD

Required/Optional: Optional

Specifies the password to use when accessing a password protected proxy server. This directive is ignored if any of the **PROXY_URL**, **PROXY_USERNAME** or **PROXY_AUTH_METHOD** directives have no value.

Example:

```
GEORSS_PROXY_PASSWORD password1234
```

Workbench Parameter: *Http Proxy Password*

PROXY_AUTH_METHOD

Required/Optional: Optional

Specifies the authentication method to use when accessing a password protected proxy server. This directive is ignored if any of the **PROXY_URL**, **PROXY_USERNAME** or **PROXY_PASSWORD** directives have no value. Acceptable values for this directive are 'Basic' or 'Digest'.

Example:

```
GEORSS_PROXY_AUTH_METHOD Basic
```

Workbench Parameter: *Http Proxy Authentication Method*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

Writer Overview

The GeoRSS writer can write any collection of features out as a GeoRSS feed. If no feature is specified to be a Feed type feature, then the writer will use default values for the metadata it produces. Any feature whose type is not 'Feed' will be treated as an 'Entry' type feature.

This means that the writer will look at features for the attributes specified by the GeoRSS schema. Any feature attributes which are not contained in the GeoRSS feature schema will be ignored. Furthermore, if a feature has no value for certain attributes, the GeoRSS writer will provide default values for these attributes. This ensures that the GeoRSS writer always tries to produce a valid Atom or RSS feed, regardless of the features that are passed to it.

Coordinate Systems

The W3C Geo and GeoRSS Simple geometry formats use the EPSG:4326 coordinate system. Thus if the GeoRSS writer is using either of these geometry formats, all features passed to the writer will be reprojected to EPSG:4326 if this functionality is licensed. Features with no coordinate system are assumed to be in EPSG:4326.

The GML geometry format supports any coordinate system, so if the GeoRSS writer is using this geometry format, features passed to the writer will be written in whichever coordinate system they have been tagged with. In the event that no coordinate system is set on a feature, the coordinate system will be assumed to be EPSG:4326.

Geometry

The three different geometry formats support varying levels of geometry complexity. The GeoRSS writer will attempt to downgrade unsupported geometry to a supported type, but this is not always possible. Since the GeoRSS specification does not allow multi-geometries, the GeoRSS writer will always only attempt to write the first item of an aggregate of multi-geometry.

The W3C Geo geometry format only supports point geometry. If a feature with any other type of geometry is passed to the GeoRSS writer while it is writing in this format, the feature's geometry will be ignored.

The GeoRSS Simple geometry format supports point, line, polygon geometry. When writing in the format, the GeoRSS writer will attempt to downgrade more complex geometries to one of these types. For example, a feature with donut geometry will have its geometry written out as a polygon, and the interior of the donut will be ignored. Similarly, areas and ellipses will also be downgraded to polygon geometry. Arcs, paths and curves will be downgraded to line geometry.

The GML geometry format supports similar geometries to the ones supported by the GeoRSS simple geometry format. However the GML format allows donut geometry, so features with donut geometry will not be written out as polygon geometry.

Writer Directives

The suffixes shown are prefixed by the current `<WriterKeyword>` in a mapping file. By default, the `<WriterKeyword>` for the GeoRSS writer is `GEORSS`.

DATASET

Required/Optional: Required

The file to which the GeoRSS writer should output the XML feed. If the file does not exist it will be created.

Example:

```
GEORSS_DATASET c:\georss\feed.xml
```

Workbench Parameter: *Destination GeoRSS File*

WRITER_CHARSET

Required/Optional: Optional

The character set encoding in which the output XML feed should be written. If no character set is specified, the feed will be written in the UTF-8 character set. If an invalid character set is specified, the translation will fail.

Example:

`GEORSS_WRITER_CHARSET UTF-16`

Workbench Parameter: *Output Character Set*

OUTPUT_FORMAT

Required/Optional: Required

This directive specifies the format of the XML feed that the GeoRSS writer will produce. Acceptable values for this directive are 'Atom' and 'RSS'. If the value of the directive is Atom, the writer will produce an Atom 1.0 feed. If the value is RSS, the writer will produce an RSS 2.0 feed.

Example:

`GEORSS_OUTPUT_FORMAT Atom`

Workbench Parameter: *Output Format*

GEOMETRY_OUTPUT_FORMAT

Required/Optional: Required

This directive specifies the format of the output feed's geometry extensions. Acceptable values for this directive are 'W3C', 'Simple' or 'GML'.

Example:

`GEORSS_GEOMETRY_OUTPUT_FORMAT GML`

Workbench Parameter: *Geometry Format*

ESCAPE_HTML

Required/Optional: Optional

This directive determines how the writer handles HTML content. If the directive is set to 'true' or 'yes', the writer will escape html content before outputting it. If the directive is set to 'no' or 'false' the writer will output the content unchanged. This only applies to feature attributes whose corresponding 'type' attribute is set to 'html'.

Note that if the directive is set to 'no' or 'false' and unescaped HTML content is passed to the writer, the output may not be a valid XML. If no value is provided for this directive, the writer will assume any HTML content is already escaped, and will not escape it.

Example:

`GEORSS_ESCAPE_HTML yes`

Workbench Parameter: *Escape HTML Content*

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Geometry

The geometry of GeoRSS features may be identified by the `georss_type` attribute. The valid values for this attribute are:

<code>georss_type</code>	Description
<code>georss_no_geom</code>	FME Feature with no geometry.
<code>georss_point</code>	Point feature.

georss_type	Description
georss_line	Linear feature.
georss_area	Simple polygon or donut feature.
georss_box	Simple rectangular polygon feature.

No Geometry

georss_type: georss_no_geom

Features with their georss_type attribute set to georss_no_geom do not contain any geometry data.

Points

georss_type: georss_point

Features with their georss_type set to georss_point are single coordinate features.

Lines

georss_type: georss_line

Features with their georss_type set to georss_line are polyline features.

Areas

georss_type: georss_area

Features with their georss_type set to georss_area are polygon features which may or may not have interior boundaries.

Boxes

georss_type: georss_box

Features with their georss_type set to georss_box are simple rectangular closed polygons. Features with this type will not have any interior boundaries.

Schema

The GeoRSS reader returns features with the same schema for each of the different XML formats. The following tables explain which XML elements are mapped to which feature attributes. When a feature is passed to the GeoRSS writer, it will be treated as an 'Entry' type feature unless it's feature type is 'Feed'. This means that the GeoRSS writer will always treat features as if they were output by the GeoRSS reader. Any attributes that could not have been produced by the GeoRSS reader will be ignored. If a feature has no value for certain required attributes, the GeoRSS writer will provide a default value for the attribute.

Feed Type Features

Feature Attribute	Description/Source/Destination
Title	Atom: <title> RSS: <title> The GeoRSS writer will provide a default value for this attribute if a feature has no value.
TitleType	Atom/RSS: type="xxx" attribute inside the source XML element of the Title attribute. The default value is "text". This attribute will only be populated if the Title attribute is populated.
Version	This attribute is a string representation of the format of the XML feed. When reading an Atom feed, it is based on the XML namespace of the feed, while in RSS feeds it is based on the version="xxx" attribute of the <rss> element. This attribute is ignored by the GeoRSS writer.
Description	Atom 0.3: <tagline> Atom 1.0: <subtitle> RSS: <description>
DescriptionType	Atom/RSS: type="xxx" attribute inside the source XML tag of the Description attribute. The default value is "text". This attribute will only be populated if the Description attribute is populated.
Id	Atom: <id> The GeoRSS writer will provide a default value for this attribute if a feature has no value.
Copyright	Atom 0.3: <copyright> Atom 1.0: <rights> RSS: <copyright>
CopyrightType	Atom/RSS: type="xxx" attribute inside the source XML tag of the Copyright attribute. The default value is "text". This attribute will only be populated if the Copyright attribute is populated.
PublishedDate	RSS: <pubDate> The GeoRSS writer will provide a default value for this attribute if a feature has no value.
LastUpdate	Atom 0.3: <modified> Atom 1.0: <updated>

Feature Attribute	Description/Source/Destination
	RSS: <lastBuildDate> The GeoRSS writer will provide a default value for this attribute if a feature has no value.
Category	Atom 1.0: term="xxx" attribute inside <category> RSS: <category>
CategoryDomain	Atom 1.0: scheme="xxx" attribute inside <category> RSS: domain="xxx" attribute inside <category>
CategoryLabel	Atom 1.0: label="xxx" attribute inside <category>
Author	Atom: <name> inside of <author>
AuthorEmail	Atom: <email> inside of <author>
AuthorURI	Atom: <uri> inside of <author>
Contributor	Atom: <name> inside of <contributor>
ContributorEmail	Atom: <email> inside of <contributor>
ContributorURI	Atom: <uri> inside of <contributor>
Generator	Atom: <generator> RSS: <generator> The GeoRSS writer will provide a default value for this attribute if a feature has no value.
GeneratorURI	Atom 0.3: url="xxx" attribute inside <generator> Atom 1.0: uri="xxx" attribute inside <generator> The GeoRSS writer will provide a default value for this attribute if a feature has no value.
GeneratorVersion	Atom: version="xxx" attribute inside <generator> The GeoRSS writer will provide a default value for this attribute if a feature has no value.
Icon	Atom 1.0: <icon>
Logo	Atom 1.0: <logo>
LinkRelation	Atom: rel="xxx" attribute inside <link>
LinkURI	Atom: href="xxx" attribute inside <link> RSS: <link>
LinkType	Atom: type="xxx" attribute inside <link>
LinkTitle	Atom: title="xxx" attribute inside <link>

Feature Attribute	Description/Source/Destination
FeatureType	The first 'featuretypetag' attribute that is encountered on an element in the GeoRSS xml namespace.
Relationship	The first 'relationshiptag' attribute that is encountered on an element in the GeoRSS xml namespace.
Elevation	The first 'elev' attribute that is encountered on an element in the GeoRSS xml namespace.
FloorNumber	The first 'floor' attribute that is encountered on an element in the GeoRSS xml namespace.
Radius	The first 'radius' attribute that is encountered on an element in the GeoRSS xml namespace.

Entry Type Features

Feature Attribute	Source
Title	Atom: <title> RSS: <title> The GeoRSS writer will provide a default value for this attribute if a feature has no value.
TitleType	Atom/RSS: type="xxx" attribute inside the source XML tag of the Title attribute. The default value is "text". This attribute will only be populated if the Title attribute is populated.
Summary	Atom: <summary>
SummaryType	Atom/RSS: type="xxx" attribute inside the source XML tag of the Summary attribute. The default value is "text". This attribute will only be populated if the Summary attribute is populated.
Id	Atom: <id> RSS: <guid> The GeoRSS writer will provide a default value for this attribute if a feature has no value.
IdIsPermaLink	RSS: isPermaLink="xxx" attribute inside <guid>
Copyright	Atom 1.0: <rights>
CopyrightType	Atom/RSS: type="xxx" attribute inside the source XML tag of the Copyright attribute. The default value is "text". This attribute will only be populated if the Copyright attribute is populated.
PublishedDate	Atom 0.3: <issued> Atom 1.0: <published> RSS: <pubDate> The GeoRSS writer will provide a default value for this attribute if a feature has no value.
LastUpdate	Atom 0.3: <modified> Atom 1.0: <updated> The GeoRSS writer will provide a default value for this attribute if a feature has no value.
Category	Atom 1.0: term="xxx" attribute inside <category> RSS: <category>
CategoryDomain	Atom 1.0: scheme="xxx" attribute inside <category> RSS: domain="xxx" attribute inside <category>

Feature Attribute	Source
CategoryLabel	Atom 1.0: label="xxx" attribute inside <category>
Author	Atom: <name> inside of <author> RSS: <author>
AuthorEmail	Atom: <email> inside of <author>
AuthorURI	Atom: <uri> inside of <author>
Contributor	Atom: <name> inside of <contributor>
ContributorEmail	Atom: <email> inside of <contributor>
ContributorURI	Atom: <uri> inside of <contributor>
Content	Atom: <content> RSS: <description>
ContentType	Atom/RSS: type="xxx" attribute inside the source XML tag of the Content attribute. The default value is "text". This attribute will only be populated if the Content attribute is populated.
ContentIsRemoteURL	Atom: This attribute will be set to 'Yes' if the value of the Content attribute came from the src="xxx" attribute inside the <content> element.
LinkRelation	Atom: rel="xxx" attribute inside <link> (unless rel="enclosure")
LinkURI	Atom: href="xxx" attribute inside <link> (unless rel="enclosure") RSS: <link>
LinkType	Atom: type="xxx" attribute inside <link> (unless rel="enclosure")
LinkTitle	Atom: title="xxx" attribute inside <link> (unless rel="enclosure")
EnclosureTitle	Atom: title="xxx" attribute inside <link> where rel="enclosure"
EnclosureURI	Atom: href="xxx" attribute inside <link> where rel="enclosure" RSS: url="xxx" attribute inside <enclosure>
EnclosureType	Atom: type="xxx" attribute inside <link> where rel="enclosure" RSS: type="xxx" attribute inside <enclosure>

Feature Attribute	Source
EnclosureLength	RSS: length="xxx" attribute inside <enclosure>
FeatureType	The first 'featuretypetag' attribute that is encountered on an element in the GeoRSS xml namespace.
Relationship	The first 'relationshiptag' attribute that is encountered on an element in the GeoRSS xml namespace.
Elevation	The first 'elev' attribute that is encountered on an element in the GeoRSS xml namespace.
FloorNumber	The first 'floor' attribute that is encountered on an element in the GeoRSS xml namespace.
Radius	The first 'radius' attribute that is encountered on an element in the GeoRSS xml namespace.

GML (Geography Markup Language) Reader/Writer

The GML Reader/Writer allows FME to read and write files in the Geography Markup Language (GML) format.

This chapter assumes familiarity with GML.

Overview

GML is an OpenGIS® Implementation Specification. The GML specification defines an XML encoding for the transport and storage of geographic information. This specification can be found at the Open GIS Consortium website www.opengeospatial.org.

GML documents must be instances of a conforming application schema. Conforming application schemas are to be defined with the W3C's XML Schema language.

Versions

The reader supports reading GML v2.1.2, v3.1.1, and v3.2.1 files.

The writer supports writing GML v3.1.1 and v3.2.1 files.

Additional GML Formats

Note that the following formats are GML formats, and their documentation is considered part of the GML Reader/Writer documentation:

Format	Reader/Writer
German AAA GML Exchange format (NAS); ALKIS	Reader
Interface 2000 (GML)	Both
NEN 3610	Reader
U.S. Census Bureau TIGER/GML	Reader

GML Quick Facts

Format Type Identifier	GML
Reader/Writer	Both
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	Varies depending on the GML application schema
Typical File Extensions	.gml, .xml
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	xml_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	no		polygon	yes
circular arc	yes		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	yes		z values	yes
none	yes			

Reader Overview

This reader supports GML documents conforming to GML v2.1.2, v3.1.1 and v3.2.1, and application schemas.

Only simple GML geometries, i.e., geometries with linear interpolation, are supported in this release.

Multi-value properties, i.e., declared with a **maxOccurs** that is greater than 1 or unbounded, are supported and are mapped into list attributes.

GML properties that are defined as complex types are supported – these complex properties are mapped as structured list attributes.

This reader supports multiple geometry properties per feature type: see the `MAP_GEOMETRY_COLUMNS` reader directive.

Reader Directives

The suffixes listed are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the GML reader is `GML`.

DATASET

Required/Optional: *Required*

This directive specifies the location for the input GML instance document.

Example:

```
GML_DATASET c:\gml_data\hydro.xml
```

Workbench Parameter: *Source Geography Markup Language (GML) File(s)*

SYSTEM_ENCODING

Required/Optional: *Optional*

Specifies the encoding to use for the GML schema and data features that are read by the reader. If not set, then features will be output in the system's encoding.

Example:

```
GML_SYSTEM_ENCODING UTF-8
```

Workbench Parameter: *System Encoding*

XSD_DOC

Required/Optional: *Optional*

A GML instance document specifies the namespace and the location of its application schema through its root element `xsi:schemaLocation` attribute. This directive allows the GML reader to use a different GML schema document from the one specified in the `xsi:schemaLocation` attribute.

The XML Schema specification states that the `xsi:schemaLocation` attribute value consists of a set of pairs: The first member each pair is the namespace for which the second member is the hint describing where to find an appropriate schema document. The presence of this hint does not require the processor to obtain or use the cited schema document, however, the processor is free to use other schemas obtained by other suitable means.

The `XSD_DOC` directive allows the usage of other schema documents on the instance besides the one stated in the instance's `xsi:schemaLocation` attribute.

This directive only takes effect if the target namespace of the dataset is not in the Safe fixed schema namespace `http://www.safe.com/xml/schemas/FMEFeatures`. The GML2 writer in `FIXED_SCHEMA_MODE` writes out documents that belong to that namespace.

Workbench Parameter: *Application Schema*

READ_PREDEFINED_GML_PROPERTIES

Required/Optional: *Optional*

This directive specifies if the default and optional GML feature properties, name and description, should be read. The valid values of this directive are YES and NO, its default value is NO.

Example:

```
GML_READ_PREDEFINED_GML_PROPERTIES YES
```

Workbench Parameter: *Read Predefined Properties*

CONTINUE_ON_GEOM_ERROR

Required/Optional: *Optional*

Rather than halting the reader, this optional directive allows the reader to continue reading and extracting features from the input GML document stream upon encountering a geometrical error. The valid values of this directive are YES and NO, its default value is YES.

Example:

```
GML_CONTINUE_ON_GEOM_ERROR NO
```

Workbench Parameter: *Continue on Geometry Error*

HTTP_PROXY

Required/Optional: *Optional*

This directive specifies the HTTP proxy to be used for network fetches. The port number may be specified at the end of the proxy by appending `:[port number]` or through the `HTTP_PROXY_PORT` directive.

Example:

```
GML_HTTP_PROXY www.someproxy.net
```

or

```
GML_HTTP_PROXY www.someproxy.net:8082
```

Note: Users may bypass the `HTTP_PROXY` and `HTTP_PROXY_PORT` directives and still have http proxy support by specifying the `http_proxy` environment variable. The value for this environment variable should be of the form `[protocol:][user:password@]machine[:port]`, where components within `[]` are optional. An example value for the `http_proxy` environment variable is: `http://www.someproxy.net:8885`.

Workbench Parameter: *Proxy Address*

HTTP_PROXY_PORT

Required/Optional: *Optional*

This directive is used if the HTTP proxy port was not specified in the `HTTP_PROXY` directive.

Example:

```
GML_HTTP_PROXY_PORT 8081
```

Workbench Parameter: *Port*

CACHE_XSD

Required/Optional: *Optional*

This directive allows the XML Schema documents that are fetched from the internet to be cached locally, this reduces the number of network fetches when traversing the GML schema documents. The valid values of this directive are YES and NO, its default value is YES.

Example:

```
GML_CACHE_XSD NO
```

Workbench Parameter: *Cache XSD Document*

CACHE_XSD_EXPIRY_TIME

Required/Optional: *Optional*

This directive is optional and takes effect only if the `CACHE_XSD` directive is set to `YES`. The valid values for this directive are positive numbers denoting the number of seconds. The default value for this directive is 300.

Example:

```
GML_CACHE_XSD_EXPIRY_TIME 600
```

Workbench Parameter: *Cache XSD Expiry Time*

CACHE_XSD_DIRECTORY

Required/Optional: *Optional*

This optional directive takes effect when `CACHE_XSD` directive is set to `YES`. The directive specifies the directory path for the location of the cache xsd directory, the directory name for the cache xsd directory is specified by the `CACHE_XSD_NAME` directive below. The default value for this directive is the user's temporary directory.

Example:

```
GML_CACHE_XSD_DIRECTORY c:\tmp
```

Workbench Parameter: *<WorkbenchParameter>*

CACHE_XSD_NAME

Required/Optional: *Optional*

This optional directive specifies the xsd cache name. The cache name must also be a valid directory name, as this value is used as the sub-directory containing the cached schema documents within the `CACHE_XSD_DIRECTORY`. The default value for this directive is `GML_XSD_CACHE`.

Example:

```
GML_CACHE_XSD_NAME gml_schema_cache
```

Workbench Parameter: *<WorkbenchParameter>*

XFMAP

Required/Optional: *Optional*

This optional directive is not for general usage.

Rather than having the reader programmatically generate an xfMap from the GML application schema, this directive directs the reader to use a predefined xfMap on the input GML dataset.

Multiple xfMaps on the same input stream may be used. These can either be specified by using several `XFMAP` directives or through a single quoted value `XFMAP` directive where each xfMap in the quoted string is separated by a semicolon.

Example:

```
GML_XFMAP C:\tmp\data\features.xmp
```

or

```
GML_XFMAP "C:\tmp\drainages.xmp;C:\tmp\pits_pipes.xmp"
```

or

```
GML_XFMAP C:\tmp\drainages.xmp  
GML_XFMAP C:\tmp\pits_pipes.xmp
```

Workbench Parameter: *<WorkbenchParameter>*

XFMAP_SCHEMA

Required/Optional: *Optional*

This optional directive is not for general usage. When reading schemas, the GML reader constructs FME schema features after examining the GML application schema corresponding to the input dataset, this directive allows the GML reader to bypass the GML application schema by constructing FME schema features with a predefined xfMap.

Example:

```
GML_XFMAP_SCHEMA C:\tmp\data\schemaschema_features.xmp
```

SRS_AXIS_ORDER

This optional directive overrides the axis order when reading <TocEntry Title="SRS_AXIS_ORDER" Link="/Content/gml/SRS_AXIS_ORDER.htm" /> a coordinate tuple in a GML <pos> or <posList> element.

Values

The valid values for this directive are "1,2", "2,1", "1,2,3" and "2,1,3". There is no default value.

For example, if the srsName in the GML document is set to "urn:ogc:def:crs:EPSG:6.6.4326", and the user is sure that the coordinate order in the GML document is lon-lat and not lat-lon order, then this directive should be set to "1,2" so that the reader reads the data in lon-lat order.

Mapping File Syntax

```
GML_SRS_AXIS_ORDER 1,2
```

Required/Optional

Optional

*** Workbench Parameter**

GML SRS Axis Order

COMPLEX_PROPERTIES_AS_NESTED_LISTS

This directive specifies whether GML properties that are defined as a complex type with complex content (that is, those that have embedded children elements) should be mapped as nested list attributes within FME features.

Some complex properties, such as those that are recursively defined, cannot be mapped as nested lists. These complex properties will always be mapped as XML fragments, regardless of the value of this directive.

Values

YES (default) | NO

If the value is set to NO, the complex properties are mapped as XML fragments.

Required/Optional

Optional

Mapping File Syntax

```
GML_COMPLEX_PROPERTIES_AS_NESTED_LISTS NO
```

*** Workbench Parameter**

Complex Properties as Nested Lists

XML_FRAGMENTS_AS_DOCUMENTS

This directive specifies whether GML properties that are mapped as XML fragments should be converted into XML documents.

The conversion will add missing namespace declarations to the fragments, it will maintain CDATA sections, and it will also prefix an XML header declaration to the fragment. Converting the XML fragments into XML documents allows XML-based parsers, e.g., XSLT and XQuery based processors, to further process the fragments.

Values

YES (default) | NO

Required/Optional

Optional

Mapping File Syntax

```
GML_XML_FRAGMENTS_AS_DOCUMENTS NO
```

✳ Workbench Parameter

Map XML Fragments as XML Documents

MAP_GEOMETRY_COLUMNS

This directive specifies whether the GML geometric properties should be represented as individual, and possibly multiple, geometry columns in FME feature type definitions.

A geometric column in an FME data feature is represented either as a single named geometry, or, if multiple geometry columns are present, as an aggregate geometry with multiple named geometry components, this aggregate geometry will also have its "Contains Individual Geometries" interpretation flag set.

A new attribute type has also been introduced for specifying the order and/or position of a geometric column in the feature type definition. If an attribute X has its type set to "xml_geometry" then this attribute X becomes a placeholder in the feature type definition. It is a placeholder because actual data features for the feature type definitions will not have this attribute; instead, the data features will have a geometry named "X".

Values

YES (default) | NO

Required/Optional

Optional

Mapping File Syntax

```
GML_MAP_GEOMETRY_COLUMNS NO
```

✳ Workbench Parameter

Map Geometry Columns

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

This writer outputs GML documents conforming to GML v3.2.1 and v3.1.1. Two XML documents are written, a GML instance and its corresponding GML application schema. The GML application schema structure is controlled by the GML writer's DEF lines.

Currently, only simple GML geometries, i.e., geometries with linear interpolation, are supported in this release.

Multiple geometrical property elements can be written per feature type definition. The name and the position for these geometric elements can also be controlled through the GML writer's feature type definitions.

Complex non-geometrical properties are supported in this release. These are to be specified in the GML Writer DEF lines as FME structured attributes. Each component in a structured attribute is separated by a period, ".", and each component may also be a list, specified by the "{}" token. Attributes that are lists will be mapped into GML property declarations having a **maxOccurs** of **unbounded**. For example, the structured FME attributes, "changeHistory.Date", and "changeHistory.Reason" are represented in the GML instance as: "<changeHistory><Date>...</Date><Reason>...</Reason></changeHistory>".

Writer Directives

The directives processed by the GML Writer are listed below. The suffixes shown are prefixed by the current <writerKeyword> in a mapping file. By default, the <writerKeyword> for the GML writer is **GML**.

DATASET

Required/Optional: *Required*

This directive specifies the location for the output GML instance document.

Example:

```
GML_DATASET c:\gml\data.xml
```

Workbench Parameter: *Destination Geography Markup Language (GML) File*

PROFILE

This directive sets the writer into GML v3.1.1 or GML v3.2.1 writing.

Values

GENERIC 3.1.1 (default) | GENERIC 3.2.1

Mapping File Syntax

```
GML_PROFILE GENERIC_3.2.1
```

Required/Optional

Optional

* Workbench Parameter

GML Profile

WRITE_ALL_FEATURE_TYPE_DEFNS

Required/Optional: *Optional*

By default, every GML writer DEF line is translated to the output GML application schema document as an XML Schema element declaration and a corresponding complex type definition. Setting this directive to NO directs the writer to ignore those DEF lines that have no corresponding feature instances. The valid values for this directive are YES and NO. The default is YES.

Example:

```
GML_WRITE_ALL_FEATURE_TYPE_DEFNS NO
```

Workbench Parameter: *Write All Feature Type Definitions*

TARGET_NS_PREFIX

Required/Optional: *Optional*

The directive specifies the GML application schema target namespace prefix. The default value for this directive is fme.

Example:

```
GML_TARGET_NS_PREFIX ns
```

Workbench Parameter: *Target Namespace Prefix*

TARGET_NS_URI

Required/Optional: *Optional*

The directive specifies the GML application schema target namespace URI. The default value for this directive is <http://www.safe.com/gml/fme>.

Example:

```
GML_TARGET_NS_URI http://www.contrivedurl.com/ns
```

Workbench Parameter: *Target Namespace URI*

XSD_DOC

Required/Optional: *Optional*

The file location for the output GML application document. If left blank, then the **.xsd** file will be created in the same directory and with the same file basename as the output dataset.

Workbench Parameter: *GML Schema Document*

OUTPUT_ENCODING

Required/Optional: *Optional*

The directive specifies the encoding to use for the output XML documents, which include both the GML instance and the GML application schema. The default value for this directive is UTF-8.

Example:

```
GML_OUTPUT_ENCODING ISO-8891-1
```

Workbench Parameter: *Output Encoding*

SYSTEM_ENCODING

Required/Optional: *Optional*

This directive specifies the encoding for the incoming schema and data feature. If unspecified, then the writer assumes that the schema and data features are encoded with the system's encoding.

Example:

For example, if features fed into the GML writer are encoded in UTF-8, then the following should be set:

```
GML_SYSTEM_ENCODING UTF-8
```

Workbench Parameter: *System encoding*

SUPPRESS_XSD_DOCUMENT

Required/Optional: *Optional*

This directive suppresses the output of the GML application schema when it is set to YES. The default value for this directive is NO.

Example:

```
GML_SUPPRESS_XSD_DOCUMENT YES
```

Workbench Parameter: *Suppress GML Schema Document*

SUPPRESS_XSI_SCHEMALOCATION

Required/Optional: *Optional*

Setting this directive to YES suppresses the output of the xsi:schemaLocation attribute in the GML instance's root element. The xsi:schemaLocation in an XML document instance is not a mandatory attribute – it is merely a hint which an XML processor may choose to ignore. Setting this keyword to YES suppresses the output of the **xsi:schemaLocation** attribute in the output GML instance. The default value for this directive is NO.

Example:

```
GML_SUPPRESS_XSI_SCHEMALOCATION YES
```

Workbench Parameter: *Suppress xsi:schemaLocation attribute*

FEATURE_COLLECTION

The target namespace of the root FeatureCollection element is influenced by this directive.

Values

gml (default) | target-namespace | wfs

- **gml** – FeatureCollection root element for the output GML instance is the predefined <gml:FeatureCollection> element.
- **wfs** – changes the root element in the output document to <wfs:FeatureCollection>; in addition, appropriate XML namespace declarations and xsi:schemaLocation entries are added to include the WFS schemas.

The wfs value is not currently supported when writing GML v3.2.1 data. This is because the WFS 1.1.0 FeatureCollection is defined for GML 3.1.1. Wrapping the data with a WFS FeatureCollection will be enabled for GML v3.2.1 writing when WFS 2.0 is supported.

- **target-namespace** – changes the root element to <[target-ns-prefix]:FeatureCollection> where [target-ns-prefix] is the prefix that is bound to the output document target namespace, e.g., <fme:FeatureCollection>. The output xsd document will also include a custom FeatureCollection declaration and FeatureCollectionType definition.

Mapping File Syntax

```
GML_FEATURE_COLLECTION target-namespace
```

Required/Optional

Optional

* Workbench Parameter

Feature Collection

FEATURE_COLLECTION_ID

GML v3.2.1 requires an gml:id on every feature and feature collection.

This directive allows you to supply a custom gml:id value to replace UUID that is automatically for the feature collection under GML v3.2.1.

Mapping File Syntax

```
GML_FEATURE_COLLECTION_ID my-id-001
```

Required/Optional

Optional

* Workbench Parameter

Feature Collection ID

TARGET_XSI_SCHEMALOCATION_URL

Required/Optional: *Optional*

By default, the URL for the target namespace URI - URL location pair in the instance document's xsi:schemaLocation attribute is the GML application schema's relative filename. This directive allows the user to overwrite the .xsd filename by supplying a custom URL. The directive only affects the value of the URL for the target namespace in the xsi:schemaLocation. It does not affect where the application schema will be written (for this, see the XSD_DOC directive).

Example:

```
GML_TARGET_XSI_SCHEMALOCATION_URL http://www.conurl/ns.xsd
```

Workbench Parameter: *Target xsi:schemaLocation URL*

SRS_NAME

Required/Optional: *Optional*

This directive allows the user to overwrite the CRS value that is written in the GML instance's srsName attributes. By default, the FME coordinate system name in a feature is transferred directly onto the srsName attribute. This directive allows users to provide their own URN CRS string for the srsName attributes.

Example:

```
GML_SRS_NAME urn:ogc:def:crs:EPSG:6.6:4326
```

Workbench Parameter: *GML srsName*

SRS_AXIS_ORDER

Required/Optional: *Optional*

This directive is required when the **SRS_NAME** directive is used. It specifies the axis order for a coordinate tuple in a GML `<pos>` or `<posList>` element. The valid values for this directive are "1,2", "2,1", "1,2,3" and "2,1,3". There is no default value.

For example, if **SRS_NAME** is set to "urn:ogc:def:crs:EPSG:6.6.4326", then the **SRS_AXIS_ORDER** should be set to "2,1" so that the coordinates in the GML `<pos>` and `<posList>` elements are written in lat-lon order:

```
GML_SRS_AXIS_ORDER 2,1
```

Workbench Parameter: *GML SRS Axis Order*

MAXIMUM_FRACTION_DIGITS

Required/Optional: *Optional*

This directive allows the user to specify the maximum number of decimal place to write to the GML file when writing coordinates. Note that this directive specifies only the maximum number of decimal places to write, not the *exact* number of decimal places to write. To specify an exact number of decimal places to write, use this directive in conjunction with the **MINIMUM_FRACTION_DIGITS** directive. The value for this directive must be an integer greater than or equal to 0. The default value is 15 decimal places.

Workbench Parameter: *Maximum Decimal Places*

MINIMUM_FRACTION_DIGITS

Required/Optional: *Optional*

This directive allows the user to specify the minimum number of decimal place write to the GML file when writing coordinates. Note that this directive specifies only the minimum number of decimal places to write, not the *exact* number of decimal places to write. To specify an exact number of decimal places to write, use this directive in conjunction with the **MAXIMUM_FRACTION_DIGITS** directive. The value for this directive must be an integer greater or equal to 0. The default value is 0 decimal places.

Workbench Parameter: *Minimum Decimal Places*

APPLY_STYLESHEET

Required/Optional: *Optional*

This directive allows an XSLT stylesheet to be applied to the final output DATASET document. The **STYLESHEET_RESULT** directive may be used in conjunction with this directive to specify the location and filename of the resulting transformation. There are no default values for this directive.

Example:

```
GML_APPLY_STYLESHEET c:\data\myTransform.xsl
```

Workbench Parameter: *XSLT Style Sheet to Apply*

STYLESHEET_RESULT

Required/Optional: *Optional*

This directive only takes effect if **APPLY_STYLESHEET** is specified. When this directive is not present or its value is the empty string, then the resulting XSLT transformation will have the same location and filename as the output DATASET with the exception that the filename will be prefixed with "transformed_".

Example:

```
GML_STYLESHEET_RESULT c:\data\myTransformedDoc.xml
```

Workbench Parameter: *Style Sheet Result*

USE_STYLESHEET_RESULT_AS_DATASET

This directive only takes effect if **APPLY_STYLESHEET** is specified.

This directive takes precedence over **STYLESHEET_RESULT**. It also suppresses the XML schema .xsd document for output.

Values

YES | NO (default)

When set to YES, a temporary file is written in the same directory as the one specified for the DATASET, the stylesheet is applied to the temporary file, and the result of the transformation is stored in the DATASET file.

Mapping File Syntax

`GML_USE_STYLESHEET_RESULT_AS_DATASET` Yes

Required/Optional

Optional

* Workbench Parameter

Dataset as stylesheet result

VALIDATE_OUTPUT_DATASET

This directive controls validation of the output file against the schemas referenced in that file.

If APPLY_STYLESHEET is specified, the result of applying the stylesheet is validated. Otherwise, the DATASET is validated.

Values

YES | NO (default)

Mapping File Syntax

`GML_VALIDATE_OUTPUT_DATASET` Yes

Required/Optional

Required

* Workbench Parameter

Validate GML Dataset File

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

The `xml_type` attribute identifies the FME GML geometry. The valid values for this attribute are:

<code>xml_type</code>	Description
<code>xml_no_geom</code>	FME Feature with no geometry.
<code>xml_point</code>	Point feature.
<code>xml_line</code>	Linear feature.
<code>xml_arc</code>	Arc feature.
<code>xml_area</code>	Areal feature, may be a donut.

Other attributes, including the feature's feature type, are dependent on the GML application schema.

It is possible to control the name and position for the GML geometry properties for a GML feature type, see DEF Lines for details.

No Geometry

xml_type: xml_no_geom

Features having their **xml_type** attribute set to **xml_no_geom** do not contain any geometry data.

Points

xml_type: xml_point

Features having their **xml_type** set to **xml_point** are single coordinate features.

Lines

xml_type: xml_line

Features having their **xml_type** set to **xml_line** are polyline features and have at least two coordinates.

Arcs

xml_type: xml_arc

Features having their **xml_type** set to **xml_arc** are circular arc segments.

Areas

xml_type: xml_area

Features having their **xml_type** set to **xml_area** are either a single closed polyline feature (simple closed polygon), a donut, or an aggregate of donuts (and/or simple polygons). A simple closed polygon contains at least four coordinates, with the first and last coordinate being equal.

DEF Lines

The DEF lines control the generation of the GML application schema. The syntax of a GML DEF line is:

```
<writerKeyword>_DEF <feature type>
    [<attribute name> <attribute type>]*
```

Where the valid values for <attribute type> are: xml_buffer, xml_char(width), xml_int32, xml_real32, xml_decimal(width,decimal), xml_boolean, xml_real64, and xml_geometry.

The <attribute name> can either be a simple attribute, a list attribute, a structured attribute, or a geometry attribute.

Simple attributes have no embedded period, and they are mapped into simple type properties in the GML application schema. List attributes end with a "{}" suffix, and are mapped into simple type property declarations whose maxOccurs are set to unbounded. Structured attributes have embedded periods, each component in a structured attribute is separated by a period, and each component may also be a list. Structured attributes are mapped into complex type properties in the GML application schema. For example, the structured FME attributes, "changeHistory.Date", and "changeHistory.Reason" are mapped into a changeHistory element whose complex type is defined as a sequence of Date and Reason elements both of which are defined as simple types.

Geometry attributes are used to control the name and the position of the GML geometry elements. Currently only the generic xml_geometry type is supported. Therefore, the geometry attribute only specifies the name and the position for the GML gml geometric property. The type for the GML geometric property is determined via the instances of the data features. For example, if all instances of a named geometry corresponding to the geometry attribute are points, then the XSD written will have its geometric element declared as **gml:PointPropertyType**; if the specific GML geometry property type cannot be determined via the data features, then the geometric element is declared as **gml:GeometryPropertyType**.

For example, the DEF line:

```
GML_DEF F1
```

\

GML SF-0 (Geography Markup Language Simple Features Level SF-0 Profile) Reader/Writer

The GMLSF Reader/Writer allows FME to read and write files in the GML Simple Features Profile format.

This chapter assumes that users are familiar with GML and the GML Simple Features Profile.

Overview

GML is a complex specification declaring a wide number of XML elements for a wide variety of capabilities, including the ability to encode dynamic features, spatial and temporal topology, coverages, and complex geometric types.

The GML Simple Features Profile restricts the wide-scope of GML by specifying a useful subset of GML, supporting features and a limited set of linearly interpolated geometric types, it also simplifies the processing of the XML-Schema by defining strict XML-Schema usage/coding patterns.

The GML Simple Features Profile specifies three compliance levels: compliance level SF-0, SF-1, and SF-2. Currently, only compliance level SF-0 is supported for writing.

GMLSF Quick Facts

Format Type Identifier	GMLSF
Reader/Writer	Both
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	Varies depending on the GML simple feature profile application schema
Typical File Extensions	.gml,.xml
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	xml_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	yes		z values	yes
none	yes			

Reader Overview

This reader supports GML documents conforming to the GML simple features profile.

Multi-value properties, i.e., declared with a **maxOccurs** that is greater than 1 or unbounded, are supported and are mapped into list attributes.

GML properties that are defined as complex types are supported – these complex properties are mapped as structured list attributes.

Reader Directives

The suffixes listed are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the GMLSF reader is **GMLSF**.

DATASET

Required/Optional: *Required*

This directive specifies the location for the input GML instance document.

Example:

```
GMLSF_DATASET c:\gml_data\hydro.xml
```

Workbench Parameter: *Source GML SF-0 File(s)*

SYSTEM_ENCODING

Required/Optional: *Optional*

Specifies the encoding to use for the GML schema and data features that are read by the reader. If not set, then features will be output in the system's encoding.

Example:

```
GMLSF_SYSTEM_ENCODING UTF-8
```

Workbench Parameter: *System encoding*

XSD_DOC

Required/Optional: *Optional*

A GML instance document specifies the namespace and the location of its application schema through its root element `xsi:schemaLocation` attribute. This directive allows the GML reader to use a different GML schema document from the one specified in the `xsi:schemaLocation` attribute.

Example:

GMLSF_XSD_DOC c:\data\gmlsf\myschema.xsd

Workbench Parameter: *Application Schema*

CONTINUE_ON_GEOM_ERROR

Required/Optional: *Optional*

Rather than halting the reader, this optional directive allows the reader to continue reading and extracting features from the input GML document stream upon encountering a geometrical error. The valid values of this directive are YES and NO, its default value is YES.

Example:

GMLSF_CONTINUE_ON_GEOM_ERROR NO

Workbench Parameter: *Continue on Geometry Error*

HTTP_PROXY

Required/Optional: *Optional*

This directive specifies the HTTP proxy to be used for network fetches. The port number may be specified at the end of the proxy by appending **:[port number]** or through the HTTP_PROXY_PORT directive.

Example:

GMLSF_HTTP_PROXY www.someproxy.net

or

GMLSF_HTTP_PROXY www.someproxy.net:8082

Note: Users may bypass the HTTP_PROXY and HTTP_PROXY directives and still have http proxy support by specifying the http_proxy environment variable. The value for this environment variable should be of the form [protocol://][user:password@]machine[:port], where components within [] are optional. An example value for the http_proxy environment variable is: http://www.someproxy.net:8885.

Workbench Parameter: *Proxy Address*

HTTP_PROXY_PORT

Required/Optional: *Optional*

This directive is used if the HTTP proxy port was not specified in the HTTP_PROXY directive.

Example:

GMLSF_HTTP_PROXY_PORT 8081

Workbench Parameter: *Port*

CACHE_XSD

Required/Optional: *Optional*

This directive allows the XML Schema documents that are fetched from the internet to be cached locally, this reduces the number of network fetches when traversing the GML schema documents. The valid values of this directive are YES and NO, its default value is YES.

Example:

GMLSF_CACHE_XSD NO

Workbench Parameter: *Cache XSD Documents*

CACHE_XSD_EXPIRY_TIME

Required/Optional: *Optional*

This directive is optional and takes effect only if the `CACHE_XSD` directive is set to YES. The valid values for this directive are positive numbers denoting the number of seconds. The default value for this directive is 300.

Example:

```
GMLSF_CACHE_XSD_EXPIRY_TIME 600
```

Workbench Parameter: *Cache XSD Expiry Time*

CACHE_XSD_DIRECTORY

Required/Optional: *Optional*

This optional directive takes effect when `CACHE_XSD` directive is set to YES. The directive specifies the directory path for the location of the cache xsd directory, the directory name for the cache xsd directory is specified by the `CACHE_XSD_NAME` directive below. The default value for this directive is the user's temporary directory.

Example:

```
GMLSF_CACHE_XSD_DIRECTORY c:\tmp
```

CACHE_XSD_NAME

Required/Optional: *Optional*

This optional directive specifies the xsd cache name. The cache name must also be a valid directory name, as this value is used as the sub-directory containing the cached schema documents within the `CACHE_XSD_DIRECTORY`. The default value for this directive is `GML_XSD_CACHE`.

Example:

```
GMLSF_CACHE_XSD_NAME gml_schema_cache
```

SRS_AXIS_ORDER

Required/Optional: *Optional*

This optional directive allows the user to override the axis order used by the reader when reading a GML document. The directive specifies axis order to use when reading a coordinate tuple in a GML `<pos>` or `<posList>` element. The valid values for this directive are "1,2", "2,1", "1,2,3" and "2,1,3". There is no default value.

For example, if the GMLSF reader is making a mistake interpreting the coordinates and the user knows that a particular GML document contains coordinate tuples in lat-lon-height order then this directive should be set to "2,1,3".

Mapping File Syntax

```
GML_SRS_AXIS_ORDER 2,1,3
```

Workbench Parameter: *GML SRS Axis Order*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

This writer outputs GML documents conforming to the GML simple features profile. Two XML documents are written: a GML instance and its corresponding GML application schema. The feature types in the GML simple features application schema are controlled by the GMLSf writer's DEF lines.

Only compliance level SF-0 is supported in this release.

Writer Directives

The directives processed by the GMLSf Writer are listed below. The suffixes shown are prefixed by the current `<writerKeyword>` in a mapping file. By default, the `<writerKeyword>` for the GMLSf writer is `GMLSf`.

DATASET

Required/Optional: *Required*

This directive specifies the location for the output GML instance document.

Example:

```
GMLSf_DATASET c:\gml\data.xml
```

Workbench Parameter: *Destination GML Simple Features Level SF-0 Profile File*

WRITE_ALL_FEATURE_TYPE_DEFNS

Required/Optional: *Optional*

By default, every GML writer DEF line is translated to the output GML application schema document as an XML Schema element declaration and a corresponding complex type definition. Setting this directive to NO directs the writer to ignore those DEF lines that have no corresponding feature instances. The valid values for this directive are YES and NO. The default is YES.

Example:

```
GMLSf_WRITE_ALL_FEATURE_TYPE_DEFNS NO
```

Workbench Parameter: *Write all feature type definitions*

TARGET_NS_PREFIX

Required/Optional: *Optional*

The directive specifies the GML application schema target namespace prefix. The default value for this directive is `fme`.

Example:

```
GMLSf_TARGET_NS_PREFIX ns
```

Workbench Parameter: *Target namespace prefix*

TARGET_NS_URI

Required/Optional: *Optional*

The directive specifies the GML application schema target namespace URI. The default value for this directive is `http://www.safe.com/gml/fme`.

Example:

```
GMLSf_TARGET_NS_URI http://www.contrivedurl.com/gmlsf
```

Workbench Parameter: *Target namespace URI*

XSD_DOC

Required/Optional: *Optional*

The file location for the output GML application document. If left blank, then the **.xsd** file will be created in the same directory and with the same file basename as the output dataset.

Workbench Parameter: *GML schema document*

OUTPUT_ENCODING

Required/Optional: *Optional*

The directive specifies the encoding to use for the output XML documents, which include both the GML instance and the GML application schema. The default value for this directive is **UTF-8**.

Example:

```
GML_OUTPUT_ENCODING ISO-8891-1
```

Workbench Parameter: *Output encoding*

SYSTEM_ENCODING

Required/Optional: *Optional*

This directive specifies the encoding for the incoming schema and data feature. If unspecified, then the writer assumes that the schema and data features are encoded with the system's encoding.

Example:

For example, if features fed into the GMLSF writer are encoded in UTF-8, then the following should be set:

```
GMLSF_SYSTEM_ENCODING UTF-8
```

Workbench Parameter: *System encoding*

SUPPRESS_XSD_DOCUMENT

Required/Optional: *Optional*

This directive suppresses the output of the GML application schema when it is set to YES. The default value for this directive is NO.

Example:

```
GMLSF_SUPPRESS_XSD_DOCUMENT YES
```

Workbench Parameter: *Suppress GML schema document*

FEATURE_COLLECTION

Required/Optional: *Optional*

The valid values for this optional directive are `target-namespace` and `wfs`. Setting this directive to `wfs` changes the root element in the output document to `<wfs:FeatureCollection>`. In addition, appropriate XML namespace declarations and `xsi:schemaLocation` entries are added to include the WFS schemas. The default value for this directive is `target-namespace`.

Example:

```
GML_FEATURE_COLLECTION wfs
```

Workbench Parameter: *Feature Collection*

TARGET_XSI_SCHEMALOCATION_URL

Required/Optional: *Optional*

By default, the URL for the target namespace URI - URL location pair in the instance document's `xsi:schemaLocation` attribute is the GML application schema's relative filename. This directive allows the user to overwrite the `.xsd` filename by supplying a custom URL. The directive only affects the value of the URL for the target namespace in the `xsi:schemaLocation`. It does not affect where the application schema will be written (for this, see the `XSD_DOC` directive).

Example:

```
GMLSf_TARGET_XSI_SCHEMALOCATION_URL http://www.conur1/ns.xsd
```

Workbench Parameter: *Target xsi:schemaLocation URL*

COMPLIANCE_LEVEL_SCHEMA_LOCATION

Required/Optional: *Optional*

This directive allows the user to set the URL location of the GMLSf compliance levels schema. This URL is transferred to `schemaLocation` attribute in the `<import>` element for the "http://www.opengis.net/gmlsf" namespace in the generated GMLSf `.xsd` document. The default value for this directive is "http://schemas.opengis.net/gml/3.1.1/profiles/gmlsfProfile/1.0.0/gmlsfLevels.xsd".

Example:

```
GMLSf_COMPLIANCE_LEVEL_SCHEMA_LOCATION http://www.example.com/gmlsf/1.0
```

Workbench Parameter: *GMLSf compliance levels schema location*

VERSION

Required/Optional: *Optional*

This directive allows the user to set the version of the GML schema document. The default value is "1.0.0".

Example:

```
GMLSf_VERSION 4.1
```

Workbench Parameter: *GMLSf application schema version*

SRS_NAME

Required/Optional: *Optional*

This directive allows the user to overwrite the CRS value that is written in the GML instance's `srsName` attributes. By default, the FME coordinate system name in a feature is transferred directly onto the `srsName` attribute. This directive allows users to provide their own URN CRS string for the `srsName` attributes.

Example:

```
GMLSf_SRS_NAME urn:ogc:def:crs:EPSG:6.6:4326
```

Workbench Parameter: *GML srsName*

SRS_AXIS_ORDER

Required/Optional: *Optional (Required if using SRS_NAME)*

This directive is required when the `SRS_NAME` directive is used. It specifies the axis order for a coordinate tuple in a GML `<pos>` or `<posList>` element. The valid values for this directive are "1,2", "2,1", "1,2,3" and "2,1,3". There is no default value for this directive.

For example, if `SRS_NAME` is set to "urn:ogc:def:crs:EPSG:6.6:4326", then the `SRS_AXIS_ORDER` should be set to "2,1" so that the coordinates in the GML `<pos>` and `<posList>` elements are written in lat-lon order:

```
GMLSf_SRS_AXIS_ORDER 2,1
```

Workbench Parameter: *GML SRS Axis Order*

SUPPRESS_XSI_SCHEMALOCATION

Required/Optional: *Optional*

Setting this directive to YES suppresses the output of the `xsi:schemaLocation` attribute in the GML instance's root element. The `xsi:schemaLocation` in an XML document instance is not a mandatory attribute – it is merely a hint which an XML processor may choose to ignore. Setting this directive to YES suppresses the output of the `xsi:schemaLocation` attribute in the output GML instance. The default value for this directive is NO.

Mapping File Syntax

```
GMLSF_SUPPRESS_XSI_SCHEMALOCATION YES
```

Workbench Parameter: *Suppress xsi:schemaLocation attribute*

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

The `xml_type` attribute identifies the FME GML geometry. The valid values for this attribute are:

<code>xml_type</code>	Description
<code>xml_no_geom</code>	FME Feature with no geometry.
<code>xml_point</code>	Point feature.
<code>xml_line</code>	Linear feature.
<code>xml_area</code>	Areal feature, may be a donut.

Other attributes, including the feature's feature type, are dependent on the GML application schema.

No Geometry

`xml_type`: `xml_no_geom`

Features having their `xml_type` attribute set to `xml_no_geom` do not contain any geometry data.

Points

`xml_type`: `xml_point`

Features having their `xml_type` set to `xml_point` are single coordinate features.

Lines

`xml_type`: `xml_line`

Features having their `xml_type` set to `xml_line` are polyline features and have at least two coordinates.

Areas

`xml_type`: `xml_area`

Features having their `xml_type` set to `xml_area` are either a single closed polyline feature (simple closed polygon), a donut, or an aggregate of donuts (and/or simple polygons). A simple closed polygon contains at least four coordinates, with the first and last coordinate being equal.

DEF Lines

The DEF lines control the generation of the GML application schema. The syntax of a GML DEF line is:

```
<writerKeyword>_DEF <feature type> \  
  [<attribute name> <attribute type>]*
```

Where the valid values for <attribute type> are: xml_char(width), xml_int32, xml_real32, xml_decimal(width,decimal), xml_boolean, and xml_real64.

The <attribute name> can either be a simple attribute, a list attribute, or a structured attribute. Simple attributes have no embedded period, and they are mapped into simple type properties in the GML application schema. List attributes end with a "{}" suffix, and are mapped into simple type property declarations whose maxOccurs are set to unbounded. Structured attributes have embedded periods, each component in a structured attribute is separated by a period, and each component may also be a list. Structured attributes are mapped into complex type properties in the GML application schema. For example, the structured FME attributes, "changeHistory.Date", and "changeHistory.Reason" are mapped into a changeHistory element whose complex type is defined as a sequence of Date and Reason elements both of which are defined as simple types.

For example, the DEF line:

```
GML_DEF F1 \  
  code xml_int32 \  
  changeHistory.Date xml_char(10) \  
  changeHistory.Reason xml_char(254)
```

Generates the following complex type definition:

```
<complexType name="F1Type">  
  <complexContent>  
    <extension base="gml:AbstractFeatureType">  
      <sequence>  
        <element name="code" minOccurs="0" type="integer"/>  
        <element name="changeHistory" minOccurs="0">  
          <complexType>  
            <sequence>  
              <element name="Date" minOccurs="0">  
                <simpleType>  
                  <restriction base="string">  
                    <maxLength value="10"/>  
                  </restriction>  
                </simpleType>  
              </element>  
              <element name="Reason" minOccurs="0">  
                <simpleType>  
                  <restriction base="string">  
                    <maxLength value="254"/>  
                  </restriction>  
                </simpleType>  
              </element>  
            </sequence>  
          </complexType>  
        </element>  
      </sequence>  
    </extension>  
  </complexContent>  
</complexType>
```

Google Earth KML Reader/Writer

Format Notes:

The KML format is described in detail in the *KML Documentation Introduction* at <http://code.google.com/apis/kml/documentation/>. Users who want to create complex KML should familiarize themselves with the KML 2.2 specification that is available on Google's website: <http://earth.google.com/kml>.

The Google Earth KML Reader and Writer allow FME to read and write KML files. This chapter refers to the Reader and Writer as the *KML Reader/Writer*.

This KML overview is from the Open Geospatial Consortium (OGC®) KML 2.2 RFC:

"KML is an XML grammar used to encode and transport representations of geographic data for display in an earth browser. Put simply: KML encodes what to show in an earth browser, and how to show it. KML uses a tag-based structure with nested elements and attributes and is based on the XML standard.

The KML community is wide and varied. Casual users create KML Placemarks to identify their homes, describe journeys, and plan cross-country hikes and cycling adventures. Scientists use KML to provide detailed mappings of resources, models, and trends such as volcanic eruptions, weather patterns, earthquake activity, and mineral deposits. Real estate professionals, architects, and city development agencies use KML to propose construction and visualize plans. Students and teachers use KML to explore people, places, and events, both historic and current. Organizations such as National Geographic, UNESCO, and the Smithsonian have all used KML to display their rich sets of global data.

KML documents and their related images (if any) may be compressed using the ZIP format into KMZ archives. KML documents and KMZ archives may be shared by e-mail, hosted locally for sharing within a private internet, or hosted on a web server."

Overview

The KML reader reads KML datasets that conform to the KML 2.0, 2.1, and 2.2 specifications. The KML Writer will write datasets that conform to the KML 2.2 specification. In addition the KML Writer also supports writing Google's 'gx' extensions to the KML 2.2 specification.

FME's KML support can be used without knowledge of the KML specification.

Note: Users who want to create complex KML should familiarize themselves with the KML 2.2 specification that is available on Google's website: <http://earth.google.com/kml>.

Deprecation Advisory

FME offers two plug-ins capable of reading and writing KML datasets: KML and OGCKML.

The KML plug-in is the original KML plug-in designed to read and write KML 2.0 datasets, and the OGCKML plug-in is designed to conform to the latest KML 2.2 specification.

The older KML plug-in is now deprecated, and will be removed from FME. Translations using the older KML plug-in should be migrated to the new plug-in, which has many more features.

Google Earth (OGCKML) Quick Facts

Format Type Identifier	OGCKML
Reader/Writer	Reader/Writer
Licensing Level	Base
Dependencies	None
Dataset Type	Reader: File Writer: Directory/File
Feature Type	Varies
Typical File Extensions	.kml, .kmz
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	Yes
Spatial Index	Never
Schema Required	No
Transaction Support	No
Geometry Type	kml_type

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	yes	point	yes
circles	no	polygon	yes
circular arc	no	raster	yes
donut polygon	yes	solid	yes
elliptical arc	no	surface	yes
ellipses	no	text	yes
line	yes	z values	yes
none	no		

Reader Overview

The KML reader supports reading KML 2.0, 2.1, and 2.2 datasets.

Reader Directives and Workbench Parameters

The KML Reader processes several directives in the mapping file. These are all prefixed by the current <ReaderKeyword>_. By default, the Reader Keyword is KML21.

DATASET

Required/Optional: *Required*

This directive specifies the location of the input KML dataset. The input KML dataset can be a kml file, a kmz file, a folder containing a kml file, or a URL referencing a KML file.

Workbench Parameter: *Source OpenGIS KML Encoding Standard File or URL*

LOG_VERBOSE

Required/Optional: *Optional*

Specifies whether or not the reader should increase the logging verbosity. Possible values are Yes and No. The default is No.

Workbench Parameter: *Verbose Logging*

FAIL_ON_NETWORK_ERROR

Required/Optional: *Optional*

Specifies whether or not the reader should terminate the translation if a network error occurs. Possible values are 'yes' or 'no'. The default is 'no'.

Workbench Parameter: *Fail on Network Errors*

DELETE_DOWNLOADED_FILES

Required/Optional: *Optional*

Specifies whether or not the reader should delete temporary files downloaded as part of the reading process. Possible values are 'yes' or 'no'. The default is 'yes'.

Workbench Parameter: *Delete Downloaded Files*

RASTER_READ_MODE

Required/Optional: *Optional*

Specifies whether the images files referenced by Overlay elements should be read as raster geometry. Possible values are 'all', 'groundoverlay', or 'none'. If the option is 'all', all GroundOverlay, PhotoOverlay, and ScreenOverlay images will be read as raster geometry. If the selected option is 'groundoverlay', then only GroundOverlay images will be read.

The default is 'groundoverlay'.

Raster's read from GroundOverlay images will be georeferenced using coordinate values from the GroundOverlay.

Workbench Parameter: *Read Overlays As Rasters*

PROXY_URL

Required/Optional: *Optional*

Specifies the url of a proxy server that will be used for all href traversal.

Workbench Parameter: *Proxy Url*

PROXY_PORT

Required/Optional: *Optional*

Specifies the port number for the proxy server. Not valid if PROXY_URL is not supported.

Workbench Parameter: *Proxy Port*

PROXY_USERNAME

Required/Optional: *Optional*

Specifies the username to use to login to the server for the proxy server. Not valid if PROXY_URL is not supported.

Workbench Parameter: *Proxy User Name*

PROXY_AUTH_METHOD

Required/Optional: *Optional*

Specifies the authentication method to use to login to the server for the proxy server. Not valid if PROXY_URL and PROXY_USERNAME are not supported.

Workbench Parameter: *Proxy Authentication Method*

SCAN_SCHEMA

Required/Optional: *Optional*

Specifies whether or not the reader should scan the KML files for schema elements. Possible values are 'yes' or 'no'. The default is 'yes'.

If 'no' is specified, KML elements will be read using the fixed schema. KML datasets using KML 2.1 schema may not be read properly unless this option is set to 'yes'.

Workbench Parameter: *Scan Schema*

TRAVERSE_NETWORKLINKS

Required/Optional: *Optional*

Specifies whether or not NetworkLink or schemaUrl references to external KML files should be traversed. I.e. to read the referenced document.

Possible values are 'all', 'local', or 'none'. If 'local' is specified, then references will only be traversed if they refer to a file on the local filesystem.

The default value is 'all'.

Workbench Parameter: *Traverse NetworkLinks*

MAX_NETWORKLINK_TRAVERSAL_DEPTH

Required/Optional: *Optional*

Specifies the max depth of the traversal tree.

The traversal depth is the number of links that must be traversed to get from the original file to the root file. For example, if the dataset root refers to DocB, which refers to DocC, the traversal depth is 2.

The default value is 5'.

Workbench Parameter: *Maximum NetworkLinks Traversal Depth*

BBOX_WEST, BBOX_SOUTH, BBOX_EAST, BBOX_NORTH

Required/Optional: *Optional*

These 4 parameters specify the bounding box that will be used to specify the BBOX passed to web services that provide KML data.

By default, no BBOX is provided, and the web service will provide all data provided.

The reader will not use these values to filter features.

Workbench Parameters: *Query Bounding Box Min X, Query Bounding Box Max X, Query Bounding Box Min Y, Query Bounding Box Max Y*

MOVE_TO_KML_WORLD_COORDSYS

Specifies whether or not the location metadata in the KML Placemark will be applied to the model geometry.

Required/Optional

Optional

Values

- yes (default): a custom coordinate system with a Azimuth Equal Distance projection will be applied to the feature.
- no: the feature will be output without any coordinate system.

* Workbench Parameter

Move to World Coordinate System

APPLY_MODEL_TRANSFORMS

Specifies whether or not scaling and orientation metadata in the KML Placemark will be applied to the model geometry prior to being output.

Required/Optional

Optional

Values

- yes: The transformations will be applied.
- no (default)

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the `mitab.dll` in the FME home directory to `mapinfo.dll`.

The syntax of the `MAPINFO_SEARCH_ENVELOPE` directive is:

```
MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

```
MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The `COORDINATE_SYSTEM` directive, which specifies the coordinate system associated with the data to be read, must always be set if the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` to the reader `COORDINATE_SYSTEM` prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Resource Traversal

A resource traversal occurs when the KML reader uses a resource reference to retrieve a file external to the current KML file. The KML reader is capable of traversing the following types of resource references:

- <NetworkLink><Link><href>
- <GroundOverlay><Icon><href>
- <ScreenOverlay><Icon><href>
- <PhotoOverlay><Icon><href>
- <SchemaData @schemaUrl>
- <Placemark><Model><Link><href>

At this time, <Placemark><styleUrl> is not traversed.

Resource Location

Resources can either be local, or remote. Local resources are located on either the local filesystem, or a mounted network drive. Examples of local resources would be "C: emp\foo.jpg" or "\\FILES\data\bar.kml. Remote resources are resources that are accessed via internet protocols. The KML reader supports the ftp and http protocols.

The value of the reader's TRAVERSE_NETWORKLINK directive determines how resource references are handled.

All: All resource references are traversed

Local: Only local references to resources are traversed. I.e. a NetworkLink reference to a local file will be traversed, but a reference to a file on a remote web server will not.

None: No resource references are traversed. This may limit some of the reader's functionality.

Relative References

Resource references often refer to resources in a location that is relative to the referring resource. An example would be roads.kml referring to schema.kml via a schemaUrl. The KML reader attempts to resolve relative references into an absolute reference by maintaining a record of the location from which the referring resource was retrieved. For example if roads.kml is retrieved from http://example.com/roads.kml, the KML reader will attempt to load schema.kml from http://example.com/schema.kml

Remote Resources Downloads

Files downloaded as a result of traversing remote resource references are each stored in unique folders in \$(FME_TEMP). Unique download locations ensure that files with similar filenames do not overwrite each other. Upon the completion of the translation, all downloaded files will be removed. To override this behaviour, the KML reader's DELETE_DOWNLOADED_FILES directive can be set to no'.

Resources can also be downloaded from remote web services, for example http://example.com/mapdata.php?layer=roads, may return KML data. As part of the downloading process, the reader performs two actions:

1. The mime-type of the server's response is compared against a set of expected mime-types. I.e. to verify that it is KML data (or image data for Overlays).
2. Any non-alphanumeric characters in the resulting filename are replaced by underscores. The appropriate file extension is also provided as needed. For example, if the above service responded with KML data, it would be stored in mapdata_php_layer_roads.kml

Traversal Depth

The KML reader is capable of limiting resource traversal depth. This option allows users to control how much data the KML reader retrieves in a give translation. For example, if reading a complex dataset from a web-service, it is possible for a.kml to refer to b.kml, which refers to c.kml, etc. By default, the traversal depth is capped at 5, but can be increased as necessary using the KML reader's MAX_NETWORKLINK_TRAVERSAL_DEPTH directive.

There are no restrictions on traversal breadth, i.e. the number of traversals per file.

Network Errors

The KML reader has the option of ignoring, but logging, network errors that are encountered during the translation. Service provider issues, or temporary network outages may cause such errors. By default, network errors will not result in a fatal error; the FAIL_ON_NETWORK_ERROR reader directive can be used to change the default behavior.

Writer Overview

The KML writer is capable of creating uncompressed (.kml) and compressed (.kmz) KML documents.

Note: All Z values are interpreted in meters, so you may need to manually convert between feet and meters.

Coordinate System Reprojection

The writer will automatically reproject features to LL-WGS84 if they are tagged with a different coordinate system.

Writer Directives and Workbench Parameters

The KML Writer processes several keywords in the mapping file. These are all prefixed by the current <WriterKeyword>_. By default, the Writer keyword is KML21.

DATASET

Required/Optional: *Required*

This keyword specifies the location for the output KML dataset. If the dataset name contains the extension ".kml", a single KML document will be created, if the dataset name contains ".kmz" a KMZ file will be created. Otherwise, it is assumed that the dataset is a directory. In the case of either a KMZ, or a directory dataset, the filename of the "root" document will be doc.kml unless otherwise specified using the DOCUMENT_FILENAME keyword.

Workbench Parameter: *Destination OpenGIS KML Encoding Standard File*

DOCUMENT_FILENAME

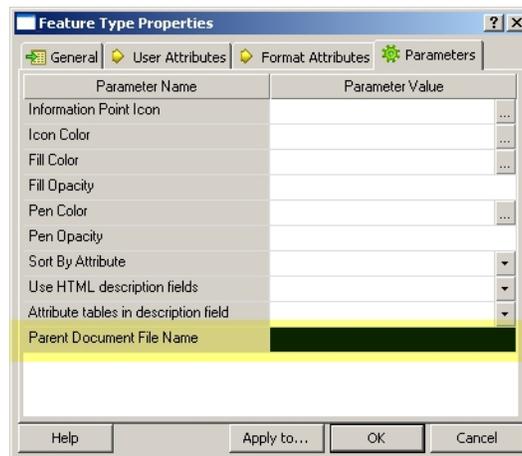
Required/Optional: *Optional*

This keyword specifies an alternative filename for the root document if the dataset is either a KMZ file or a directory. Use of this keyword may result in a dataset that is unreadable by Google Earth.

Workbench Parameter: *Parent Document Filename*

This parameter is specified at the feature-type level. Specifying a filename forces a certain set of features to be segregated in a separate file.

Open the Destination Feature Type Properties dialog and click the Parameters tab.



OUTPUT_SCHEMA

Required/Optional: *Optional*

Specifies whether or not schema elements should be output. Possible values are Yes, or No. The default is Yes.

Workbench Parameter: *Create KML Schema Elements*

UPPERCASE_SCHEMA

Note: This directive is deprecated as of FME 2008.

Required/Optional: *Optional*

Specifies whether or not the metadata names should be converted to uppercase to avoid namespace clashes.

AUTO_CREATE_NETWORK_LINKS

Required/Optional: *Optional*

Specifies whether or not NetworkLink elements should be automatically generated when documents are referenced. Possible values are Yes, or No. The default is Yes.

Workbench Parameter: *Automatically Create Network Links*

ATTR_IN_DESCRIPTION

Required/Optional: *Optional*

Determines whether or not user-defined attributes will be included in the feature's description field. The value of the HTML_DESCRIPTIONS preference will determine if the attributes, and their corresponding values will be displayed using a HTML table, or a series of colon-separated key-value pairs. Values can be Yes or No. The default is Yes.

Workbench Parameter: *Attribute tables in description field*

HTML_DESCRIPTIONS

Required/Optional: *Optional*

Specifies whether or not the description, snippet, and balloon text elements should be wrapped in a CDATA block. Possible values are Yes, or No. The default is Yes.

Workbench Parameter: *Use HTML description fields*

DETECT_RASTERS

Required/Optional: *Optional*

Controls whether or not the writer will examine incoming features to determine if they have raster geometry. If a raster geometry is detected, the feature will be forced to be a GroundOverlay feature. The value can be Yes or No. The default is Yes.

Workbench Parameter: *Automatically detect raster features*

RASTER_MODE

Required/Optional: *Optional*

Specifies how the writer will handle raster features. This preference only applies if the DETECT_RASTERS preference is enabled. The possible values, and their corresponding behaviours are detailed below

- **write** - (Default behavior) Write the raster feature using the FME writer that corresponds to the RASTER_FORMAT option. This option allows the KML writer to handle raster features irregardless of their origin, as well as to take advantage of any resampling performed during the transformation process.
- **copy** - Copy the image file that the raster feature originated in to the directory of the KML dataset. The resulting <GroundOverlay> element will reference the copy of the image file. Note: This will only work for PNG, JPEG, or TIFF images.
- **relative** - Similar to the "copy" option, but leaves the image file in its original location. This option avoids gratuitous file copying.

Workbench Parameter: *Raster Handling Mode*

RASTER_FORMAT

Required/Optional: *Optional*

Specifies what file format should be used if the RASTER_MODE preference is "write". The possible values are "tiff" and "jpeg". The default is "jpeg".

Workbench Parameter: *Raster Output Format*

RASTER_COMPRESSION_LEVEL

Required/Optional: *Optional*

Specifies the raster compression level that should be used if the RASTER_MODE preference is "write". See the GEO-TIFF and JPEG writers for appropriate values.

Workbench Parameter: *Not applicable*

INFORMATION_POINT_ICON

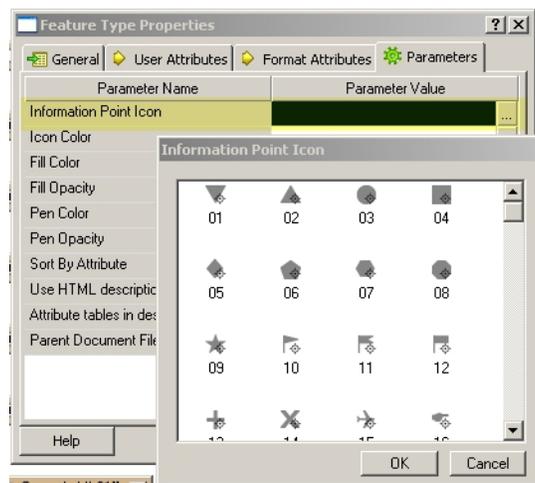
Required/Optional: *Optional*

Specifies the icon that should be used for information point icons. The value should be either the full path to the icon, or the name of an icon in \$(FME_HOME)/icons. If no value is specified, the information point icon will not be created.

Workbench Parameter: *Information Point Icon*

This parameter is specified at the feature-type level, and specifies the information point icons that will be used for certain features.

Open the Destination Feature Type Properties dialog and click the Parameters tab. Click the Browse button to the right of the Information Point Icon field.



COPY_ICON

Required/Optional: *Optional*

Specifies whether icons should be copied from their location to the dataset's images directory. The possible values are Yes and No. The default is Yes.

Note: This only applies to icons referenced by the kml_icon attribute, icons referenced by the kml_icon_href will not be copied. If the kml_icon_href is present, the value of kml_icon will be ignored.

Workbench Parameter: *Copy icons to destination dataset*

LOG_VERBOSE

Required/Optional: *Optional*

Specifies whether or not the writer should increase the logging verbosity. Possible values are Yes and No. The default is No.

Workbench Parameter: *Verbose Logging*

ORIENTATION

Required/Optional: *Optional*

Specifies whether @Orient should be used to orient features prior to writing. The possible values are None, Right and Left. The default is None.

Workbench Parameter: *Force Geometry Orientation*

REGIONATE_DATA

Required/Optional: *Optional*

Specifies whether or not features with vector geometry (Placemarks) should be passed through the regionator pipeline prior to writing. Possible values are Yes and No. The default is No.

Workbench Parameter: *Regionate Vectors (beta)*

REGIONATOR_PIPELINE

Required/Optional: *Optional*

Specifies a specific FME factory pipeline that will be used to pre-process (regionate) vector features prior to being written by the writer.

The value is either the path to the pipeline file, or the name of a file located in \$(FME_HOME)/plugins/kml21.

The default value is fmregionator.fmi.

Workbench Parameter: *Vector Regionation Pipeline*

EXEC_GO_PIPELINE

Required/Optional: *Optional*

Specifies whether or not features with raster geometry (GroundOverlays) should be passed through the GO pyramid pipeline prior to writing. Possible values are yes' or no'. The default is no'.

Workbench Parameter: *Pyramid GroundOverlays (beta)*

GO_PYRAMIDER_PIPELINE

Required/Optional: *Optional*

Specifies a specific FME factory pipeline that will be used to pre-process (pyramid) raster features prior to being written by the writer.

The value is either the path to the pipeline file, or the name of a file located in \$(FME_HOME)/plugins/kml21.

The default value is fmgroundoverlaypyramider.fmi.

Workbench Parameter: *GroundOverlay Pipeline*

EXEC_PO_PIPELINE

Required/Optional: *Optional*

Specifies whether or not PhotoOverlay features should be passed through the PO pyramid pipeline prior to writing. Possible values are 'yes' or 'no'. The default is 'no'.

Workbench Parameter: *Pyramid PhotoOverlays (beta)*

PO_PYRAMIDER_PIPELINE

Required/Optional: *Optional*

Specifies a specific FME factory pipeline that will be used to pre-process (pyramid) raster features prior to being written by the writer.

The value is either the path to the pipeline file, or the name of a file located in \$(FME_HOME)/plugins/kml21.

No default pipeline is provided.

Workbench Parameter: *PhotoOverlay Pipeline*

WATERMARK_NAME

Required/Optional: *Optional*

The name element value of the ScreenOverlay element used to provide the watermark.

Workbench Parameter: *Watermark Name*

WATERMARK_SNIPPET

Required/Optional: *Optional*

The snippet element value of the ScreenOverlay element used to provide the watermark.

Workbench Parameter: *Watermark Snippet Text*

WATERMARK_ICON

Required/Optional: *Optional*

The icon to use for the ScreenOverlay element used to provide the watermark. The value can be any value allowable for the kml_icon format attribute.

Workbench Parameter: *Watermark Overlay Icon*

DATASET_HINT

Required/Optional: *Optional*

The value of the hint' attribute of the kml element for the dataset.

The default is none. Users wishing to write datasets for Google Sky should use the value "target=sky". See the Google KML documentation for further information.

Workbench Parameter: *KML hint attribute*

STYLE_DOC

Required/Optional: *Optional*

Specifies the filename of an external kml file that will be used to store Style or StyleMap elements. If specified all Style and Stylemap elements will be routed to this document.

The value cannot be a path, and must include the ".kml" extension.

Workbench Parameter: *External Style Document*

SCHEMA_DOC

Required/Optional: *Optional*

Specifies the filename of an external kml file that will be used to store Schema information for the dataset. If specified, Schema elements will be written to this file, and the schemaUrl value of each placemark will be updated accordingly.

The value cannot be a path, and must include the ".kml" extension.

Workbench Parameter: *External Schema Document*

ATOM_AUTHOR_NAME

Required/Optional: *Optional*

Specifies the value of the <atom:name> element for the dataset.

Workbench Parameter: *Author Name*

ATOM_AUTHOR_EMAIL

Required/Optional: *Optional*

Specifies the value of the <atom:email> element for the dataset.

Workbench Parameter: *Author Email*

ATOM_LINK_HREF

Required/Optional: *Optional*

Specifies the value of the <atom:link> element for the dataset.

Workbench Parameter: *Author URL*

OMIT_DOCUMENT_ELEMENT

Required/Optional: *Optional*

Controls whether a root-level <Document> element is created.

Possible values are Yes and No. The default is No.

Workbench Parameter: *Omit Document Element*

TARGET_HREF

Required/Optional: *Optional*

Specifies the URL value of the targetHref for the NetworkLinkControl Updates

Workbench Parameter: *Target Href*

CREATE_EMPTY_FOLDERS

Required/Optional: *Optional*

If the value is "no" (default), then the folders for user-defined feature types will only be created if at least one feature is written with a feature type corresponding to the folder name. If the value is "yes", then a <Folder> will be created for each user-defined feature type, regardless of whether or not any features are written with that feature type. Note: This option does not apply to feature type fannout.

Workbench Parameter: *Create Empty Folders*

FANOUT_TYPE

Required/Optional: *Optional*

If the value is "folder", then a single level of folders will be created, where each folder's name maps to the value of the fanout attribute. If the value is "subfolder", then a two-level folder hierarchy will be created, where the top-level folder's name will be the same as the original feature type, and each sub-folder's name maps to the value of the fanout attribute.

Workbench Parameter: *Fanout Type*

MOVE_TO_KML_LOCAL_COORDSYS

Specifies whether or not features with model geometry should be reprojected such that the model geometry is a local coordinate system with meter units.

Required/Optional

Optional

Values

- yes (default): The input feature must have a valid coordinate system, and will be reprojected as necessary.
- no: The placement location of the model must be specified via format attributes, and the geometry will be written to a COLLADA file as-is.

Note: Google Earth requires the geometry to use values with valid COLLADA units.

*** Workbench Parameter**

Move To Local Coordinate System

WRITE_TEXTURES_TXT_FILE

Specifies whether or not a textures.txt file should be generated to map the texture references in the output COLLADA files to their corresponding texture files.

Required/Optional

Optional

Values

- yes: A single textures.txt file will be generated for the whole dataset.
- no (default): Texture mapping will be performed by adding the required <ResourceMap> elements to the Placemark element's Model geometry.

Note: The OGC KML 2.2 specification has deprecated use of the textures.txt file in favor of ResourceMap elements.

* Workbench Parameter

Write texture list to textures.txt

DEF Line Parameters

The following DEF line parameters can be specified for user-defined features. They do not apply to writer-specific feature type names, such as "Placemark" or "Folder".

DEF line parameters that affect Placemark styling correspond to a single <Style> element that will be created for each feature type. All features with that feature type will have their <styleUrl> element set to refer to the feature type <Style> element. Note: if a given feature already has a kml_style_url attribute, its value will take precedence.

Name	Value
KML21_DOCUMENT_FILENAME	The name of the file that will be used to store all the features of the feature type.
KML21_SORT_BY_ATTRIBUTE	The name of the attribute that will be used to sort features within the folder that corresponds to the feature type. Only applies to user-defined feature types. If unspecified, the order of the features will be in the original arrival order.
KML21_OPACITY	Specifies the opacity of both the fill and pen colors. Valid values are 0 to 1.
KML21_PEN_OPACITY	Specifies the pen opacity. Valid values are 0 to 1.
KML21_FILL_OPACITY	Specifies the fill opacity. Valid values are 0 to 1.
KML21_FILL_COLOR	Specifies the fill color. Valid values are any fme color specification.
KML21_PEN_COLOR	Specifies the pen color. Valid values are any fme color specification.
KML21_INFORMATION_POINT_ICON	Specifies the information point icon to use for each feature.

Name	Value
KML21_ICON_COLOR	Specifies the color for each information point icon. Valid values are any fme color specification.
KML21_LABEL_COLOR	Specifies the label color. Valid values are any fme color specification.

In addition, the following define parameters override the corresponding writer directives for the specified feature type:

```

KML21_ATTR_IN_DESCRIPTION
KML21_HTML_DESCRIPTIONS
KML21_RASTER_MODE
KML21_RASTER_FORMAT
KML21_RASTER_COMPRESSION_LEVEL
KML21_INFORMATION_POINT_ICON
KML21_COPY_ICON

```

Writer Pipelines

Writer factory pipelines provides users with an opportunity to alter the KML writer's feature processing, and reduce translation complexity. Pipelines are currently used to provide the KML writer's vector regionation and raster image pyramiding functionality, but could also be used for other purposes.

To use the pipeline functionality first enable the pipeline, and then specify the pipeline file. If no pipeline file is specified, a default pipeline file from \$(FME_HOME)/plugins/kml21 will be used.

Pipeline files can be located in \$(FME_HOME)/plugins/kml21 or \$(FME_MF_DIR). In addition, a full path to the pipeline file can also be specified.

Vector (Placemark)

The vector pipeline processes features with vector geometry (typically Placemarks). The default pipeline, fmeregionator.fmi performs regionation.

Note: The default regionator pipeline requires Python.

Google Earth has difficulty opening and displaying large KML datasets that are contained in a single KML file. Regionation provides a mechanism for dividing vector data into many regions, each of which may contain sub-regions. Each region is contained within a separate kml file; Region and NetworkLink elements are used to instruct Google Earth when to load each set of vector data.

The basic idea behind regionation is that users should first be presented with the most important features, and as the user zooms to lower altitudes, less important features are displayed. The regionation pipeline uses relative weights to determine which features are displayed first.

The following format attributes are used to configure the regionation pipeline. See the sections below for further explanation.

KML Format Attribute	Notes
kml_region_weight	A floating point value specifying the relative weight of the feature. If not present, heuristics will be applied to calculate a weight for the feature.

KML Format Attribute	Notes
kml_region_group	The name of a group of features that will be regionated together. By default, all features will be regionated together.
kml_features_per_region	The number of features to include in each region. The default value is 16.
kml_minimum_lod	The minimum lod for each region. The default is 256.

Relative Weights

During the regionation process, all input features are sorted according to their weight. Features with the highest weight are displayed first. It's important to understand the feature weight only determines relative display order; it does not guarantee that an item will be displayed at a certain altitude.

A feature's weight can be specified using the kml_region_weight format attribute. If no weight is specified, the following heuristic will be used to create a weight for each feature.

Feature Geometry	Weight Calculation
Point	Constant value of 1.0
Line	@Length()*1000
Area (and all others)	@Area(1000)

Level Of Detail

Each region is defined as a quadrant. Google Earth decides whether or not each region is displayed by comparing the number of pixels required to display the region in Google Earth to a particular minimum Level of Detail (LoD) specification. By default, the regionation pipeline uses a minimum LoD of 256, which can be overridden with the kml_minimum_lod format attribute. It is important to note that the the maximum LoD is always set to -1, which means that the region will always be turned on once the minimum LoD has been achieved.

Features per Quadrant

Each region can contain a certain number of features. When the region is activated, all features in the region are displayed. By default, each region contains at most 16 features, however this can be overridden using the kml_features_per_region format attribute.

Region Groups

By default, all vector features written to the writer will be regionated together. By using the kml_region_group format attribute, features can be assigned to groups for separated regionation.

Note: The regionation pipeline assumes that input features are sorted according to their individual region group. When the pipeline encounters a feature with a groupname that differs from the current group name, the current group will be completed, and regionation of a new group will begin.

Raster (GroundOverlay)

The raster pipeline processes features with raster geometry (typically GroundOverlays). The default pipeline, fme-groundoverlaypyramider.fmi performs imagepyramidding.

Note: The default pipeline requires Python. An alternative pipeline, fme-groundoverlaypyramider_nopython.fmi, is also available.

Each feature will have a `kml_images_directory` format attribute added prior to input to the pipeline. This format attribute contains the location of the dataset's images' directory.

Raster (PhotoOverlay)

The raster pipeline processes features with PhotoOverlay feature types. No default pipeline is provided.

Each feature will have a `kml_images_directory` format attribute added prior to input to the pipeline. This format attribute contains the location of the dataset's images' directory.

Watermarks

Watermarks are a simple shortcut for creating a ScreenOverlay in the lower left-hand corner of the Google Earth view window.

Watermarks require the overlay icon (specified using the `WATERMARK_ICON` directive). An optional name & snippet, which is displayed in the Google Earth tree) can also be provided. The `WATERMARK_NAME` and `WATERMARK_SNIPPET` directives are used to specify the name and snippet.

Atom Metadata

Atom metadata is used to provide data about the author, and point to a related website. This information is used by Google for inclusion in GeoSearch results. To specify the author name & email, use the `ATOM_AUTHOR_NAME` and `ATOM_AUTHOR_EMAIL` directives. The `ATOM_LINK_HREF` directive can be used to specify the url of the website that will contain the KML file.

Style Management and Common Styles

The KML writer maintains a global 'registry' of style definitions. The style registry is used to simplify references to style elements, as well as to enable routing of style elements to external files.

Common Styles

Common styles provide an FME-twist on providing styling for KML elements that allow arbitrary groups of FME features to share the same KML `<Style>` element. The goal of this feature is two-fold:

- Make it trivially easy for arbitrary groups of features to share styling without having to manually create Style features.
- Reduce the size of KML datasets by preventing the creation of inline Style elements.

This feature was primarily designed to support the KMLStyler transformer, but may have other applications.

To enable common styling, each feature in the group of features being styled must satisfy two conditions:

1. They must have a `kml_common_style` attribute that is unique for the group
2. They must have KML Style element attributes that define the styles for the group.

As each feature is added to its corresponding document, it is checked for the `kml_common_style` attribute. If the corresponding style does not exist yet, it is created from the feature's attributes, otherwise the feature's style attributes are ignored. It is very important to note that this is a "first feature wins" scenario, and unless you can guarantee the order in which the KML writer receives the features, you are better off ensuring that all features have the same style information.

Unique Id Values

The style registry assumes that each style id is unique. Style ids are specified by the `kml_id` format attribute of features with Style and StyleMap feature types. This includes Style elements created as a result of Common Style' references; the resulting Style element has an id that is the same as the value of the `kml_common_style` format attribute.

Style Element Location

By default, all style elements (Style and StyleMap) are written to the root document. The name of an alternative file can be specified with the `'STYLE_DOC'` writer directive.

In addition, each feature with Style or StyleMap feature type can use the `kml_document` format attribute to specify the name of the document that will contain its associated Style or StyleMap element.

Targeting Styles

Previous versions of the KML writer required the use of the `kml_style_url` attribute to manually reference style elements. Use of `kml_style_url` is still supported, but use of the new `kml_target_style` format attribute is now recommended.

The value of the `kml_target_style` format attribute should be the id of a Style or StyleMap element that is also being written. As the dataset is being written, the writer will resolve each target id into the required style url, irregardless of what kml file the style element resides in.

Implicit StyleMaps

Implicit StyleMaps make it much easier to create StyleMaps. To create an implicit style map, simply set the values of the `kml_target_style_normal` and `kml_target_style_highlight` format attributes to the ids of style elements for the normal and highlight states. The writer will take care of creating the required StyleMap element as needed.

Document Element Omission

Although the KML specification permits multiple child elements of the `<kml>` element, Google Earth, and many other applications only support a single child element. The most common child element is `<Document>`, and the default behavior of the KML writer is to create a `<Document>` element that contains the rest of the dataset's features.

The KML writer's default behavior is sufficient for most users, however there is occasionally a need for alternative root-level elements. The KML writer supports requirement via the `OMIT_DOCUMENT_ELEMENT` directive. If set to 'yes', the writer will skip writing the `<Document>` element, and write the child element of the document as the child of the `<kml>` element.

Note: The Document can only have 1 root element, which can be a folder that can in turn contain other elements.

Raw Text

The `kml_raw_text` format attribute provides a mechanism for users to insert XML snippets directly into the KML document. The value of that attribute will be inserted into the contents of the element that corresponds to the feature being written. This allows users to use features of KML that are not otherwise supported by FME.

Note: It is the user's responsibility to ensure that the value of this attribute is valid XML. Failure to do so may lead to unreadable KML data

Update Mode

The writer supports writing features to either parent `<Document>` element, or a `<NetworkLinkControl>` element that itself contains `<Update>` elements. The writer also supports creating `<AnimatedUpdate>` elements that themselves contain `<Update>` elements. Any feature that has a `kml_update_mode` attribute is designated a "Update" feature, and will be routed to an `<Update>` element. If the `kml_tour` and `kml_animated_update` format attributes are specified, the update feature will be routed to the corresponding `<Tour>` and `<AnimatedUpdate>` objects, otherwise, the update feature will be routed to a `<NetworkLinkControl>`.

Root `<Document>` Element

It is often desirable for a kml file to either contain a `<NetworkLinkControl>` element or a `<Document>` element, as such, the writer will omit the root `<Document>` element unless it receives features that lack the specification of an update mode.

Ancillary `<NetworkLinkControl>` Elements

Each `<kml>` element may contain zero or one `<NetworkLinkControl>` elements and zero or one `<Document>` elements. If a `<Document>` element and `<NetworkLinkControl>` element are both present, then the `<NetworkLinkControl>` element can be used to either update the contents of a document other than it's sibling `<Document>`, or it can be used to change certain parameters of the calling `<NetworkLink>`, such as refresh time, link name, or expiration.

Update Item Order

Geo browsers, such as Google Earth, process each child element of the <Update> element in sequential order. As a result it is important to ensure that dependencies between individual update items are respected. For example, it is important to ensure that the "Create" for a parent Folder is executed prior to the "Create" for a Placemark. The writer sorts the contents of <Update> element in the following order:

1. Update items are grouped according to their update mode in the following order: Delete, Create, Change
2. Operations on parent nodes are executed prior to those on child nodes
3. Operations on containers are executed prior to those on non-containers.
4. By order of kml_id, where none of the above apply.

Mixing Update and Non-Update Features

The writer will accept groups of features where some of the features have an update mode specified, and some do not. In this situation, the writer will create a <NetworkLinkControl> element, and a <Document> element. The inclusion of a <Document> element will replace the result of the previous NetworkLink request, so it is important to ensure that the update mode and targetHref are appropriately.

In instances where a folder hierarchy contains Folders and Child features with a mix of update and non-update features, the entire hierarchy will be assumed to have an update mode of "create", and will be added to the NetworkLinkControl accordingly.

Target Href

The <targetHref> sub-element of the <Update> element specifies the url of the document to which the update should be applied the url can either be a full url or a partial relative url. The url may specify kml documents other than the url of the NetworkLink service, although the document must be from the same server, and must be the result of a HTTP request. Specification of local files using a file:// will not work.

The writer supports the specification of targetHref either at the writer level via the TARGET_HREF directive, or the kml_target_href format attribute at the feature level. Using a combination of the kml_document_name, and kml_target_href format attributes will permit the create of multiple KML documents, each of which contains it's own <NetworkLinkControl> and associated update features.

Note: The OGC KML 2.2 schema specifies that each <NetworkLinkControl> element can contain at most one <Update> element. As such, each kml file can only update a single kml document.

Feature Representation

KML Dataset Structure

KML is a hierarchical data format that can span multiple data files.

A KML dataset can consist of:

- A single KML file
- A directory with multiple KML files
- A KMZ file, which is a compressed directory containing one or more KML files.

In addition to KML files, a KML dataset can also contain icon images, raster images, and model files. Each KML dataset contains a single KML file that is considered the "root" document for the dataset.

Unless explicitly specified, FME will follow the KML specification, and use "doc.kml" as the filename for the dataset's root.

KML files can reference other KML files via three mechanisms:

1. NetworkLink elements
2. schemaUrl elements
3. the schemaUrl attribute of SchemaData elements

File Structure

KML, like other XML-based formats, can be visualized as a tree structure, where each node in the tree corresponds to an XML element. The root element of a kml file is the <kml> xml element. The <kml> element can contain a <NetworkLinkControl> element, and one other element that inherits from the kml <Feature> type; for practical purposes, the KML writer creates a <Document> element to contain all sub-elements. The KML reader makes no assumptions regarding the contents of the <kml> element.

The <Document> element is a container element that can, in turn contain the following sub-elements:

- **<Folder>**: Another container element that contains the same sub-elements as a <Document>.
- **<Placemark>**: Displays a feature with either Vector or 3D geometry
- **<GroundOverlay>**: Displays (overlays) a feature with Raster geometry
- **<ScreenOverlay>**: Draws a image overlay that is fixed to a certain location in the viewspace.
- **<PhotoOverlay>**: Geographically locates a photograph on the earth
- **<NetworkLink>**: References a KML file or KMZ archive on a local or remote network.
- **<gx:Tour>**: Provides a guided tour of the dataset for the user.

In addition, the <Document> element can contain the following elements that provide important display and meta-information

- **<Style>**: Defines an addressable style group that can be referenced by Stylemaps and Placemarks.
- **<StyleMap>**: Defines a mapping between two different styles.
- **<Schema>**: Defines a schema for embedding custom data within KML

<gx:Tour> elements can contain an arbitrary number of <TourPrimitive> elements. These include:

- **<gx:AnimatedUpdate>**: Uses an <Update> element to create temporary changes to the dataset.
- **<gx:FlyTo>**: Changes the user's view to focus on a certain area.
- **<gx:SoundCue>**: Plays a sound file.
- **<gx:TourControl>**: Changes tour's play mode.
- **<gx:Wait>**: Adds an arbitrary time delay.

With the exception of <Schema> elements, FME maps a fixed schema to each of the aforementioned KML elements; each of the element names map to a well-defined FME feature type. I.e. a feature with a Placemark feature type maps to a <Placemark> kml element.

Identifying Elements

Reading and writing KML elements requires that each element is uniquely identified. This unique identity is used by KML to cross-reference related elements. In addition FME uses the individual element ids to resolve/build a KML hierarchy from FME features.

FME uses the kml_id format attribute to uniquely identify each feature that corresponds to a KML element. FME will generate a unique id if one does not already exist.

If the feature has an update mode specified, but lacks a kml_id, a fatal error will occur.

Documents and Folders

KML has two container element types: <Folder> and <Document>, which in many ways are functionally equivalent. For practical purposes, the KML writer assumes that each KML file has a single <Document> root element, which cannot contain subsequent <Document> elements. Furthermore, the KML writer also assumes that each <Folder> element cannot contain any <Document> elements.

Document Specification

The KML reader and writer are designed to simultaneously handle multiple KML files in the same translation. By default, the KML writer assumes that each feature sent to the writer should be written to the root file/document. To

write a feature to an alternate document, the feature must have a `kml_document` attribute that contains the filename of the destination file/document.

When the KML reader reads a feature from a KML file other than the root file, it will add a `kml_document` attribute to the feature specifying the source documents filename. In addition, because it is possible for the reader to read two files with the same file name, but different file paths, the KML reader will also add the `kml_document_path`, and `kml_document_href` to assist in uniquely identifying the source document for each feature.

Folder Specification

Each feature that is written to the KML writer can also use the `kml_parent` format attribute to identify the `<Folder>` element that should contain the feature. The value of the `kml_parent` format attribute should be the same as the id of the Folder. The KML reader will also add `kml_parent` attributes as necessary. Note: if `kml_parent` is not specified, it is assumed that the container for a given feature is the `<Document>` element.

The `kml_document` and `kml_parent` format attributes are designed to be used in conjunction with each other. I.e. to add a Placemark feature to the folder "A" in document "other.kml", the feature needs to have the following attribute values: `kml_document="other.kml"` and `kml_parent="A"`.

Document and Folder Creation

When Documents and Folders are referenced by the `kml_document`, and `kml_parent` format attributes, the KML writer will implicitly create the appropriate Document and Folder elements to contain the referencing features. Implicitly creating those elements ensures that the writer can properly maintain the dataset hierarchy.

The implicitly created Document and Folder elements can be overridden using features that have Document and Folder feature types. The documents should be uniquely identified using the `kml_document` format attribute, and the folders should be uniquely identified using the `kml_id` attribute. The `kml_document` and `kml_id` values should correspond to the values of the `kml_document` and `kml_parent` attributes on the referencing feature.

The root document/file is a special case. To override the `<Document>` element created for the root file, use a Document feature that does not have a `kml_document` attribute specified; the root document will be assumed.

Element Order

The KML writer uses the following criteria to determine the order of the contents of a KML Folder or Document:

1. If the container has a "sort-by" attribute name specified via the `kml_sort_by_attribute` format attribute, the value of that attribute on each feature will be used to sort the contents of the container in ascending alphanumeric order. The "sort-by" attribute can also be specified using the `KML21_SORT_BY_ATTRIBUTE` define parameter for features with user-defined schema
2. Individual features can over-ride the value of any "sort-by" attribute with the `kml_sort_value` format attribute; the value of which will be used for determining the sort order.
3. If neither of the above criteria is met, then the arrival order of each feature is used to determine the order of the feature in the container. This makes it possible to sort the features prior to being written to the KML writer.

Tours

Each kml file can contain multiple `<Tour>` elements. As a sub-type of the abstract `<Feature>` type, either `<Document>` or `<Folder>` elements can contain one or more `<Tour>` elements. As such, the `kml_document` and `kml_parent` format attributes can be used to determine the location of each `<Tour>` element in the overall hierarchy.

Tour Primitives

Tour primitives (`<AnimatedUpdate>`, `<FlyTo>`, `<SoundCue>`, `<TourControl>`, and `<Wait>`) are routed to specific tours via the `kml_document`, and `kml_tour` format attributes.

This is very similar to the relationship between Folders and their children.

Update Items

Update items are features that are part of the `<Update>` element of a specific `<AnimatedUpdate>`.

A feature is designated as an update item via the `kml_update_mode` format attribute. To route an update item to a specific `<AnimatedUpdate>`, the `kml_tour`, and `kml_animated_update` attributes must be present and contain the ID's of

the corresponding <Tour> and <AnimatedUpdate> elements. The kml_document format attribute can also be specified.

Tour Generation

The KML writer supports automatically generating KML Tours from input Placemark features.

Tour generation is controlled via kml_tour_stop_* format attributes that are specific to Placemark features. The tour consists of tour stops that correspond to each feature. The location of the tour stop corresponds to the center point of the input feature.

In the case of features with line geometry, a series of tour stops will be generated, where each vertex in the line becomes a tour stop.

Extended Data

KML 2.2 provides the option to store attribute data in individual placemarks using either <Data> or <SchemaData> elements. For background, please read: <http://code.google.com/apis/kml/documentation/extendeddata.html>.

The key distinction between Data, and SchemaData is that SchemaData elements are associated with formal schema definitions in <Schema> elements.

The following element attributes are used by the KML reader and writer to read/write KML 2.2 extended data.

The KML reader and writer use a set of kml_data and kml_schema_data structured list attributes to read/write extended data.

KML Element Attribute	KML Element
kml_data{}.name	name attribute of <Data>
kml_data{}.value	<Data><value>
kml_data{}.display_name	<Data><displayName> Optional. Note: xml entities will be encoded on writing.
kml_data{}.display_name_raw_text	<Data><displayName> Optional. Note: display_name values will be written as raw text, i.e., xml entities will not be encoded. CDATA blocks should be added as necessary.
kml_schema_url	schemaUrl attribute of <SchemaData>
kml_schema_data{}.name	name attribute of <SimpleData>
kml_schema_data{}.value	Value of <SimpleData>

Reader

The KML reader will always create the kml_data and kml_schema_data structured list attributes if the corresponding KML elements are found, regardless of whether or not the user-defined schema is used.

Writer

If the feature does not have an associated user-defined feature type, i.e. it is using the fixed schema, then the `kml_data` and `kml_schema_data` structured list attributes can be used to write the feature's extended data.

Notes:

The KML writer does not support manually writing `<Schema>` elements. If `<SchemaData>` elements are manually created, the `kml_schema_url` attribute should be used to specify an external schema definition.

If the feature has an update mode specified, the extended data will not be written.

User Defined Schema

In addition to the fixed schema, and explicit definition of extended data, the KML reader and writer also provide automatic support for user-defined schema.

Reader

The KML reader supports reading KML datasets that use KML 2.0, 2.1, or 2.2 schema declarations.

Upon opening a KML dataset, the KML reader will attempt to scan the dataset's schema by reading all the Schema elements in the file, and traversing all `schemaUrl` and `NetworkLink` references in the file. The default schema scanning behavior can be disabled with the `SCAN_SCHEMA` reader directive.

If Schema declarations are found, the associated Placemark elements will be read as user-defined feature types with the associated set of user-defined attributes. In addition to the user-defined attributes, the `kml_data` and `kml_schema_data` structured list attributes will be created as necessary.

Writer

The KML writer only supports writing KML 2.2 schema elements. Older style schema formats have been deprecated, and are not well supported by applications other than Google Earth.

The KML writer creates the appropriate `<Schema>` elements for each user-defined feature type; each feature is written as either a Placemark or GroundOverlay depending on the feature's geometry type. This behavior can be disabled using the `OUTPUT_SCHEMA` writer directive.

When writing large, multi-file datasets, it is useful to be able to store the schema data in an external file. The KML writer allows the filename of an external schema file to be specified using the `SCHEMA_DOC` directive.

Note: the value must be a filename including the "kml" extension, and will be overwritten during the translation. The `schemaUrl` attributes of each `<SchemaData>` element will be adjusted to point to the external file.

Layers and Feature Type Fanout

The KML writer uses KML Folder elements to create a quasi-layer for each user-defined feature type. The name of the folder will correspond to the name of feature type. If Feature Type Fanout is enabled, either a one or two level folder hierarchy will be created, depending on the value of the `FANOUT_TYPE` directive.

Feature Attributes

The KML Reader and Writer support the following classes of attributes:

- **FME Attributes:** Attributes used by the FME core
- **KML Element Attributes:** Attributes that directly correspond to the value of a KML element.
- **KML Format Attributes:** Attributes that affect how the KML writer behaves, and/or kml elements that are written.
- **User Defined Schema:** Attributes that correspond to the feature's user-defined schema.

Geometry

Vector

The KML reader and writer provide full support for all KML vector geometry types. Vector features are read/written as <Placemark> elements.

Raster

KML supports raster images via three types of overlay elements:

- <GroundOverlay>
- <ScreenOverlay>
- <PhotoOverlay>

Reader

The KML reader can read overlays as either null or raster geometry. If the raster is read as null geometry, the location of the image tile is available in the kml_icon_href attribute.

By default, <GroundOverlay> elements are read with raster geometry, and the other overlay types are read as null geometry. In addition, the KML reader will use the <GroundOverlay> element's <LatLonBox> values to georeference the image.

Writer

Features with either null or raster geometry can be used for writing any of the supported overlay elements. If no raster geometry is provided, it is important to ensure that the feature has the appropriate format or element attributes specifying the location of the image file.

3D

Reader

Full support for 3D models is provided. Any placemark with a <Model> geometry will result in a feature with a 3D geometry corresponding to the contents of the referenced COLLADA file.

Writer

Full support for 3D models is provided. If the feature has one or more 3D geometry components, the feature's entire geometry will be written to a COLLADA file.

<Model> elements can also be written by placing a set of geometry traits on a Placemark feature with Null geometry.

Fixed Schema

The KML reader and writer support a fixed schema where each of the following FME feature types maps directly to a corresponding KML element with the same name. The fixed schema types can be arranged in three groups, according to their parent kml type:

- <Feature> Sub-Types
 - Document
 - Folder
 - Placemark
 - GroundOverlay
 - ScreenOverlay
 - PhotoOverlay
 - NetworkLink
 - Tour

- <StyleSelector> Sub-Types:
 - Style
 - StyleMap
- <TourPrimitive> Sub-Types:
 - AnimatedUpdate
 - FlyTo
 - SoundCue
 - TourControl
 - Wait

Common Element Attributes

The following KML element attributes are common to all of the fixed schema feature types that map to kml types that are sub-types of <Feature>.

KML Element Attribute	KML Element
kml_id	@id, i.e. the elements id attribute.
kml_name	<name>
kml_visibility	<visibility>
kml_open	<open>
kml_address	<address>
kml_phone_number	<phoneNumber>
kml_snippet	<Snippet>
kml_snippet_raw_text	<Snippet> Element value written without xml entity encoding.
kml_description	<description>
kml_description_text	<description> Completely overrides any text provided by FME.
kml_description_raw_text	<description> Element value written without xml entity encoding.
kml_style_url	<styleUrl>
kml_atom_author_name	<atom:author>
kml_atom_author_email	<atom:email>
kml_atom_link_href	<atom:link href="@">
kml_atom_link_rel	<atom:link rel="@">

KML Element Attribute	KML Element
kml_gx_balloon_visibility	<gx:balloonVisibility>

Camera

Defines a "six degrees of freedom" virtual camera for viewing the feature. New in KML 2.2, and can be used in place of LookAt.

The following attributes are required for creating a Camera: kml_camera_longitude, kml_camera_latitude, and kml_camera_altitude.

KML Element Attribute	KML Element
kml_camera_longitude	<Camera> <longitude>
kml_camera_latitude	<Camera> <latitude>
kml_camera_heading	<Camera> <heading>
kml_camera_altitude	<Camera> <altitude>
kml_camera_tilt	<Camera> <tilt>
kml_camera_roll	<Camera> <roll>
kml_camera_altitude_mode	<Camera> <altitudeMode>
kml_gx_camera_altitude_mode	<Camera> <gx:altitudeMode>

LookAt

Defines a simpler virtual camera for viewing the feature.

The following attributes are required for creating a LookAt: kml_lookat_longitude, kml_lookat_latitude, and kml_lookat_altitude.

KML Element Attribute	KML Element
kml_lookat_heading	<LookAt> <heading>
kml_lookat_tilt	<LookAt> <tilt>
kml_lookat_altitude	<LookAt> <altitude>
kml_lookat_range	<LookAt> <range>
kml_lookat_latitude	<LookAt> <latitude>
kml_lookat_longitude	<LookAt> <longitude>
kml_lookat_altitude_mode	<LookAt> <altitudeMode>
kml_gx_lookat_altitude_mode	<LookAt> <gx:altitudeMode>

TimeStamp/TimeSpan

Describes either a moment in time (TimeStamp), or a period of time (TimeSpan). Either a TimeStamp or a TimeSpan may be specified, but not both. The values can be in either the XML Schema time format

(<http://www.w3.org/TR/xmlschema-2/#isoformats>), or the FME DateTime format (http://www.fmeopedia.com/index.php/What_is_the_standard_format_for_dates_in_FME).

KML Element Attribute	KML Element
kml_timespan_begin	<TimeSpan> <begin>
kml_timespan_end	<TimeSpan> <end>
kml_timestamp_when	<TimeStamp> <when>
kml_gx_timespan_begin	<gx:TimeSpan> <begin>
kml_gx_timespan_end	<gx:TimeSpan> <end>
kml_gx_timestamp_when	<gx:TimeStamp> <when>

Region

<Region> elements can be added to any feature, and describe an area of interest defined by a bounding box, and either a altitude or a specified Level of Detail (LoD) extent. Regions are either active or inactive depending on whether or not their constraints are met. If active, the associated feature is displayed within Google Earth.

KML Element Attribute	KML Element
kml_latlonaltbox_altitude_mode	<LatLonAltBox> <altitudeMode>
kml_gx_latlonaltbox_altitude_mode	<LatLonAltBox> <gx:altitudeMode>
kml_latlonaltbox_min_altitude	<LatLonAltBox> <minAltitude>
kml_latlonaltbox_max_altitude	<LatLonAltBox> <maxAltitude>
kml_latlonaltbox_north	<LatLonAltBox> <north>
kml_latlonaltbox_south	<LatLonAltBox> <south>
kml_latlonaltbox_east	<LatLonAltBox> <east>
kml_latlonaltbox_west	<LatLonAltBox> <west>
kml_lod_min_lod_pixels	<Lod> <minLodPixels>
kml_lod_max_lod_pixels	<Lod> <maxLodPixels>
kml_lod_min_fade_extent	<Lod> <minFadeExtent>
kml_lod_max_fade_extent	<Lod> <maxFadeExtent>

Common Format Attributes

The following KML format attributes are common to most of the fixed schema feature types.

KML Format Attribute	Notes
kml_id	A unique string value that will be used by other elements/features to refer to this one.

KML Format Attribute	Notes
kml_parent	The id of the parent Folder element. Does not apply to Document elements.
kml_document	The name of the document to which this feature belongs. For Document features, this specifies the filename of the kml file for the document.
kml_tour	The id of the Tour to which the node belongs.
kml_animated_update	The id of the AnimatedUpdate element to which the node belongs.
kml_document_filename	See kml_document.
kml_attr_in_description	Overrides ATTR_IN_DESCRIPTION setting for the feature.
kml_html_descriptions	Overrides HTML_DESCRIPTIONS setting for the feature. If enabled, a CDATA wrapper will be added to the description, snippet, and balloon text elements.
kml_detect_rasters	Overrides DETECT_RASTERS setting for the feature.
kml_raster_mode	Overrides RASTER_MODE setting for the feature.
kml_raster_compression_level	Overrides RASTER_COMPRESSION_LEVEL setting for the feature.
kml_icon	Overrides INFORMATION_POINT_ICON setting for the feature.
kml_copy_icon	Overrides COPY_ICON setting for the feature.
kml_sort_value	Specifies the value to use for ordering the feature within the parent container.
kml_document_href	The href from which the parent document was retrieved.
kml_document_path	The local path to the parent document. This path may be in a temporary location, and may become invalid upon the completion of the translation. By default all temporary files are removed when the reader closes.

KML Format Attribute	Notes
kml_raw_text	Raw XML text that can be added to contents of the feature's element.
kml_update_mode	Specifies the update mode for the feature. Values can be "create", "change", or "delete".
kml_target_href	Overrides the TARGET_HREF setting.

Writer Notes

Creation of the <name> element

In most cases, the kml_name attribute is used to specify the name of the <Feature>. If the kml_name attribute is not present, the fme_text_string attribute, or the feature's id will be used.

Creation of the <description> element

There are several writer operations that affect how the description field is constructed.

If the FME attribute kml_description_text is present, its value will be used for the description field. I.e. no further processing will take place.

If the FME attribute kml_description is present, its value will be inserted into the description field.

If the "ATTR_IN_DESCRIPTION" preference is enabled, a table of the features user-defined attributes will be added to the description field.

To add raw HTML to the description, use the kml_description_raw_text format attribute.

If the "HTML_DESCRIPTIONS" preference is enabled, the entire description field will be wrapped in a CDATA block that facilitates the inclusion of HTML in a XML document.

Reader Notes

None.

NetworkLinkControl

KML Element Attribute	
kml_min_refresh_period	<minRefreshPeriod>
kml_max_session_length	<maxSessionLength>
kml_cookie	<cookie>
kml_message	<message>
kml_link_name	<linkName>
kml_link_description	<linkDescription>
kml_link_snippet	<linkSnippet>
kml_link_snippet_max_lines	@maxLines of <linkSnippet>
kml_expires	<expires>

Overview

<NetworkLinkControl> elements are used to control the behavior of files fetched by a <NetworkLink>. The <NetworkLinkControl> can change the behavior of the <NetworkLink>, update the contents of kml files downloaded prior, or both.

Writer Notes

None.

Reader Notes

None.

Document

KML Format Attribute	Notes
kml_document	The name of the document. This value will also be used for the document's file name; ".kml" will be appended if necessary.

Overview

A <Document> element is the root-level container element of a KML dataset that extends the abstract <Feature> element.

Note: The kml_id attribute does not set the filename of the kml file that contains the document; kml_id is only used to set the id attribute of the <Document> element.

Writer Notes

None.

Reader Notes

None.

Folder

KML Format Attribute	Notes
kml_sort_by_attribute	Specifies the name of the attribute whose value will be used to sort the contents of the folder. Has the same behavior as the KML21_SORT_BY_ATTRIBUTE defline parameter. The sort order can be overridden or augmented using the kml_sort_value format attribue on individual features.

Overview

A <Folder> element can contain any other abstract <Feature> element, including <Folder> elements, but excluding <Document> elements.

Writer Operations

None.

Reader Operations

None.

Placemark

KML Format Attribute	Remarks
kml_common_style	A unique string that identifies the common style. See Writer Operations for more information.
kml_icon	Either a name of an icon, or a path to an icon. This value is overridden by the value of the kml_icon_href attribute.
kml_create_info_point	Forces the creating of an aggregate containing the original feature geometry, and a point. Used for creating Placemarks that are clickable, but styled using an external <Style> element.
kml_target_style	Contains the id of an associated style element. Will be used to generate the appropriate <styleUrl> element.
kml_target_style_normal	Contains the ID of an associated style element that should be active when the placemark is not active. If specified, kml_target_style_highlight must also be specified. Will result in the creation of the appropriate <StyleMap> element.
kml_target_style_highlight	Contains the ID of an associated style element that should be active when the placemark is highlighted on mouse-over. If specified, kml_target_style_normal must also be specified. Will result in the creation of the appropriate <StyleMap> element.
kml_tour_stop_tour_name	The name of the tour for which a tour stop corresponding to this placemark should be generated.
kml_tour_stop_tour_duration	The total duration for the tour of which the tour stop corresponding to this placemark is a component.
kml_tour_stop_flyto_mode	The flyto mode to be used by FlyTo component of the tour stop associated with this Placemark.
kml_tour_stop_display_balloon	Whether or not to display the Placemark's balloon when the tour reaches the Placemarks tour stop. Valid values are yes, and no.
kml_tour_stop_delay_type	The type of delay for the tour stop corresponding to this Placemark.

KML Format Attribute	Remarks
	Valid values are <i>none</i> , <i>wait</i> , and <i>pause</i> .
kml_tour_stop_delay_duration	The duration of the tour stop's delay if the delay type is <i>wait</i> .
kml_tour_stop_view_perspective	<p>The perspective from which the tour stop (and Placemark) should be observed.</p> <p>If the value is <i>First Person</i>, the view will be from the precise coordinates of the tour stop location.</p> <p>If the value is <i>Third Person</i>, the view will be from a point orbiting the tour stop location.</p>
kml_tour_stop_view_range	<p>The distance, in meters, from the tour stop's view point to the tour stop location in the <i>Third Person</i> perspective.</p> <p>If the value is <calculate>, the range value will be calculated such that view show the tour stop, as well as a portion of the remaining tour. The calculated range is constant for every tour stop.</p>
kml_tour_stop_view_heading	The direction (azimuth) of the tour stop's view, in degrees, relative to north. If the value is <calculate>, the heading value will be calculated such the heading for the current stop is in the direction of the next stop.
kml_tour_stop_view_tilt	<p>The rotation, in degrees, of the tour stop's view around the X axis. A value of 0 indicates that the view is aimed straight down, and a value of 90 indicates that the view is aimed toward the horizon.</p> <p>Values greater than 90 only apply if the view perspective is <i>First Person</i>, and indicate that the view is pointed up into the sky. If the value is <calculate>, the tilt value will be calculated such the tilt for the current stop is in the direction of the next stop.</p>

Overview

Placemark elements contain the vector geometry that is displayed within Google Earth.

Geometry

Placemark elements contain an element that extends the <Geometry> element. The <Geometry> element is either a vector geometry or a 3D model. FME does not currently support reading 3D models via <Model> elements. <Model> elements can be written by adding a set of geometry traits to any Null geometry.

KML requires all vector geometry to use the LL84 coordinate system, and to have three dimensions. The KML writer will reproject input feature geometry to ensure that this constraint is fulfilled. Note: An error will occur if the KML Writer encounters a feature with no specified coordinate system.

Geometry Mapping

The KML writer only supports "classic" FME geometry. Features with "enhanced" geometry will be converted to classic geometry prior to output. The following table shows how classic FME Geometry Types are mapped to their corresponding KML Geometry elements.

FME Geometry Type	KML Geometry Element	Remarks
fme_point	<Point>	
fme_line	<LineString>	
fme_polygon	<Polygon>	
fme_donut	<Polygon>	Holes must have polygon geometry
fme_aggregate	<MultiGeometry>	
fme_arc	<LineString>	FME will stroke the arc.
fme_ellipse	<Polygon>	FME will stroke the ellipse.

Geometry Attributes

The following FME Attributes can be used to add sub-elements to <Geometry> elements.

Attribute	Element
kml_extrude	<extrude>
kml_tessellate	<tessellate>
kml_altitude_mode	<altitudeMode>
kml_gx_altitude_mode	<gx:altitudeMode>

Note: The KML reader will create geometry traits for the above attributes. The KMLwriter supports specifying the above attributes using either format attributes or geometry traits.

Model Geometry Traits

A <Model> element will be created for each FME Null geometry that contains the following geometry traits.

Trait	Child Elements of Model
kml_location_altitude	<Location><altitude>
kml_location_latitude	<Location><latitude>
kml_location_longitude	<Location><longitude>
kml_orientation_heading	<Orientation><heading>
kml_orientation_roll	<Orientation><roll>
kml_orientation_tilt	<Orientation><tilt>
kml_scale_x	<Scale><x>
kml_scale_y	<Scale><y>
kml_scale_z	<Scale><z>
kml_link_href	<Link><href>
kml_link_refresh_mode	<Link><refreshMode>
kml_link_refresh_interval	<Link><refreshInterval>

Trait	Child Elements of Model
kml_link_view_refresh_mode	<Link> <viewRefreshMode>
kml_link_view_refresh_time	<Link> <viewRefreshTime>
kml_link_view_bound_scale	<Link> <viewBoundScale>
kml_link_view_format	<Link> <viewFormat>
kml_link_view_http_query	<Link> <httpQuery>

Each <Model> element's <ResourceMap> element contains a series of <Alias> elements that remap the location of texture files for the model. Each <Alias> element contains a <targetHref>, and a <sourceHref> element.

Two delimited-value geometry traits are used to provide the list of target & source hrefs: kml_resourcemap_sources, and kml_resourcemap_targets. By default, comma-separated href lists are expected. The writer iterates over the source and target href lists in parallel to create the <sourceHref> and <targetHref> element pairs for each <Alias> element.

Writer Notes

Z Values

The KML specification requires all coordinate data to be 3D. Any input 2D features will be forced to 3D with a z-axis value of 0.0.

Note: All Z values are interpreted in meters, so you may need to manually convert between feet and meters.

Orientation

By default, no changes are made to the orientation of each feature's geometry. The ORIENTATION writer directive can be used to orient the geometry with either the right hand rule or the left hand rule. The orientation of the geometry will affect the appearance of extruded features.

Coordinate System Reprojection

The KML specification requires all coordinate data to use the LL84 coordinate system. All features will be reprojected to LL84 prior to output; if the writer encounters a feature that is not tagged with a coordinate system, the translation will terminate.

Styling

The writer supports the creation of "inline" styles. I.e. <Style> elements that are contained by the <Placemark> element. To add an inline style to a element, merely add any of the KML Element attributes for the Style element.

Information Point Icons

Information Point Icons are a mechanism that allows polygons to be "clickable" within the Google Earth interface. By default, polygons displayed in Google Earth are only selectable via the left-hand navigation tree. If, however, the polygon is part of an aggregate that in turn contains a point geometry element, Google Earth will display an icon that the user can select to pop up the description bubble.

The KML writer will create an information point icon if the feature has an icon specified. This can be done one of two ways:

- The icon name or path can be specified as a defline parameter (Feature Type Properties in Workbench)
- The icon name or path can be specified using the kml_icon attribute.

The KML writer supports two types of icons:

- Any icon that can be referenced via a path
- Well-known icons that reside in \$(FME_HOME)/icons. These icons can be referenced by name only. E.g. A1

Reader Notes

None.

ScreenOverlay

KML Element Attribute	KML Element
kml_overlay_color	<color>
kml_draw_order	<drawOrder>
kml_screenoverlay_rotation	<rotation>

Icon

Defines an image that is associated with the overlay. The kml_icon_href element attribute is used to explicitly specify the final location of the icon image file. As an alternative, the kml_icon format attribute can be used to specify the name or path of an icon that will be copied to the images folder of the KML dataset.

Note: If kml_icon_href is present on the feature, the value of kml_icon will be ignored.

KML Element Attribute	
kml_icon_href	<Icon><href>
kml_icon_refresh_mode	<Icon><refreshMode>
kml_icon_refresh_interval	<Icon><refreshInterval>
kml_icon_view_refresh_mode	<Icon><viewRefreshMode>
kml_icon_view_bound_scale	<Icon><viewBoundScale>
kml_icon_view_format	<Icon><viewFormat>
kml_icon_http_query	<Icon><httpQuery>

overlayXY

Specifies a point on (or outside of) the overlay image that is mapped to the screen coordinate (screenXY).

FME Attribute	overlayXY Attribute
kml_overlayxy_x	x
kml_overlayxy_y	y
kml_overlayxy_xunits	xunits
kml_overlayxy_yunitss	yunits

screenXY

Specifies a point relative to the screen origin that the overlay image is mapped to.

FME Attribute	screenXY Attribute
kml_screenxy_x	x
kml_screenxy_y	y
kml_screenxy_xunits	xunits
kml_screenxy_yunits	yunits

rotationXY

Specifies a point relative to the screen about which the screen overlay is rotated.

FME Attribute	rotationXY Attribute
kml_rotationxy_x	x
kml_rotationxy_y	y
kml_rotationxy_xunits	xunits
kml_rotationxy_yunits	yunits

size

Specifies the size of the image for the screen overlay.

FME Attribute	size Attribute
kml_size_x	x
kml_size_y	y
kml_size_xunits	xunits
kml_size_yunits	yunits

Writer Notes

None.

Reader Notes

None.

GroundOverlay

KML Element Attribute	KML Element
kml_overlay_color	<color>
kml_draw_order	<drawOrder>
kml_altitude	<altitude>
kml_altitude_mode	<altitudeMode>
kml_gx_altitude_mode	<gx:altitudeMode>

KML Format Attribute	Remarks
kml_raster_compression_level	An integer from 1 to 100, indicating the desired compression level.
kml_raster_format	The type of writer to use to write the raster. Either "tiff" or "jpeg". "jpeg" is the default.

FME Attribute	Remarks
fme_basename	The basename of the raster tile to write.

LatLonBox

Specifies where the top, bottom, right, and left sides of a bounding box for the ground overlay are aligned. Not required for writing rasters that are appropriately georeferenced.

KML Element Attribute	KML Element
kml_latlonbox_north	<LatLonBox> <north>
kml_latlonbox_south	<LatLonBox> <south>
kml_latlonbox_west	<LatLonBox> <west>
kml_latlonbox_east	<LatLonBox> <east>
kml_latlonbox_rotation	<LatLonBox> <rotation>

Icon

Defines an image that is associated with the overlay. The kml_icon_href element attribute is used to explicitly specify the final location of the icon image file. As an alternative, the kml_icon format attribute can be used to specify the name or path of an icon that will be copied to the images folder of the KML dataset.

Note: If kml_icon_href is present on the feature, the value of kml_icon will be ignored.

Attribute	Element
kml_icon_href	<Icon> <href>
kml_icon_refresh_mode	<Icon> <refreshMode>
kml_icon_refresh_interval	<Icon> <refreshInterval>
kml_icon_view_refresh_mode	<Icon> <viewRefreshMode>
kml_icon_view_bound_scale	<Icon> <viewBoundScale>
kml_icon_view_format	<Icon> <viewFormat>
kml_icon_http_query	<Icon> <httpQuery>

Overview

A <GroundOverlay> element must contain a <LatLonBox> element that defines the bounding box of the overlay. If the ground overlay feature has a raster geometry, these FME attributes will be automatically populated using the feature's bounding box.

Writer Notes

There are two separate and distinct methods to create elements:

Has Raster Geometry	Remarks
No	If the feature has no geometry, and has the Icon and LatLonBox

Has Raster Geometry	Remarks
	element attributes specified, the writer will directly write the <GroundOverlay> element without performing any further operations.
Yes	The writer can handle the raster geometry be either directly writing the raster geometry using either the GEOTIFF or JPEG writers, or the writer can merely copy or reference the original raster source file. Directly writing the raster geometry is preferable, and is the default behaviour.

Notes

If the “copy” or “reference” modes are used, it is very important to ensure that the associated raster tile has a JPEG or TIFF format, and uses a LL84 coordinate system.

If the (default) “write” mode is used, the feature will be reprojected to the LL84 coordinate system using a raster reprojection, and then written using a FME raster writer.

The KML writer does not attempt to pre-process raster tiles so that they are suitable for writing to the selected output format. It may be necessary to manipulate the raster’s bands or palettes prior to writing.

Reader Notes

None.

PhotoOverlay

KML Element Attribute	KML Element
kml_overlay_color	<color>
kml_draw_order	<drawOrder>
kml_photooverlay_rotation	<rotation>

Icon

Defines an image that is associated with the overlay. The kml_icon_href element attribute is used to explicitly specify the final location of the icon image file. As an alternative, the kml_icon format attribute can be used to specify the name or path of an icon that will be copied to the images folder of the KML dataset. Note: if kml_icon_href is present on the feature, the value of kml_icon will be ignored.

KML Element Attribute	Element
kml_icon_href	<Icon><href>
kml_icon_refresh_mode	<Icon><refreshMode>
kml_icon_refresh_interval	<Icon><refreshInterval>

KML Element Attribute	Element
kml_icon_view_refresh_mode	<Icon> <viewRefreshMode>
kml_icon_view_bound_scale	<Icon> <viewBoundScale>
kml_icon_view_format	<Icon> <viewFormat>
kml_icon_http_query	<Icon> <httpQuery>

ViewVolume

Defines how much of the current scene is visible.

KML Element Attribute	Element
kml_viewvolume_leftfov	<ViewVolume> <leftFov>
kml_viewvolume_rightfov	<ViewVolume> <rightFov>
kml_viewvolume_bottomfov	<ViewVolume> <bottomFov>
kml_viewvolume_topfov	<ViewVolume> <topFov>
kml_viewvolume_near	<ViewVolume> <near>

ImagePyramid

The image pyramid corresponding to a large original image.

KML Element Attribute	Element
kml_imagepyramid_tilesize	<ImagePyramid> <tileSize>
kml_imagepyramid_maxwidth	<ImagePyramid> <maxWidth>
kml_imagepyramid_minwidth	<ImagePyramid> <minWidth>
kml_imagepyramid_gridorigin	<ImagePyramid> <gridOrigin>

KML Format Attribute	Remarks
kml_raster_compression_level	An integer from 1 to 100, indicating the desired compression level.

Overview

The <PhotoOverlay> element allows you to geographically locate a photograph on the Earth and to specify its viewing parameters.

Writer Notes

None.

Reader Notes

None.

NetworkLink

KML Element Attribute	KML Element
kml_refresh_visibility	<refreshVisibility>
kml_fly_to_view	<flyToView>

Link

Specifies the location and loading parameters of an external resource.

KML Element Attribute	KML Element
kml_link_href	<Link> <href>
kml_link_refresh_mode	<Link> <refreshMode>
kml_link_refresh_interval	<Link> <refreshInterval>
kml_link_view_refresh_mode	<Link> <viewRefreshMode>
kml_link_view_refresh_time	<Link> <viewRefreshTime>
kml_link_view_bound_scale	<Link> <viewBoundScale>
kml_link_view_format	<Link> <viewFormat>
kml_link_http_query	<Link> <httpQuery>

Writer Operations

None.

Reader Operations

None.

Style

FME Attribute	Value	Remarks
fme_color	An FME color specification.	Will be converted to a kml_line-style_color attribute
fme_fill_color	An FME color specification	Will be converted to a kml_poly-style_color attribute
fme_pen_opacity	A float from 0.0 to 1.0	Will be converted to a kml_line-style_color attribute, if there is a corresponding fme_color attribute
fme_fill_opacity	A float from 0.0 to 1.0	Will be converted to a kml_poly-style_color attribute, if there is a corresponding fme_fill_color attribute

LabelStyle

Specifies how the <name> element if a feature is drawn in the 3D viewer.

KML Element Attribute	KML Element
kml_labelstyle_color	<LabelStyle><color>
kml_labelstyle_color_mode	<LabelStyle><colorMode>
kml_labelstyle_scale	<LabelStyle><scale>

KML Format Attribute	Remarks
kml_label_color	Specifies the label color using the fme color spec. Used to generate the RGB portion of the kml_labelstyle_color attribute.
kml_label_opacity	Specifies the label opacity using a float with a value from 0 to 1.0. Used to generate the Alpha portion of the kml_labelstyle_color attribute.

LineStyle

Specifies the drawing style for all line geometry, including polygon outlines.

KML Element Attribute	KML Element
kml_linestyle_color	<LineStyle><color>
kml_linestyle_color_mode	<LineStyle><colorMode>
kml_linestyle_width	<LineStyle><width>

FME Attribute	Remarks
fme_color	Specifies the line colour using the fme color spec. Used to generate the RGB portion of the kml_linestyle_color attribute.
fme_pen_opacity	Specifies the line opacity using a float with a value from 0 to 1.0. Used to generate the Alpha portion of the kml_linestyle_color attribute.

PolyStyle

Specifies the drawing style for all polygons, included line extrusions.

KML Element Attribute	KML Element
kml_polystyle_color	<PolyStyle><color>
kml_polystyle_color_mode	<PolyStyle><colorMode>
kml_polystyle_fill	<PolyStyle><fill>
kml_polystyle_outline	<PolyStyle><outline>

FME Attribute	Remarks
fme_fill_color	Specifies the fill colour using the fme color spec. Used to generate the RGB portion of the kml_polystyle_color attribute.
fme_fill_opacity	Specifies the fill opacity using a float with a value from 0 to 1.0. Used to generate the Alpha portion of the kml_polystyle_color attribute.

BalloonStyle

Specifies how the description balloon for placemarks is drawn.

KML Element Attribute	KML Element
kml_balloonstyle_bgcolor	<BalloonStyle> <bgColor>
kml_balloonstyle_text_color	<BalloonStyle> <textColor>
kml_balloonstyle_text	<BalloonStyle> <text>
kml_balloonstyle_raw_text	<BalloonStyle> <text> Element value written without xml entity encoding.

KML Format Attribute	Remarks
kml_balloon_color	Specifies the balloon colour using the fme color spec. Used to generate the RGB portion of the kml_balloonstyle_bgcolor attribute.
kml_balloon_opacity	Specifies the balloon opacity using float with a value from 0 to 1.0. Used to generate the Alpha portion of the kml_balloonstyle_bgcolor attribute.
kml_balloon_text_color	Specifies the balloon text colour using the fme color spec. Used to generate the RGB portion of the kml_balloonstyle_text_color attribute.
kml_balloon_text_opacity	Specifies the balloon text opacity using float with a value from 0 to 1.0. Used to generate the Alpha portion of the kml_balloonstyle_text_color attribute.

ListStyle

Specifies how a feature is displayed in the list view.

KML Element Attribute	KML Element
kml_liststyle_bgcolor	<ListStyle> <bgColor>
kml_liststyle_list_item_type	<ListStyle> <listItemType>

ItemIcon

Specifies the icon used in the list view.

KML Element Attribute	KML Element
kml_liststyle_item_icon{}.state	<ItemIcon> <state>
kml_liststyle_item_icon{}.href	<ItemIcon> <href>

A <ListStyle> element can contain 0 or more <ItemIcon> elements. The FME kml_liststyle_item_icon{} list, which has two sub-elements 'state' and 'href', maps to N <ItemIcon> elements.

IconStyle

Specifies how the icons for point placemarks are drawn. This also applies to multi geometry placemarks with a point component.

KML Element Attribute	KML Element
kml_iconstyle_color	<IconStyle> <color>
kml_iconstyle_color_mode	<IconStyle> <colorMode>
kml_iconstyle_scale	<IconStyle> <scale>
kml_iconstyle_heading	<IconStyle> <heading>

KML Format Attribute	Remarks
kml_icon_color	Specifies the icon colour using the fme color spec. Used to generate the RGB portion of the kml_iconstyle_color attribute.
kml_icon_opacity	Specifies the icon opacity using float with a value from 0 to 1.0. Used to generate the Alpha portion of the kml_iconstyle_color attribute.

Icon

Defines an image that is associated with the style. The kml_icon_href element attribute is used to explicitly specify the final location of the icon image file. As an alternative, the kml_icon format attribute can be used to specify the name or path of an icon that will be copied to the images folder of the KML dataset.

Note: If kml_icon_href is present on the feature, the value of kml_icon will be ignored.

Attribute	Element
kml_icon_href	<Icon> <href> Required.

Attribute	Element
kml_icon_refresh_mode	<Icon> <refreshMode>
kml_icon_refresh_interval	<Icon> <refreshInterval>
kml_icon_view_refresh_mode	<Icon> <viewRefreshMode>
kml_icon_view_bound_scale	<Icon> <viewBoundScale>
kml_icon_view_format	<Icon> <viewFormat>
kml_icon_http_query	<Icon> <httpQuery>

hotspot

Specifies the point within the Icon that is anchored to the "Point" specified within the Placemark.

KML Element Attribute	hotspot Element Attribute
kml_hotspot_x	<hotSpot @x>
kml_hotspot_y	<hotSpot @y>
kml_hotspot_xunits	<hotSpot @xunits>
kml_hotspot_yunits	<hotSpot @yunits>

Writer Notes

As noted above, FME color & opacity attributes will be converted to KML color attributes.

Reader Notes

None.

StyleMap

KML Element Attribute	KML Element
kml_style_url_normal	<Pair> <styleUrl> where <key> is 'normal'
kml_style_url_highlight	<Pair> <styleUrl> where <key> is 'highlight'

Overview

A <StyleMap> element maps between two different styles. Typically used to specify the styling for the normal and highlighted states of a Placemark.

Writer Notes

None.

Reader Notes

None.

Tour

The Tour element has no specific attributes; the common format and element attributes listed at the beginning of the section do apply.

Overview

<gx:Tour> elements are used by Google Earth to provide users with a guided tour of a KML dataset. Each Tour has a playlist of one or more elements that are sub-types <TourPrimitive>; these include <AnimatedUpdate>, <FlyTo>, <SoundCue>, <TourControl>, and <Wait>. Tour primitives are routed to their parent Tour via the kml_tour format attribute.

Writer Notes

If a tour primitive does not specify the id of a tour, it will be assigned to a 'default' tour that has the id 'fme_default_tour', and the name "Tour".

Reader Notes

None.

AnimatedUpdate

KML Element Attribute	KML Element
kml_gx_duration	<gx:duration>

Overview

<AnimatedUpdate> elements create temporary changes to a KML dataset via a child <Update>. AnimatedUpdates work very similarly to the updates within a NetworkLinkControl, with the exception that changes made by an AnimatedUpdate revert upon completion of the parent tour.

Writer Notes

To route a node to a particular AnimatedUpdate for update purposes, the following three format attributes must be specified:

- kml_tour: The id of the parent AnimatedUpdate's tour
- kml_animated_update: The id of the parent AnimatedUpdate
- kml_update_mode: The type of update to create

Reader Notes

The KML reader does not currently support reading <AnimatedUpdate> elements.

FlyTo

KML Element Attribute	KML Element
kml_gx_duration	<gx:duration>
kml_gx_flyto_mode	<gx:flyToMode>

Overview

<FlyTo> elements are used to guide a tour to a particular viewpoint, usually looking at an existing Placemark or some other geographic feature. The viewpoint is specified using an abstract view. I.e. either a <LookAt> or a <Camera> element. For the element attributes corresponding to <LookAt> and <Camera> please refer to the prior sections.

Writer Notes

None.

Reader Notes

The KML reader does not currently support reading <FlyTo> elements.

SoundCue

KML Element Attribute	KML Element
kml_href	<href>

Overview

<SoundCue> elements are used to specify a particular sound file that should play during the tour.

Writer Notes

None.

Reader Notes

The KML reader does not currently support reading <SoundCue> elements.

TourControl

KML Element Attribute	KML Element
kml_gx_play_mode	<gx:playMode>

Overview

<TourControl> elements are used to change the Tour's play mode. Currently, the only available mode is 'pause', but that may change in the future.

Writer Notes

None.

Reader Notes

The KML reader does not currently support reading <TourControl> elements.

Wait

KML Element Attribute	KML Element
kml_gx_duration	<gx:duration>

Overview

<Wait> elements introduce an arbitrary delay in the Tour prior to advancing to the next Tour Primitive.

Writer Notes

None.

Reader Notes

The KML reader does not currently support reading <Wait> elements.

GPS eXchange Format (GPX) Reader

Format Notes: This format is not supported by FME Base Edition.

Overview

GPX (the GPS Exchange Format) is a lightweight XML data format for the interchange of GPS data (waypoints, routes, and tracks) between applications and Web services on the Internet. FME is capable of reading both GPX 1.0 and GPX 1.1 and writing GPX 1.1.

When reading GPX 1.0, the reader returns Bounds, Waypoint, Route, Routepoint, Track and Trackpoint features. When reading GPX 1.1, the reader returns Metadata, Waypoint, Route, and Track features.

Note: If you are using FME 2008, the GPX reader now reads GPX Route and Track elements differently.

For more information, go to:

<http://www.topografix.com/gpx.asp>

GPX Quick Facts

Format Type Identifier	GPX
Reader/Writer	Reader
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	Metadata, Waypoint, Route, Track
Typical File Extensions	.gpx .xml
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	xml_type

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	yes	point	yes
circles	no	polygon	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
circular arc	no		raster	no
donut polygon	no		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	yes		z values	no
none	yes			

Reader Overview

The GPX reader supports reading GPX 1.0 and 1.1 datasets.

The GPX reader has changed from the official FME 2008 release. The GPX reader now generates FME features from GPX 1.1 files with the format specified in the "Schema Overview" section below. It returns Metadata, Waypoint, Route, and Track features.

The GPX 1.1 reader released with FME 2008 generated FME features of types Metadata, Waypoint, Route, Routepoint, Track, and Trackpoint. The FME 2008 reader is deprecated, but is still available for backward compatibility. To use the 2008 reader, the source dataset's settings dialog "Reader Mode" option should be set to "Backward-Compatibility(FME2008)".

Coordinate Systems

The GPX reader supports data in decimal degrees (WGS84 datum).

Reader Directives

The suffixes shown are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the GPX reader is `GPX`.

READER_MODE

Required/Optional: Optional

Specifies how GPX elements are read into FME features. Backward compatibility mode will read Trackpoints and Route-points in as features, as well as Tracks and Routes. Normal mode will only read in Track and Route features, storing the point information as traits of the features' geometries, as specified in the schema overview. The default value is Normal for new workspaces, but backward compatibility mode will be used if the keyword value is not present.

Examples:

```
GPX_READER_MODE Normal
GPX_READER_MODE Backward-Compatibility(FME2008)
```

Workbench Parameter: *Reader Mode*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Schema Overview

Fixed Schema and Feature Representation

The GPX reader supports a fixed schema.

The reader generates FME features with the same schema that the writer accepts for writing. The GPX elements and their corresponding FME feature representations are mapped by thereader according to the schemas in the following sections.

Feature Types

- Metadata
- Waypoint
- Routepoint and Trackpoint
- Route
- Track

Metadata

To write a <metadata> tag, pass a feature of the following form to the writer.

Feature Type: Metadata

```
<metadata>
```

feature's attributes as xml tags

```
</metadata>
```

Feature Type: Metadata

Geometry type: any (geometry is ignored by writer)

Feature Attribute	GPX XML Entity
name	<name>
description	<desc>
author_name	<author> <name>
author_email	<author> <email>
author_link_text	<author> <link> <text>
author_link_type	<author> <link> <type>

Feature Attribute	GPX XML Entity
author_link_href	<author> <link href="">
copyright_year	<copyright> <year>
copyright_license	<copyright> <license>
copyright_author	<copyright author="">
link_text	<link> <text>
link_type	<link> <type>
link_href	<link href="">
creation_time	<time>
keywords	<keywords>

Waypoint

To write a <wpt> tag, pass a feature of the following form to the writer.

```
<wpt>
```

feature's attributes as xml tags

```
</wpt>
```

Feature Type: Waypoint

Geometry type: IFMEPoint

Feature Attribute	GPX XML Entity
elevation	<ele>
creation_time	<time>
magnetic_variation	<magvar>
geoid_height	<geoidheight>
comment	<cmt>
description	<src>
link_text	<link> <type>
link_type	<copyright> <year>
link_href	<link href="">
symbol	<sym>
type	<type>
gps_fix_type	<fix>
number_of_satellites	<sat>

Feature Attribute	GPX XML Entity
hdop	<hdop>
vdop	<vdop>
pdop	<pdop>
age_of_dgps_data	<ageofdgpsdata>
dgps_id	<dgpsid>

Routepoint and Trackpoint

Routepoints and Trackpoints are not created by the reader as FME features.

Instead, Routes and Tracksegments are constructed as IFMELine features, and their Routepoints and Trackpoints are represented by x-y paired coordinates in the IFMELine feature's geometry. A point's extra information is stored in the IFMELine feature's geometry traits as a list trait of the form *type{index}.name*.

type is either 'Routepoint' or 'Trackpoint'.

index is the numeric index of the point in the IFMELine's geometry.

name is an FME name that corresponds to an XML entity name

The *names* are the same as Waypoint's feature attribute names.

Geometry type: (implicit) IFMELine coordinate

Route

Note: GPX writing may not be applicable to your FME license.

Refer to the Routepoint and Trackpoint section for information on writing Routepoints with more than just longitude and latitude.

<rte>

feature's attributes as xml tags

<rtept lon="" lat="">

feature's geometry traits of name Routepoint{index}.name as xml tags

</rtept>

</rte>

Feature Type: Route

Geometry type: IFMELine

To write an <rte> tag, pass a feature of the following form to the writer:

Feature Attribute	GPX XML Entity
name	<name>
comment	<cmt>
description	<desc>
source	<src>

Feature Attribute	GPX XML Entity
link_text	<link><text>
link_type	<link><type>
link_href	<link href="">
number	<number>
type	<type>

Track

Note: GPX writing may not be applicable to your FME license.

Refer to the Routepoint and Trackpoint section for information on writing Trackpoints with more than just longitude and latitude.

```
<trk>
```

```
<trkpt lon="" lat="">
```

feature's geometry traits of name Trackpoint{index}.name as xml tags

```
</trkpt>
```

```
</trk>
```

Feature Type: Track

Geometry type: IFMEAgregate of IFMELines

IBM DB2 Reader/Writer

FME's DB2 Database (Attributes only) reader and writer modules (called *DB2 Reader/Writer* throughout the rest of this chapter) provide the Feature Manipulation Engine (FME) with access to attribute data held in IBM's DB2 database tables.

Overview

This data may or may not have a spatial component to it. Thus DB2 reader can read from DB2 databases which may or may not be spatially enabled. The FME provides read and write access to live databases accessible via DB2 CLI.

Tip: See the @SQL function in the Functions and Factories manual. This function allows arbitrary Structured Query Language (SQL) statements to be executed against any database.

DB2 Database Quick Facts

Format Type Identifier	DB2
Reader/Writer	Both
Licensing Level	Professional
Dependencies	None
Dataset Type	Data source name
Feature Type	Table name
Typical File Extensions	N/A
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	Yes
Geometry Type	db2_type

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	point	no
circles	no	polygon	no
circular arc	no	raster	no
donut polygon	no	solid	no
elliptical arc	no	surface	no
ellipses	no	text	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
line	no		z values	n/a
none	yes			

Reader Overview

FME considers a DB2 dataset to be a collection of relational tables. The tables must be defined in the mapping file before they can be read. Arbitrary **WHERE** clauses and joins are fully supported.

Reader Directives

The suffixes listed are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the DB2 reader is `DB2`.

DATASET

Required/Optional: *Required*

This is the data source name similar to ODBC data source name.

Example:

```
DB2_DATASET sample
```

Workbench Parameter: *Source IBM DB2 Non-spatial Dataset*

USER_NAME

Required/Optional: *Optional*

The name of the user who will access the database. By default, `USER_NAME` will be considered the same as the schema name. e.g. any table name which explicitly does not have the schema name prefixed will be considered as a table from the schema for that user.

Example:

```
DB2_USER_NAME bond007
```

Workbench Parameter: *User Name*

PASSWORD

Required/Optional: *Optional*

The password to access the database.

Example:

```
DB2_PASSWORD moneypenny
```

Workbench Parameter: *Password*

DEF

Required/Optional: *Optional*

Each database table must be defined before it can be read. There are two forms that the definition may take.

The syntax of the first form is:

```
DB2_DEF <tableName> \
  [SQL_WHERE_CLAUSE <whereClause>] \
  [<fieldName> <fieldType>] +
```

In this form, the fields and their types are listed. The `<fieldType>` of each field must be given, but it is not verified against the database definition for the field. In effect, it is ignored.

The `<tableName>` must match a table in the database. This will be used as the feature type of all the features read from the table.

If no `<whereClause>` is specified, all rows in the table will be read and returned as individual features, unless limited by a global directive:

```
<ReaderKeyword>_WHERE_CLAUSE
```

If a `<whereClause>` is specified, only those rows that are selected by the clause will be read. Note that the `<whereClause>` does not include the word "WHERE."

In this example, the all records whose ID is less than 5 will be read from the supplier table:

```
DB2_DEF supplier \  
  SQL_WHERE_CLAUSE "id < 5" \  
  ID integer \  
  NAME char(100) \  
  CITY char(50)
```

The syntax of the second form is:

```
DB2_DEF <tableName> \  
  SQL_STATEMENT <sqlStatement>
```

In this form, an arbitrary complete `<sqlStatement>` will be executed. The statement is passed untouched to the database (and therefore may include non-portable database constructions). The results of the statement will be returned, one row at a time, as features to FME. This form allows the results of complex joins to be returned to FME.

Note:

If the table has a column of type BIGINT then use the DB2's CHAR() function to convert it to a string. This also applies when an arbitrary SQL statement is passed to FME using @SQL() function or the SQLExecutor transformer in Workbench. For example,

```
SELECT CHAR(myBigIntColumn), myID FROM myTable
```

All features will be given the feature type `<tableName>`, even though they may not necessarily have come from that particular table. Indeed, with this form, the `<tableName>` need not exist as a separate table in the database.

In this example, the results of joining the `employee` and `city` tables are returned. All attributes from the two tables will be present on each returned feature. The feature type will be set to `complex`.

```
DB2_DEF complex \  
  SQL_STATEMENT \  
    "SELECT * FROM EMPLOYEE, CITY WHERE EMPLOYEE.CITY = CITY.NAME"
```

WHERE_CLAUSE

Required/Optional: *Optional*

This optional specification is used to limit the rows read by the reader from each table. If a given table has no `SQL_WHERE_CLAUSE` or `SQL_STATEMENT` specified in its `DEF` line, the global `<ReaderKeyword>_WHERE_CLAUSE` value, if present, will be applied as the `WHERE` specifier of the query used to generate the results. If a table's `DEF` line does contain its own `SQL_WHERE_CLAUSE` or `SQL_STATEMENT`, it will override the global `WHERE` clause.

The syntax for this clause is:

```
DB2_WHERE_CLAUSE <whereClause>
```

Note that the `<whereClause>` does not include the word "WHERE."

The example below selects only the features whose lengths are more than 2000:

```
DB2_WHERE_CLAUSE LENGTH > 2000
```

Workbench Parameter: *WHERE Clause*

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined database table files that will be read. If no **IDs** are specified, then all defined and available tables are read. The syntax of the **IDs** keyword is:

```
DB2_IDS <featureType1> \
<featureType2> ... \
<featureTypeN>
```

The feature types must match those used in **DEF** lines.

The example below selects only the **HISTORY** table for input during a translation:

```
DB2_IDS HISTORY
```

Workbench Parameter: *Feature Types to Read*

RETRIEVE_ALL_SCHEMAS

Required/Optional: *Optional*

This directive is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

This optional directive is used to tell the reader to retrieve the names and the schemas of all the tables in the source database. If this value is not specified, it is assumed to be "No".

The syntax of the **RETRIEVE_ALL_SCHEMAS** directive is:

```
DB2_RETRIEVE_ALL_SCHEMAS Yes
```

RETRIEVE_ALL_TABLE_NAMES

Required/Optional: *Optional*

This directive is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

Similar to **RETRIEVE_ALL_SCHEMAS**; this optional directive is used to tell the reader to only retrieve the table names of all the tables in the source database. If **RETRIEVE_ALL_SCHEMAS** is also set to "Yes", then **RETRIEVE_ALL_SCHEMAS** will take precedence. If this value is not specified, it is assumed to be "No".

The syntax of the **RETRIEVE_ALL_TABLE_NAMES** directive is:

```
DB2_RETRIEVE_ALL_TABLE_NAMES Yes
```

PERSISTENT_CONNECTION

A user may want to keep a connection to a database for reuse during a particular FME session. For example, when running a batch of 100 mapping files on the same database connection, it may be desirable to keep a connection open and save the processing time required to make and break a database connection.

A database connection will be determined to be the same when the database name, the username, the password, and the transaction interval are the same.

Values: *YES* | *NO*

Default value: *NO*

Example:

```
DB2_PERSISTENT_CONNECTION YES
```

Workbench Parameter: *Persistent Connection*

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The Database writer module stores attribute records into a live relational database. The Database writer provides the following capabilities:

- **Transaction Support:** The Database writer provides transaction support that eases the data loading process. Occasionally, a data load operation terminates prematurely due to data difficulties. The transaction support provides a mechanism for reloading corrected data without data loss or duplication.
- **Table Creation:** The Database writer uses the information within the FME mapping file to automatically create database tables as needed.
- **Bulk Loading:** The Database writer uses a bulk loading technique to ensure speedy data load only when there are no LOB (BLOBs or CLOBs) columns in the table. The performance vastly exceeds a one-insert-at-a-time approach.

Writer Directives

The directives processed by the DB2 Writer are listed below. The suffixes shown are prefixed by the current **<writerKeyword>** in a mapping file. By default, the **<writerKeyword>** for the DB2 writer is **DB2**.

DATASET, USER_NAME, PASSWORD

The **DATASET**, **USER_NAME**, and **PASSWORD** directives operate in the same manner as they do for the DB2 reader. The remaining writer-specific directives are discussed in the following sections.

ABORT_ON_BAD_DATA

Required/Optional: *Optional*

Some features may contain out-of-range or invalid attribute values. These features will be rejected and cannot be written to the database. If the value of this directive is YES then the translation will be aborted immediately after encountering such a problem. If this directive is set to NO then the translation will continue but the features with rejected feature will not be written to the database.

Values: *YES* | *NO*

Default: *NO*

Example:

DB2_ABORT_ON_BAD_DATA YES

Workbench Parameter: *Abort Translation on Bad Data*

DEF

Required/Optional: *Optional*

Each database table must be defined before it can be written. For the DB2 writer, only one form of the **DEF** line is used:

```
DB2_DEF <tableName> \  
  [db2_overwrite_table (YES|NO|TRUNCATE)] \  
  [<fieldName> <fieldType>] +
```

In this form, the fields and their types are listed. If the table already exists in the database, and `db2_overwrite_table` is not specified with a parameter of YES, FME will append its information to the existing database table. In this case, it is not necessary to list the fields and their types – FME will use the schema information in the database to determine this. If the fields and types are listed, they must match those in the database. However, not all fields must be listed.

If the table does not exist, or `db2_overwrite_table` is specified with a value of YES, then the field names and types are used to first create the table. In any case, if a `<fieldType>` is given, it may be any field type supported by the target database.

This example defines the **SUPPLIER** table for the FME. If the table did not exist, it will be created just before the first **SUPPLIER** row is written. If the table already exists, the data will be appended to the existing table.

```
DB2_DEF SUPPLIER \  
  ID integer \  
  NAME char(100) \  
  CITY char(50)
```

The following example is exactly the same, except that it replaces any existing table named **SUPPLIER** with a new table having the specified definition. If the table **SUPPLIER** does not exist in the database, then a new table is simply created.

```
DB2_DEF SUPPLIER \  
  db2_overwrite_table YES \  
  ID integer \  
  NAME char(100) \  
  CITY char(50)
```

In the following example, the definition line only makes the pre-existing **EMPLOYEE** table known to FME:

```
DB2_DEF EMPLOYEE
```

Features may later be routed to this table.

PERSISTENT_CONNECTION

A user may want to keep a connection to a database for reuse during a particular FME session. For example, when running a batch of 100 mapping files on the same database connection, it may be desirable to keep a connection open and save the processing time required to make and break a database connection.

A database connection will be determined to be the same when the database name, the username, the password, and the transaction interval are the same.

Values: YES | NO

Default value: NO

Example:

```
DB2_PERSISTENT_CONNECTION YES
```

Workbench Parameter: *Persistent Connection*

TRANSACTION_INTERVAL

This statement informs FME about the number of features to be placed in each transaction before a transaction is committed to the database.

If the DB2_TRANSACTION_INTERVAL statement is not specified, then a value of 1000 is used as the transaction interval.

Parameter	Contents
<transaction_interval>	The number of features in a single transaction.

Example:

```
DB2_TRANSACTION_INTERVAL 5000
```

Workbench Parameter: *Transaction Interval*

Feature Representation

Features read from a DB2 database consist of a series of attribute values. They have no geometry. The attribute names are as defined in the **DEF** line if the first form of the **DEF** line was used. If the second form of the **DEF** line was used, then the attribute names are as they are returned by the query, and as such may have their original table names as qualifiers. The feature type of each DB2 feature is as defined on its **DEF** line.

Features written to the database have the destination table as their feature type, and attributes as defined on the **DEF** line.

DATE, TIME and DATETIME Fields

When a **DATE**, **TIME** or **TIMESTAMP** field is read by the DB2 reader, two attributes are set in the FME feature. The first attribute has the name of the database column, and its value is of the form **YYYYMMDD** or **HHMMSS**. This is compatible with all other FME date and time values.

The second attribute has a suffix of **.full** and is of the form **YYYYMMDDHHMMSS**. It specifies the date and the time, with the time portion specified using the 24-hour clock.

For example, if a date field called **UPDATE_DATE** is read, the following attributes will be set in the retrieved FME feature:

```
UPDATE_DATE = '19980820'  
UPDATE_DATE.full = '19980820000000'
```

The DB2 writer looks for both attributes when a **date** or **datetime** column is being output. Either may be specified. If both attributes are specified, then the value specified in **UPDATE_DATE.full** is used to populate the **DATE** or **DATETIME** portion of the date; otherwise, this portion is set to 0.

Using DEF Lines to Read from an ODBC Datasource

This example illustrates how the two forms of the **DEF** lines can be used to read from an ODBC database source, which is named **rogers**.

```
READER_TYPE DB2  
DB2_DATASET sampledb  
DB2_USER_NAME <userName>  
DB2_PASSWORD <password>
```

```
# Form 1 of the DEF line is used like this -- it reads just
```

```
# the two fields we list and applies the where clause

DB2_DEF supplier \
  db2_where_clause "id < 5" \
  ID integer \
  CITY char(50)

# Form 2 of the DEF line is used like this -- we let SQL
# figure out what fields we want and do a complex join
# involving 3 tables. The FME features will have whatever
# fields are relevant. The "feature type" as far as
# FME is concerned is whatever was put on the DEF line.
# In this case "complex" is the feature type, even though no
# table named "complex" is present in the database.

DB2_DEF complex \
  db2_sql "SELECT CUSTOMER.NAME, CUSTOMER.ID,
  VIDEOS.ID, VIDEOS.TITLE FROM RENTALS, CUSTOMER,
  VIDEOS WHERE RENTALS.customerID = CUSTOMER.ID AND
  VIDEOS.ID = RENTALS.videoID AND CUSTOMER.ID = 1"

# Finally, define the NULL writer as our output -- we will
# just log everything we read to the log file for inspection.

WRITER_TYPE NULL
NULL_DATASET null

FACTORY_DEF * SamplingFactory \
  INPUT FEATURE_TYPE * @Log()
```

IBM DB2 Spatial Reader/Writer

This format is not supported by FME Base Edition.

Object writing is available only with FME DB2 Edition.

DB2 7.2 and 8.1 are currently supported.

DB2 Spatial currently provides support for 2D geometries only. For 3D and 3D with Measures support, please contact Safe Software.

FME's DB2 Spatial Reader/Writer module(referred to as DB2 Spatial in this chapter) enables FME to read spatial and attribute data from IBM's DB2 database, and write spatial and attribute data to the existing database.

Overview

DB2 Spatial can read from databases which are spatially enabled but the tables may or may not have spatial information stored. This module communicates directly with DB2 using CLI for maximum throughput.

Note: DB2 7.2 and 8.1 are currently supported. DB2 Spatial currently provides support for 2D geometries only. For 3D and 3D with Measures support, please contact Safe Software.

This section assumes familiarity with IBM DB2 Spatial Extender, the geometry types it supports, and its indexing mechanisms.

Tip:

See the QueryFactory in the FME Functions and Factories manual. This factory also exploits the powerful query capabilities of DB2 Spatial.

See the @SQL function, also in the FME Functions and Factories manual. This function allows arbitrary Structured Query Language (SQL) statements to be executed against any DB2 database.

DB2 Spatial Quick Facts

Format Type Identifier	DB2
Reader/Writer	Both
Licensing Level	Reading: Professional Object Writing: DB2 Edition
Dependencies	None
Dataset Type	Data source name
Feature Type	Table name
Typical File Extensions	N/A
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	Yes
Geometry Type	db2_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	yes		z values	n/a
none	yes			

Reader Overview

The FME considers a DB2 Spatial dataset to be a database containing a collection of relational tables together with their geometry. The tables to be read may be defined in the mapping file. If no tables are specified, then all tables are read. Arbitrary **WHERE** clauses and joins are fully supported. An entire arbitrary SQL **SELECT** statement may also be used as a source of results.

Reader Directives

The suffixes listed are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the DB2 Spatial reader is `DB2SPATIAL`.

DATASET

Required/Optional: *Required*

This specifies the data source name for the DB2 Spatial database. The data source name must have been set up in the Client Configuration Assistant or on the command line.

Example:

```
DB2SPATIAL_DATASET citySource
```

Workbench Parameter: *Source IBM DB2 Spatial Dataset*

USER_NAME

Required/Optional: *Required*

The name of the user who will access the database.

Example:

```
DB2SPATIAL_USER_NAME shadow
```

Workbench Parameter: *User Name*

PASSWORD

Required/Optional: *Required*

The password to access the database.

Example:

```
DB2SPATIAL_PASSWORD puppy
```

Workbench Parameter: *Password*

DEF

Required/Optional: *Optional*

The syntax of the definition is:

```
DB2SPATIAL_DEF <tableName> \  
    [db2_type <type>] \  
    [db2_envelope_minx <xmin>] \  
    [db2_envelope_miny <ymin>] \  
    [db2_envelope_maxx <xmax>] \  
    [db2_envelope_maxy <ymax>] \  
    [db2_spatial_predicate <spatialPredicate>] \  
    [db2_predicate_result <predicateResult>] \  
    [db2_where_clause <whereClause>] \  
    [db2_sql <sqlQuery>] \  
    [<fieldName> <fieldType>] +
```

The `<fieldType>` of each field must be given, but it is not verified against the database definition for the field. In effect, it is ignored.

The column(s) which has geometry should not be specified on the `DEF` line. In case a feature type has more than one registered geometry column or layer, than DB2 Spatial Reader module will arbitrarily choose one as the primary geometry column and consider the other(s) as attribute columns.

The **<tableName>** can be either fully qualified or not. A fully qualified table name consist of two parts separated by a period (.). The first part is the **<schema name>** and second part is the **<table name>**. The **<table name>** part must match a table in the schema specified by the **<schema name>** part of the **<tableName>**. If a schema name is not provided as part of the table name, then the username will be considered the schema name. This will be used as the feature type of all the features read from the table. For example, if a user wants to read a table from its own schema then only the table name can be provided, but if the user wants to read from a different user's schema, then table name should be qualified with schema name.

The definition allows specification of separate search parameters for each table. If any of the configuration parameters are given, they will override, for that table, whatever global values have been specified by the reader directives listed above. If any of these parameters is not specified, the global values will be used.

The following table summarizes the definition line configuration parameters:

Parameter	Contents
db2_type	This specifies the type of geometry the features to be read from the layer will have.
db2_geometry_column	This specifies the spatial layer or geometry column to use for reading spatial data in case the table has multiple geometry/spatial columns.
db2_envelope_minx db2_envelope_miny db2_envelope_maxx db2_envelope_maxy	These specify the spatial extent of the features to be read from the layer. If these are not all specified, the values from the <ReaderKeyword>_SEARCH_ENVELOPE directive are used.
db2_spatial_predicate	This specifies the spatial predicate to be tested for this layer. Its default value is set to INTERSECTS. Note: This DEF line option is valid only if there is a valid spatial envelope specified by db2_envelope_minx, db2_envelope_miny, db2_envelope_maxx and db2_envelope_maxy.
db2_predicate_result	This specifies the result to be used for the Spatial predicate specified in db2_spatial_predicate option.
db2_where_clause	This specifies the SQL WHERE clause applied to the attributes of the layer's features to limit the set of features returned. If this is not specified, the value of the <ReaderKeyword>_WHERE_CLAUSE directive is used.
db2_sql	This specifies an SQL SELECT query to be used as the source for the results. If this is specified, the DB2 Spatial reader will execute the query, and use the resulting rows as the features instead of reading from the table <layerName>. All returned features will have a feature type of <layerName>, and attributes for all columns selected by the query. The db2_where_clause and all parameters which specify a spatial constraint – db2_envelope_minx, db2_interaction, and so on – are ignored if db2_sql is supplied.

If no `<whereClause>` is specified, all rows in the table will be read and returned as individual features. If a `<whereClause>` is specified, only those rows that are selected by the clause will be read. Note that the `<whereClause>` does not include the word "where".

The `db2_sql` parameter allows a user to specify an arbitrary SQL `SELECT` query. If this is specified, FME will execute the query, and use each row of data returned from the query to define a feature. Each of these features will be given the feature type named in the `DEF` line, and will contain attributes for every column returned by the `SELECT`. In this case, all `DEF` line parameters regarding a `WHERE` clause or spatial querying is ignored, as it is possible to embed this information directly in the text of the `<sqlQuery>`.

The following example joins the tables `ROADS` and `ROADNAMES`, placing the resulting data into FME features with a feature type of `MYROADS`. Imagine that `ROADS` defines the geometry for the roads, and has a numeric field named `ID`, and that `ROADNAMES` joins the numeric field `ID` with character arrays with the roads' names.

```
DB2SPATIAL_DEF MYROADS \
  db2_sql "SELECT * FROM ROADS, \
          ROADNAMES WHERE ROADS.ID = ROADNAMES.ID"
```

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined database tables files that will be read. If no `IDs` are specified, then all defined and available tables are read. The syntax of the `IDs` directive is:

```
DB2SPATIAL_IDS <featureType1> \
  <featureType2> \
  <featureTypeN>
```

The feature types must match those used in `DEF` lines.

The example below selects only the `ROADS` table for input during a translation:

```
DB2SPATIAL_IDS ROADS
```

Workbench Parameter: *Feature Types to Read*

SIMPLIFY_AGGREGATES

Required/Optional: *Optional*

This directive specifies whether multi-geometry or aggregate features with one member are read as stored or simplified and read as single member. e.g. an aggregate of points or multipoint features with only one point will be returned as a simple point if the value of this directive is `YES`.

Values: `YES` | `NO`

Default value: `NO`

Example:

The syntax of the `DB2SPATIAL_SIMPLIFY_AGGREGATES` directive is:

```
DB2SPATIAL_SIMPLIFY_AGGREGATES YES
```

Workbench Parameter: *Simplify Aggregate Geometry*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

SPATIAL_PREDICATE

Required/Optional: *Optional*

This specifies the type of spatial relationship which must exist between the search envelope and the geometry in the target layer. Any supported relationship, in combination with the **SPATIAL_PREDICATE_RESULT** directive, can be used to filter the features being read.

Values: *CONTAINS, CROSSES, DISJOINT, EQUALS, INTERSECTS, ORDERINGEQUALS, OVERLAPS, TOUCHES, WITHIN*

Default value: *INTERSECTS*

For example,

```
DB2SPATIAL_SPATIAL_PREDICATE INTERSECTS  
DB2SPATIAL_SPATIAL_PREDICATE_RESULT FALSE
```

This would result in a spatial filter using DB2 Spatial's native spatial function

```
DB2GSE.ST_Intersects( g1 geometry, g2 geometry) = 0
```

where **g1** is the search envelope and **g2** is the target feature. This will cause FME to return only those features that satisfy the spatial predicate above.

The following table lists the valid spatial predicate relationships.

Search Method	Description
CONTAINS	Determines whether the search envelope is com-

Search Method	Description
	pletely contained by the target feature.
CROSSES	Determines whether the intersection of search envelope and the target feature results in a geometry object whose dimension is one less than the maximum dimension of the source geometries. Also determines if the intersection object contains points that are interior to both source geometries and are not equal to either of the source objects.
DISJOINT	Determines whether the intersection of search envelope with the target feature is an empty set.
EQUALS	Determines whether the search envelope and target feature are of the same type and have identical x,y coordinate values.
INTERSECTS	Determines whether the intersection of search envelope and target feature does not result in an empty set. This is the exact opposite of DISJOINT.
ORDERINGEQUALS	Determines whether the search envelope and target feature are equal and the coordinates are in the same order.
OVERLAPS	Determines whether the search envelope and target feature overlap each other.
TOUCHES	Determines whether any of the points common to search envelope and target feature intersect the interiors of both geometries. At least one geometry must be a linestring, polygon, multilinestring, multipolygon.
WITHIN	Determines whether the target feature is completely within the search envelope. This is exactly opposite to CONTAINS .

For more details on Spatial predicate, please refer to the *IBM DB2 Spatial Extender User's Guide and Reference*.

Workbench Parameter: *Spatial Relationship to Search Envelope*

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]

* Workbench Parameter

Clip To Envelope

WHERECLAUSE

Required/Optional: *Optional*

This specifies an SQL **WHERE** clause, which is applied to the table's columns to limit the resulting features. This feature is currently limited to apply only to the attributes of the target table, and does not allow for joining multiple tables together. The effect of table joins can be achieved using the object model, by specifying the entire queries in the **DEF** line with a **db2_sql** parameter.

By default, there is no **WHERE** clause applied to the results, so all features in the layer are returned.

Example:

```
DB2SPATIAL_WHERECLAUSE "se_row_id > 45"
```

Workbench Parameter: *WHERE Clause*

TRANSACTION_INTERVAL

Required/Optional: *Optional*

The features can be read from the DB2 Spatial database using a bulk reading technique to maximize performance. Normally 1000 rows of data are read from the database at a time. However, when we are reading LOB (BLOBs or CLOBs) data, we are restricted to a transaction interval of size 1. Since geometry columns are normally BLOB types, reading of spatial features will not be affected by this directive.

This directive allows users to tune the performance of the reader. It specifies how many rows are read from the database at a time.

Example:

```
DB2SPATIAL_TRANSACTION_INTERVAL "se_row_id > 45"
```

Workbench Parameter: *Transaction Buffer Size*

BEGIN_SQL{n}

Occasionally you must execute some ad-hoc SQL prior to opening a table. For example, it may be necessary to ensure that a view exists prior to attempting to read from it.

Upon opening a connection to read from a database, the reader looks for the directive <ReaderKeyword>_BEGIN_SQL{n} (for n=0,1,2,...), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the **FME_SQL_DELIMITER** keyword, embedded at the beginning of the SQL block. The single character following this keyword will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;  
DELETE FROM instructors;  
DELETE FROM people  
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

* Workbench Parameter

SQL Statement to Execute Before Translation

END_SQL{n}

Occasionally you must execute some ad-hoc SQL after closing a set of tables. For example, it may be necessary to clean up a temporary view after writing to the database.

Just before closing a connection on a database, the reader looks for the directive `<ReaderKeyword>_END_SQL{n}` (for $n=0,1,2,\dots$), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the `FME_SQL_DELIMITER` directive, embedded at the beginning of the SQL block. The single character following this directive will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;  
DELETE FROM instructors;  
DELETE FROM people  
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

* Workbench Parameter

SQL Statement to Execute After Translation

PERSISTENT_CONNECTION

A user may want to keep a connection to a database for reuse during a particular FME session. For example, when running a batch of 100 mapping files on the same database connection, it may be desirable to keep a connection open and save the processing time required to make and break a database connection.

A database connection will be determined to be the same when the database name, the username, the password, and the transaction interval are the same.

Values: *YES* | *NO*

Default value: *NO*

Example:

```
DB2SPATIAL_PERSISTENT_CONNECTION YES
```

Workbench Parameter: *Persistent Connection*

RETRIEVE_ALL_SCHEMAS

This specification is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

This optional specification is used to tell the reader to retrieve the names and the schemas of all the tables in the source database. If this value is not specified, it is assumed to be "No".

The syntax of the RETRIEVE_ALL_SCHEMAS directive is:

```
DB2SPATIAL_RETRIEVE_ALL_SCHEMAS Yes
```

RETRIEVE_ALL_TABLE_NAMES

This specification is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

Similar to **RETRIEVE_ALL_SCHEMAS**; this optional specification is used to tell the reader to only retrieve the table names of all the tables in the source database. If **RETRIEVE_ALL_SCHEMAS** is also set to "Yes," then **RETRIEVE_ALL_SCHEMAS** is chosen. If this value is not specified, it is assumed to be "No".

The syntax of the RETRIEVE_ALL_TABLE_NAMES directive is:

```
DB2SPATIAL_RETRIEVE_ALL_TABLE_NAMES Yes
```

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

✳ Workbench Parameter

Additional Attributes to Expose

Writer Overview

The DB2 Spatial writer module stores both geometry and attributes into DB2 Spatially enabled databases. The DB2 Spatial writer provides the following capabilities:

- **Table Creation:** Uses the information within the FME mapping file to automatically create database tables as needed.
- **Coordinate System:** Checks for a coordinate system and if a matching one is not found then it will create one automatically. The matching criteria is the OGC WKT definition of the coordinate system.

Note: When writing to DB2 V7.2, the exact coordinate system definition of the incoming features should exist in the database; otherwise FME will not be able to create a new coordinate system if the user does not have enough privileges to write to the DB2GSE.GSE_COORD_REF metadata table. User need to have either DBADB authority or at least have **INSERT** privileges for the coordinate system metadata table DB2GSE.GSE_COORD_REF for the translation to succeed.

- **Spatial Reference System:** Uses an existing Spatial reference system with matching parameters or create new one as required when registering spatial layer(column).
- **Spatial Grid Index Creation:** Creates spatial indexes only if valid values are specified for different levels of grid.
- **ESRI's ArcExplorer 3.0 JDBC Edition:** In order to view spatial data written by DB2 Spatial writer in ArcExplorer 3.0 each spatial table must have a column named "SE_ROW_ID" of type **integer**. It does not really matter whether the column is populated or not. Also note that ArcExplorer expects users to at least have execute privileges on certain functions in DB2GSE schema or have DBADM authority.

Note: **Bulk Loading:** The DB2 Spatial writer does not use a bulk loading technique due to certain limitations in DB2.

Writer Directives

The directives processed by the DB2 Spatial writer are listed below. The suffixes shown are prefixed by the current **<writerKeyword>** in a mapping file. By default, the **<writerKeyword>** for the DB2 Spatial writer is **DB2SPATIAL** when using the object model.

DATASET, USER_NAME, PASSWORD, PERSISTENT_CONNECTION, TRANSACTION_INTERVAL, BEGIN_SQL{ }, and END_SQL{ }

The DATASET, USER_NAME, PASSWORD, PERSISTENT_CONNECTION, TRANSACTION_INTERVAL, BEGIN_SQL{ }, and END_SQL{ } directives operate in the same manner as they do for the DB2 Spatial reader. The remaining writer-specific directives are discussed in the following sections.

DEF

Required/Optional: *Optional*

Each DB2 Spatial table must be defined before it can be written. The general form of a DB2 Spatial definition statement is:

```
DB2SPATIAL_DEF <tableName> \
  [db2_overwrite_table <YES|NO|TRUNCATE>] \
  [db2_multi_geometry <YES|NO|FIRST_FEATURE>] \
  [db2_geometry_column <geometry>] \
  [db2_offset_x <x offset value>] \
  [db2_offset_y <y offset value>] \
  [db2_scale_x <x scale value>] \
  [db2_scale_y <y scale value>] \
  [db2_grid_0 <finest grid size>] \
  [db2_grid_1 <middle grid size>] \
  [db2_grid_2 <coarsest grid size>] \
  [db2_sql <sql statement>] \
  [db2_update_key_columns <column>[,<column>]...] \
  [db2_delete_key_columns <column>[,<column>]...] \
  [<fieldName> <fieldType>]*
```

The table definition allows complete control of the layer that will be created. If the layer already exists, the majority of the **DEF** line parameters will be ignored and need not be given. As well, if the table already exists in the database, then it is not necessary to list the fields and their types – FME will use the schema information in the database to determine this. FME will ignore the field names and types specified on the **DEF** line, except for the one with type geometry.

If the table does not exist, then the field names and types are used to first create the table. In any case, if a **<fieldType>** is given, it may be any field type supported by the target database.

The DB2 Spatial writer will use **db2_geometry_column** parameter to set the name of geometry column for the new table. If the **db2_geometry_column** parameter is not specified then a default name "geometr" will be used for the geometry column.

The configuration parameters present on the definition line are described in the following table:

Parameter	Contents
db2_overwrite_table	This parameter can have one of <YES NO TRUNCATE> option. If YES , then the table will be dropped and created again. If TRUNCATE , then all the rows from the table will be deleted. If NO , then data will be appended to the existing table.
db2_multi_geometry	<p>This specifies whether the db2 types for point, linestring and polygon should be written as multi-geometries or single geometries. If YES, the table created has multi-geometries (that is, the geometry column type will be ST_MULTIPPOINT, and the features are coerced into multi-geometries if they are not already). If NO, the geometry column of the created table is singular (that is, ST_POINT), and multi-geometries are split. FIRST_FEATURE allows this setting to be based on the first feature in the table.</p> <p>This setting is used for DB2SPATIAL-to-DB2SPATIAL translations.</p>
db2_geometry_column	This parameter can be used to specify name of the spatial layer (geometry column name). If it is not specified db2 spatial writer module will use default name "geometry" for the spatial layer.
db2_offset_x	The x offset value for the dataset, defaults to 0. If this parameter is non-zero, then it overrides the global OFFSET_X directive.
db2_offset_y	The y offset value for the dataset, defaults to 0. If this parameter is non-zero, then it overrides the global OFFSET_Y directive.
db2_scale_x	The x scale value for the dataset, defaults to 1. If this parameter is not equal to 1, then it overrides the global SCALE_X directive
db2_scale_y	The y scale value for the dataset, defaults to 1. If this parameter is not equal to 1, then it overrides the global SCALE_Y directive.
db2_grid_0	This parameter specifies the finest spatial index grid size. If 0, then a spatial index is not created.
db2_grid_1	This parameter specifies the middle spatial index grid size. If 0, then a spatial index is not created.
db2_grid_2	This parameter specifies the coarsest spatial index grid size. If 0, then a spatial index is not created.
db2_sql	This specifies an SQL INSERT or UPDATE query to be used to

Parameter	Contents
	<p>define the results. If this is specified, the DB2 Spatial writer will execute the query, defining one row for each feature from FME. The values in the query are specified by embedding “?attrName” in the query itself, where attrName is the name of the FME feature’s attribute.</p> <p>For example: INSERT INTO MyTable VALUES(?ID,?NAME,?DESC) In this example, the attributes named ID, NAME and DESC will be taken from each feature written to <tableName>.</p> <p>or INSERT INTO MyTable (ID,NAME) VALUES(?ID,?NAME) In this example, the attributes named ID and NAME will be taken from each feature written to <tableName>. If not all attributes are to be written, then a column list should be specified as shown in the second example statement where 2 out of 3 columns are being written.</p> <p>It is also very important that the attributes named in the query must be listed on the DEF line so that FME knows what type to use. There is no necessary or implied correlation between the FME attribute name and the db2 column name.</p>
db2_update_key_columns	<p>This instructs the DB2 Spatial writer to perform an UPDATE operation on the table, rather than performing an INSERT. The argument is a comma-separated list of the columns which are matched against the corresponding FME attributes’ values to specify which rows are to be updated with the other attribute values.</p> <p>For example: db2_update_key_columns ID,NAME In this case the FME attribute is always matched against the db2 column with the same name. Also, the target table is always the feature type specified in the DEF line. Each column listed with the db2_update_key_columns directive must be defined with a type on the DEF line, in addition to the columns whose values will be updated by the operation. This cannot be used with db2_delete_key_columns. Also, the keys cannot be of type BLOB, CLOB, or LONG_VARCHAR.</p>
db2_delete_key_columns	<p>This instructs the DB2 Spatial writer to perform a DELETE operation on the table, rather than performing an INSERT. The argument is a comma-separated list of the columns which are matched against the corresponding FME attributes’ values to specify which rows are to be deleted when their values match the other attribute values.</p> <p>For example:</p>

Parameter	Contents
	<p><code>db2_delete_key_columns ID,NAME</code> would delete those rows in the table whose values match the attribute values passed in through this <code>DEF</code> line. The FME attribute is always matched against the DB2 Spatial column with the same name. Also, the target table is always the feature type specified in the <code>DEF</code> line. Each column listed with the <code>db2_delete_key_columns</code> directive must be defined with a type on the <code>DEF</code> line, in addition to the columns whose values will be updated by the operation. This cannot be used with <code>db2_update_key_columns</code>. Also, the keys cannot be of type <code>BLOB</code>, <code>CLOB</code>, or <code>LONG_VARCHAR</code>.</p>

TRANSACTION_INTERVAL

This statement informs the FME about the number of features to be placed in each transaction before a transaction is committed to the database.

If the `DB2SPATIAL_TRANSACTION_INTERVAL` statement is not specified, then a value of 1000 is used as the transaction interval.

Parameter	Contents
<code><transaction_interval></code>	The number of features in a single transaction.

Default: `1000`

Example:

```
DB2SPATIAL_TRANSACTION_INTERVAL 2500
```

Workbench Parameter: *Transaction Interval*

PERSISTENT_CONNECTION

Required/Optional: *Optional*

A user may want to keep a connection to a database for reuse during a particular FME session. For example, when running a batch of 100 mapping files on the same database connection, it may be desirable to keep a connection open and save the processing time required to make and break a database connection.

A database connection will be determined to be the same when the database name, the username, the password, and the transaction interval are the same.

Values: `YES` | `NO`

Default: `NO`

Example:

```
DB2SPATIAL_PERSISTENT_CONNECTION YES
```

Workbench Parameter: *Persistent Connection*

ABORT_ON_BAD_DATA

Required/Optional: *Optional*

Some features' geometries may fail DB2 Spatial Extender's check constraints based on the offset, scale, and coordinate system values. These features, as well as others with out-of-range or invalid attribute values, will be rejected and cannot be written to the database. If the value of this directive is YES then the translation will be aborted immediately after encountering such a problem. If this directive is set to NO then the translation will continue but the rejected features will not be written to the database.

Values: *YES* | *NO*

Default: *YES*

Example:

```
DB2SPATIAL_ABORT_ON_BAD_DATA YES
```

Workbench Parameter: *Abort Translation On Bad Data*

OFFSET_X

Required/Optional: *Optional*

This directive can be used to set the global **x** offset for the entire translation. If a dataset contains many different tables but the same **x** offset applies to all of them, then this is a convenient way of setting the **x** offset. This value can be overridden by **DEF** line parameter **db2_offset_x**.

Default: *0*

Example:

```
DB2SPATIAL_OFFSET_X -12456
```

Workbench Parameter: *Offset X*

OFFSET_Y

Required/Optional: *Optional*

This directive can be used to set the global **y** offset for the entire translation. If a dataset contains many different tables but the same **y** offset applies to all of them, then this is a convenient way of setting the **y** offset. This value can be overridden by **DEF** line parameter **db2_offset_y**.

Default: *0*

Example:

```
DB2SPATIAL_OFFSET_Y -1245
```

Workbench Parameter: *Offset Y*

SCALE_X

Required/Optional: *Optional*

This directive can be used to set the global **x** scale value for the entire translation. If a dataset may contains many different tables but the same **x** scale applies to all of them, then this is a convenient way of setting the **x** scale value. This value can be overridden by **DEF** line parameter **db2_scale_x**.

Default: *1*

Example:

```
DB2SPATIAL_SCALE_X 1000
```

Workbench Parameter: *Scale X*

SCALE_Y

Required/Optional: *Optional*

This directive can be used to set the global **y** scale value for the entire translation. If a dataset contain many different tables but the same **y** scale applies to all of them, then this is a convenient way of setting the **y** scale value. This value can be overridden by **DEF** line parameter **db2_scale_y**.

Default: *1*

Example:

```
DB2SPATIAL_SCALE_Y 1000
```

Workbench Parameter: *Scale Y*

GRID_0

Required/Optional: *Optional*

This directive can be used to set the global finest grid size for the spatial grid index . If a dataset contains many different tables but the same finest grid size applies to all of them, then this is a convenient way of setting the finest grid size value. This value can be overridden by **DEF** line parameter **db2_grid_0**.

Default: *0*

Example:

```
DB2SPATIAL_GRID_0 10
```

Workbench Parameter: *Finest Spatial Grid Index Size*

GRID_1

Required/Optional: *Optional*

This directive can be used to set the global middle grid size for the spatial grid index. If a dataset contains many different tables but the same middle grid size applies to all of them, then this is a convenient way of setting the middle grid size value. This value can be overridden by **DEF** line parameter **db2_grid_1**.

Default: *0*

Example:

```
DB2SPATIAL_GRID_1 100
```

Workbench Parameter: *Middle Spatial Grid Index Size*

GRID_2

Required/Optional: *Optional*

This directive can be used to set the global coarsest grid size for the spatial grid index. If a dataset contains many different tables but the same coarsest grid size applies to all of them, then this is a convenient way of setting the coarsest grid size value. This value can be overridden by **DEF** line parameter **db2_grid_2**.

Default: *0*

Example:

```
DB2SPATIAL_GRID_2 1000
```

Workbench Parameter: *Coarsest Spatial Grid Size*

Feature Representation

Features read from DB2 Spatial consist of a series of attribute values and geometry. The feature type of each Database feature is as defined on its **DEF** line.

Features written to the database have the destination table as their feature type, and attributes as defined by on the **DEF** line.

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), the DB2 Spatial module adds the format-specific attributes described below:

Attribute Name	Contents
db2_type	The type of geometric entity stored within the feature. The valid values for the object model are listed below: db2_nil db2_point db2_linestring db2_polygon

Features read from, or written to, DB2 Spatial also have an attribute for each column in the database table. The feature attribute name will be the same as the source or destination column name. The attribute and column names are not case-sensitive.

No Coordinates

db2_type: db2_nil

Features with no coordinates are tagged with this value when reading or writing to or from db2 Spatial.

Points

db2_type: db2_point

All DB2 Spatial point and multipoint features are read as **db2_point**. The only difference being the geometry type of feature, which will be set to **fme_aggregate** if it is a multipoint and **fme_point** if it is a point.

Lines

db2_type: db2_line

All DB2 Spatial linestring and Multilinestring features are read as **db2_line**. The only difference is the geometry type of feature, which will be set to **fme_aggregate** if it is a multilinestring and **fme_line** if it is a linestring.

Polygons

db2_type: db2_polygon

All DB2 Spatial polygon and Multipolygon features are read as **db2_polygon**. The only difference is the geometry type of feature, which will be set to **fme_aggregate** if it is a multipolygon and **fme_polygon** if it is a polygon. Polygon features include donut polygons with one or more holes.

Aggregates are written out as "multipolygon" geometry containing several polygonal elements, just as if the feature had been tagged with **db2_multiline**. Any non-polygonal elements contained in the aggregate are discarded.

The following table summarizes all of the db2_type values that are possible with DB2 Spatial geometry, and provides a description of each representation.

db2_type	DB2 Spatial type	Representation
db2_nil	N/A	No geometry
db2_point	POINT	Single point geometry. fme_geometry = fme_point fme_type = fme_point
	MULTIPOINT	Aggregate containing one or more points. fme_geometry = fme_aggregate fme_type = fme_point

db2_type	DB2 Spatial type	Representation
db2_line	LINestring	Single line geometry. fme_geometry = fme_line fme_type = fme_line
	MULTILINESTRING	An aggregate of linestrings. fme_geometry = fme_aggregate fme_type = fme_line
db2_polygon	POLYGON	A single polygon or donut geometry. fme_geometry = fme_polygon or fme_donut fme_type = fme_polygon
	MULTIPOLYGON	An aggregate of simple polygons or donut polygons. fme_geometry = fme_aggregate fme_type = fme_polygon

Troubleshooting

Problems sometimes arise when attempting to connect to an DB2 Spatial database. This is almost always due to a mis-configuration in the user's environment. The following suggestions can often help detect and overcome such problems.

- Ensure you can connect to the database with the data source name, username, and password using DB2 Command Line processor.
- Ensure that you have the correct version of the DB2 client software installed.
- Ensure that the appropriate version of DB2 Spatial Extender is installed and the database is 'Spatially enabled'. If you get an error which says something like "DB2GSE.*.. is an undefined name", then it is most likely that the database is not enabled for spatial operations. For enabling a DB2 database for spatial operations, please refer to *IBM DB2 Spatial Extender User's Guide and Reference*.
- Ensure that you have the appropriate privileges to perform the operations like creating, dropping, inserting into, and deleting from tables when writing. DBADM privileges may be required to create indexes. Please check the DB2 database manuals for more information.
- When reading/writing large volumes of data, please ensure that the database configuration parameters are set for large data processing. For example, when reading/writing large volumes of data, failure may occur due to "app_ctl_heap_sz" and/or "logprimary" parameters not set to appropriate values. Most database errors will be logged as obtained from the database. Some error messages may not immediately imply the actual problem. For such messages, please refer to DB2 database manuals.
- If offset and scale values are not chosen appropriately for the dataset, some or all geometries may be rejected. Error messages may not indicate the actual problem. For example, not choosing an appropriate scale may result in duplicate coordinates and the error message may be "not enough points" or "polygon intersects itself". Please refer to the *IBM DB2 Spatial Extender User's Guide and Reference* manual for the resolution of such errors.

IBM Informix Reader/Writer

The IBM Informix Reader/Writer (called *Informix Reader/Writer* throughout the rest of this chapter) provides FME with access to attribute data held in IBM's Informix database tables.

Overview

This data may or may not have a spatial component to it. Thus the Informix reader can read from Informix databases which may or may not be spatially enabled. The FME provides read and write access to live databases accessible via the Informix Connect driver.

Informix Database Quick Facts

Format Type Identifier	INFX
Reader/Writer	Both
Licensing Level	Professional
Dependencies	Informix Connect software
Dataset Type	Database
Feature Type	Table name
Typical File Extensions	N/A
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	db_type

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	point	no
circles	no	polygon	no
circular arc	no	raster	no
donut polygon	no	solid	no
elliptical arc	no	surface	no
ellipses	no	text	no
line	no	z values	N/A
none	yes		

Reader Overview

FME considers an Informix dataset to be a collection of relational tables. The tables must be defined in the mapping file before they can be read. Arbitrary **WHERE** clauses and joins are fully supported.

Reader Directives

The suffixes listed are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the Informix reader is **INFX**.

DATASET

Required/Optional: *Required*

This is the data source name. On Windows, it is an ODBC data source name.

Example:

```
INFX_DATASET sample
```

Workbench Parameter: *Source IBM Informix Dataset*

USER_NAME

Required/Optional: *Optional*

The name of the user who will access the database. By default, **USER_NAME** will be considered the same as the schema name. For example, any table name which explicitly does not have the schema name prefixed will be considered as a table from the schema for that user.

Example:

```
INFX_USER_NAME bond007
```

Workbench Parameter: *User ID*

PASSWORD

Required/Optional: *Optional*

The password to access the database.

Example:

```
INFX_PASSWORD moneypenny
```

Workbench Parameter: *Password*

DEF

Required/Optional: *Optional*

Each database table must be defined before it can be read. There are two forms that the definition may take.

The syntax of the first form is:

```
INFX_DEF <tableName> \  
    [SQL_WHERE_CLAUSE <whereClause>] \  
    [<fieldName> <fieldType>] +
```

In this form, the fields and their types are listed. The **<fieldType>** of each field must be given, but it is not verified against the database definition for the field. In effect, it is ignored.

The **<tableName>** must match a table in the database. This will be used as the feature type of all the features read from the table.

If no `<whereClause>` is specified, all rows in the table will be read and returned as individual features, unless limited by a global directive:

```
<ReaderKeyword>_WHERE_CLAUSE
```

If a `<whereClause>` is specified, only those rows that are selected by the clause will be read. Note that the `<whereClause>` does not include the word "WHERE."

In this example, the all records whose ID is less than 5 will be read from the supplier table:

```
INFX_DEF supplier \  
  INFX_WHERE_CLAUSE "id < 5" \  
  ID integer \  
  NAME char(100) \  
  CITY char(50)
```

The syntax of the second form is:

```
INFX_DEF <tableName> \  
  INFX_SQL <sqlStatement>
```

In this form, an arbitrary complete `<sqlStatement>` will be executed. The statement is passed untouched to the database (and therefore may include non-portable database constructions). The results of the statement will be returned, one row at a time, as features to FME. This form allows the results of complex joins to be returned to FME.

In this example, the results of joining the `employee` and `city` tables are returned. All attributes from the two tables will be present on each returned feature. The feature type will be set to `complex`.

```
INFX_DEF complex \  
  SQL_STATEMENT \  
  "SELECT * FROM EMPLOYEE, CITY WHERE EMPLOYEE.CITY = CITY.NAME"
```

WHERECLAUSE

Required/Optional: *Optional*

This optional specification is used to limit the rows read by the reader from each table. If a given table has no `INFX_WHERE_CLAUSE` or `INFX_SQL` specified in its `DEF` line, the global `<ReaderKeyword>_WHERECLAUSE` value, if present, will be applied as the `WHERE` specifier of the query used to generate the results. If a table's `DEF` line does contain its own `SQL_WHERE_CLAUSE` or `SQL_STATEMENT`, it will override the global `WHERE` clause.

The syntax for this clause is:

```
INFX_WHERECLAUSE <whereClause>
```

Note that the `<whereClause>` does not include the word "WHERE."

The example below selects only the features whose lengths are more than 2000:

```
INFX_WHERECLAUSE LENGTH > 2000
```

Workbench Parameter: *Where Clause*

BEGIN_SQL{n}

Occasionally you must execute some ad-hoc SQL prior to opening a table. For example, it may be necessary to ensure that a view exists prior to attempting to read from it.

Upon opening a connection to read from a database, the reader looks for the directive `<ReaderKeyword>_BEGIN_SQL{n}` (for `n=0,1,2,...`), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the **FME_SQL_DELIMITER** keyword, embedded at the beginning of the SQL block. The single character following this keyword will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;
DELETE FROM instructors;
DELETE FROM people
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

* Workbench Parameter

SQL Statement to Execute Before Translation

END_SQL{n}

Occasionally you must execute some ad-hoc SQL after closing a set of tables. For example, it may be necessary to clean up a temporary view after writing to the database.

Just before closing a connection on a database, the reader looks for the directive **<ReaderKeyword>_END_SQL{n}** (for $n=0, 1, 2, \dots$), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the **FME_SQL_DELIMITER** directive, embedded at the beginning of the SQL block. The single character following this directive will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;
DELETE FROM instructors;
DELETE FROM people
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

* Workbench Parameter

SQL Statement to Execute After Translation

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined database tables that will be read. If no **IDS** are specified, then all defined and available tables are read. The syntax of the **IDS** keyword is:

```
INFX_IDS <featureType1> \  
<featureType2> ... \  
<featureTypeN>
```

The feature types must match those used in **DEF** lines.

The example below selects only the **HISTORY** table for input during a translation:

```
INFX_IDS HISTORY
```

RETRIEVE_ALL_SCHEMAS

Required/Optional: *Optional*

This directive is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

This optional directive is used to tell the reader to retrieve the names and the schemas of all the tables in the source database. If this value is not specified, it is assumed to be "No".

The syntax of the **RETRIEVE_ALL_SCHEMAS** directive is:

```
INFX_RETRIEVE_ALL_SCHEMAS Yes
```

RETRIEVE_ALL_TABLE_NAMES

Required/Optional: *Optional*

This directive is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

Similar to **RETRIEVE_ALL_SCHEMAS**; this optional directive is used to tell the reader to only retrieve the table names of all the tables in the source database. If **RETRIEVE_ALL_SCHEMAS** is also set to "Yes", then **RETRIEVE_ALL_SCHEMAS** will take precedence. If this value is not specified, it is assumed to be "No".

The syntax of the **RETRIEVE_ALL_TABLE_NAMES** directive is:

```
INFX_RETRIEVE_ALL_TABLE_NAMES Yes
```

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

*** Workbench Parameter**

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The Informix writer module stores attribute records into a live relational database. The writer provides the following capabilities:

- **Table Creation:** The Database writer uses the information within the FME mapping file to automatically create database tables as needed.

Writer Directives

The directives processed by the Informix Writer are listed below. The suffixes shown are prefixed by the current `<writerKeyword>` in a mapping file. By default, the `<writerKeyword>` for the Informix writer is `INFX`.

DATASET, USER_NAME, PASSWORD, BEGIN_SQL{n}, END_SQL{n}

The `DATASET`, `USER_NAME`, `PASSWORD`, `BEGIN_SQL{n}`, and `END_SQL{n}` directives operate in the same manner as they do for the Informix reader. The remaining writer-specific directives are discussed in the following sections.

ABORT_ON_BAD_DATA

Required/Optional: *Optional*

Some features may contain out-of-range or invalid attribute values. These features will be rejected and cannot be written to the database. If the value of this directive is YES then the translation will be aborted immediately after encountering such a problem. If this directive is set to NO then the translation will continue but the features with rejected feature will not be written to the database.

Values: `YES` | `NO`

Default: `NO`

Example:

```
INFX_ABORT_ON_BAD_DATA YES
```

Workbench Parameter: *Abort Translation on Bad Data*

DEF

Required/Optional: *Required*

Each database table must be defined before it can be written. For the Informix writer, the DEF line is specified in one of three forms. The first one is used for inserting data:

```
INFX_DEF <tableName> \  
    [infx_overwrite_table (YES|NO|TRUNCATE)] \  
    [<fieldName> <fieldType>] +
```

In this form, the fields and their types are listed. If the table already exists in the database, and `infx_overwrite_table` is not specified with a parameter of YES, FME will append its information the existing database table. In this case, the fields and their types must still be listed, but a subset may be used. If a subset is used, NULL values will be written into the unspecified columns.

If the table does not exist, or `infx_overwrite_table` is specified with a value of YES, then the field names and types are used to first create the table. In any case, if a `<fieldType>` is given, it may be any field type supported by the target database.

This example defines the `SUPPLIER` table for the FME. If the table did not exist, it will be created just before the first `SUPPLIER` row is written. If the table already exists, the data will be appended to the existing table.

```
INFX_DEF SUPPLIER \  
    ID integer \  
    NAME char(100) \  
    CITY char(50)
```

The following example is exactly the same, except that it replaces any existing table named SUPPLIER with a new table having the specified definition. If the table **SUPPLIER** does not exist in the database, then a new table is simply created.

```
INFX_DEF SUPPLIER \
  infx_overwrite_table YES \
  ID integer \
  NAME char(100) \
  CITY char(50)
```

The second form is used for updating data:

```
INFX_DEF <tableName> \
  [infx_overwrite_table (YES|NO|TRUNCATE)] \
  [infx_update_key_columns <columns>] \
  [<fieldName> <fieldType>] +
```

The set of key columns to use (shown <columns> as above) is specified as a comma-separated list. The infx_overwrite_table parameter should always be set to YES in this mode.

The third form is used for deleting data:

```
INFX_DEF <tableName> \
  [infx_overwrite_table (YES|NO|TRUNCATE)] \
  [infx_delete_key_columns <columns>] \
  [<fieldName> <fieldType>] +
```

The set of key columns to use (shown <columns> as above) is specified as a comma-separated list. The infx_overwrite_table parameter should always be set to YES in this mode.

TRANSACTION_INTERVAL

This statement informs FME about the number of features to be placed in each transaction before a transaction is committed to the database.

If the **INFX_TRANSACTION_INTERVAL** statement is not specified, then a value of 1000 is used as the transaction interval.

Parameter	Contents
<transaction_interval>	The number of features in a single transaction.

Default: 1000

Example:

```
INFX_TRANSACTION_INTERVAL 5000
```

Workbench Parameter: *Transaction Interval*

Feature Representation

Features read from an Informix database consist of a series of attribute values. They have no geometry. The attribute names are as defined in the **DEF** line if the first form of the **DEF** line was used. If the second form of the **DEF** line was used, then the attribute names are as they are returned by the query, and as such may have their original table names as qualifiers. The feature type of each Informix feature is as defined on its **DEF** line.

Features written to the database have the destination table as their feature type, and attributes as defined on the **DEF** line.

DATE and DATETIME Fields

When a **DATE** field is read by the Informix reader, two attributes are set in the FME feature. The first attribute has the name of the database column, and its value is of the form **YYYYMMDD**. This is compatible with all other FME date and time values.

The second attribute has a suffix of **.full** and is of the form **YYYYMMDDHHMMSS**. It specifies the date and the time, with the time portion specified using the 24-hour clock.

For example, if a date field called **UPDATE_DATE** is read, the following attributes will be set in the retrieved FME feature:

```
UPDATE_DATE = '19980820'  
UPDATE_DATE.full = '19980820000000'
```

The Informix writer looks for both attributes when a **DATE** or **DATETIME** column is being output. Either may be specified. If both attributes are specified, then the value specified in the **<name>.full** attribute takes precedence.

IBM Informix Spatial Reader/Writer

FME's Informix Spatial Reader/Writer module (referred to as *Informix Spatial* in this chapter) enables FME to read and write spatial and attribute data inside Informix databases.

Overview

Informix Spatial can read from databases which are spatially enabled but the tables may or may not have spatial information stored.

Informix Spatial Quick Facts

Format Type Identifier	INFXSPATIAL
Reader/Writer	Both
Licensing Level	Reading: Professional Writing: DB2, Oracle, or SQL Server Edition
Dependencies	Informix Connect Software
Dataset Type	Database
Feature Type	Table name
Typical File Extensions	N/A
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	infx_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	yes		z values	no
none	yes			

Reader Overview

The FME considers an Informix Spatial dataset to be a database containing a collection of relational tables together with their geometry. The tables to be read are defined in the mapping file. Arbitrary **WHERE** clauses are supported.

Reader Directives

The suffixes listed are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the Informix Spatial reader is **INFXSPATIAL**.

DATASET

Required/Optional: *Required*

This is the data source name. On Windows, it is an ODBC data source name.

Example:

```
INFXSPATIAL_DATASET sample
```

Workbench Parameter: *Source IBM Informix Spatial Dataset*

USER_NAME

Required/Optional: *Required*

The name of the user who will access the database. By default, **USER_NAME** will be considered the same as the schema name. For example, any table name which explicitly does not have the schema name prefixed will be considered as a table from the schema for that user.

Example:

```
INFXSPATIAL_USER_NAME bond007
```

Workbench Parameter: *User ID*

PASSWORD

Required/Optional: *Required*

The password to access the database.

Example:

```
INFXSPATIAL_PASSWORD moneypenny
```

Workbench Parameter: *Password*

DEF

Required/Optional: *Optional*

The syntax of the definition is:

```
INFXSPATIAL_DEF <tableName> \  
    [infx_geometry_column <geometryColumn>] \  
    [infx_where_clause <whereClause>] \  
    [<fieldName> <fieldType>] +
```

The **<fieldType>** of each field must be given, but it is not verified against the database definition for the field. In effect, it is ignored.

The geometry column or columns should not be specified as attributes on the **DEF** line. Rather, the desired geometry column may be specified in the `infx_geometry_column` parameter. In the case that `infx_geometry_column` is not specified and a feature type has more than one registered geometry column or layer, the Informix Spatial Reader will arbitrarily choose one as the primary geometry column.

The `<tableName>` can be either fully qualified or not. A fully qualified table name consist of two parts separated by a period (.). The first part is the `<schema name>` and second part is the `<table name>`. The `<table name>` part must match a table in the schema specified by the `<schema name>` part of the `<tableName>`. If a schema name is not provided as part of the table name, then the username will be considered the schema name. This will be used as the feature type of all the features read from the table. For example, if a user wants to read a table from its own schema then only the table name can be provided, but if the user wants to read from a different user's schema, then table name should be qualified with schema name.

The definition allows specification of separate search parameters for each table. If any of the configuration parameters are given, they will override, for that table, whatever global values have been specified by the reader directives listed above. If any of these parameters is not specified, the global values will be used.

The following table summarizes the definition line configuration parameters:

Parameter	Contents
<code>infx_geometry_column</code>	This specifies the spatial layer or geometry column to use for reading spatial data in case the table has multiple geometry/spatial columns.
<code>infx_where_clause</code>	This specifies the SQL WHERE clause applied to the attributes of the layer's features to limit the set of features returned. If this is not specified, the value of the <code><ReaderKeyword>_WHERE_CLAUSE</code> directive is used.

If no `<whereClause>` is specified, all rows in the table will be read and returned as individual features. If a `<whereClause>` is specified, only those rows that are selected by the clause will be read. Note that the `<whereClause>` does not include the word "where".

WHERECLAUSE

Required/Optional: *Optional*

This optional specification is used to limit the rows read by the reader from each table. If a given table has no `INFX_WHERE_CLAUSE` specified in its `DEF` line, the global `<ReaderKeyword>_WHERECLAUSE` value, if present, will be applied as the `WHERE` specifier of the query used to generate the results.

The syntax for this clause is:

```
INFX_WHERECLAUSE <whereClause>
```

Note that the `<whereClause>` does not include the word "WHERE."

The example below selects only the features whose lengths are more than 2000:

```
INFX_WHERECLAUSE LENGTH > 2000
```

Workbench Parameter: *Where Clause*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

*** Workbench Parameter**

Minimum X, Minimum Y, Maximum X, Maximum Y

INFXSPATIAL Mapping File Example

The example below selects a small area for extraction:

```
INFXSPATIAL_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

*** Workbench Parameter**

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

*** Workbench Parameter**

Clip To Envelope

SIMPLIFY_AGGREGATES

Required/Optional: *Optional*

This directive specifies whether multi-geometry or aggregate features with one member are read as stored or simplified and read as single member. e.g. an aggregate of points or multipoint features with only one point will be returned as a simple point if the value of this directive is **YES**.

Values: YES | NO

Default value: *NO*

Example:

The syntax of the SIMPLIFY_AGGREGATES directive is:

```
INFSPATIAL_SIMPLIFY_AGGREGATES YES
```

BEGIN_SQL{n}

Occasionally you must execute some ad-hoc SQL prior to opening a table. For example, it may be necessary to ensure that a view exists prior to attempting to read from it.

Upon opening a connection to read from a database, the reader looks for the directive `<ReaderKeyword>_BEGIN_SQL{n}` (for $n=0,1,2,\dots$), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the `FME_SQL_DELIMITER` keyword, embedded at the beginning of the SQL block. The single character following this keyword will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;  
DELETE FROM instructors;  
DELETE FROM people  
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

* Workbench Parameter

SQL Statement to Execute Before Translation

END_SQL{n}

Occasionally you must execute some ad-hoc SQL after closing a set of tables. For example, it may be necessary to clean up a temporary view after writing to the database.

Just before closing a connection on a database, the reader looks for the directive `<ReaderKeyword>_END_SQL{n}` (for $n=0,1,2,\dots$), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the `FME_SQL_DELIMITER` directive, embedded at the beginning of the SQL block. The single character following this directive will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;  
DELETE FROM instructors;  
DELETE FROM people  
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

* Workbench Parameter

SQL Statement to Execute After Translation

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined database tables that will be read. If no **IDs** are specified, then all defined and available tables are read. The syntax of the **IDs** directive is:

```
INFSPATIAL_IDS <featureType1> \  
    <featureType2> \  
    ... \  
    <featureTypeN>
```

The feature types must match those used in **DEF** lines.

The example below selects only the **ROADS** table for input during a translation:

```
INFSPATIAL_IDS ROADS
```

RETRIEVE_ALL_SCHEMAS

Required/Optional: *Optional*

This directive is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

This optional directive is used to tell the reader to retrieve the names and the schemas of all the tables in the source database. If this value is not specified, it is assumed to be "No".

The syntax of the **RETRIEVE_ALL_SCHEMAS** directive is:

```
INFSPATIAL_RETRIEVE_ALL_SCHEMAS Yes
```

RETRIEVE_ALL_TABLE_NAMES

Required/Optional: *Optional*

This directive is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

Similar to **RETRIEVE_ALL_SCHEMAS**; this optional directive is used to tell the reader to only retrieve the table names of all the tables in the source database. If **RETRIEVE_ALL_SCHEMAS** is also set to "Yes", then **RETRIEVE_ALL_SCHEMAS** will take precedence. If this value is not specified, it is assumed to be "No".

The syntax of the **RETRIEVE_ALL_TABLE_NAMES** directive is:

```
INFSPATIAL_RETRIEVE_ALL_TABLE_NAMES Yes
```

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The Informix Spatial writer module stores both geometry and attributes into spatially-enabled Informix databases. The Informix Spatial writer provides the following capabilities:

- **Table Creation:** The Database writer uses the information within the FME mapping file to automatically create database tables as needed.
- **Coordinate System:** Checks for a coordinate system and if a matching one is not found then it will create one automatically. The matching criteria is the OGC WKT definition of the coordinate system.

Writer Directives

The directives processed by the Informix Spatial writer are listed below. The suffixes shown are prefixed by the current `<writerKeyword>` in a mapping file. By default, the `<writerKeyword>` for the Informix Spatial writer is `INFSPATIAL`.

DATASET, USER_NAME, PASSWORD, BEGIN_SQL{n}, and END_SQL{n}

The `DATASET`, `USER_NAME`, `PASSWORD`, `BEGIN_SQL{n}`, and `END_SQL{n}` directives operate in the same manner as they do for the Informix Spatial reader. The remaining writer-specific directives are discussed in the following sections.

DEF

Required/Optional: *Required*

Each Informix Spatial table must be defined before it can be written. The general form of an Informix Spatial definition statement is:

```
INFSPATIAL_DEF <tableName> \  
  [inf_type <type>] \  
  [inf_overwrite_table <YES|NO|TRUNCATE>] \  
  [inf_update_key_columns <column>[,<column>]...] \  
  [inf_delete_key_columns <column>[,<column>]...] \  
  [inf_geometry_column <geometry>] \  
  [inf_multi_geometry <YES|NO|FIRST_FEATURE>] \  
  [inf_offset_x <x offset value>] \  
  [inf_offset_y <y offset value>] \  
  [inf_scale_x <x/y scale value>] \  
  [<fieldName> <fieldType>]*
```

The table definition allows complete control of the layer that will be created. If the layer already exists, the majority of the `DEF` line parameters will be ignored and need not be given.

If the table does not exist, then the field names and types are used to first create the table. In any case, if a `<field-Type>` is given, it may be any field type supported by the target database.

The Informix Spatial writer will use the `infx_geometry_column` parameter to set the name of geometry column for the new table. If the `infx_geometry_column` parameter is not specified then a default name of “`geometry`” will be used for the geometry column.

The configuration parameters present on the definition line are described in the following table:

Parameter	Contents
infx_type	This specifies the type of geometry the features to be written to the layer will have.
infx_overwrite_table	This parameter can be set to one of <code><YES NO TRUNCATE></code> . If <code>YES</code> , then the table will be dropped and created again. If <code>TRUNCATE</code> , then all the rows from the table will be deleted. If <code>NO</code> , then data will be appended to the existing table.
infx_update_key_columns	<p>This instructs the Informix Spatial writer to perform an <code>UPDATE</code> operation on the table, rather than performing an <code>INSERT</code>. The argument is a comma-separated list of the columns which are matched against the corresponding FME attributes' values to specify which rows are to be updated with the other attribute values.</p> <p>For example: <code>infx_update_key_columns ID,NAME</code></p> <p>In this case the FME attribute is always matched against the Informix column with the same name. Also, the target table is always the feature type specified in the <code>DEF</code> line. Each column listed with the <code>infx_update_key_columns</code> directive must be defined with a type on the <code>DEF</code> line, in addition to the columns whose values will be updated by the operation. This cannot be used with <code>infx_delete_key_columns</code>. Also, the keys cannot be of type <code>BLOB</code>, <code>CLOB</code>, or <code>LONG_VARCHAR</code>.</p>
infx_delete_key_columns	<p>This instructs the Informix Spatial writer to perform a <code>DELETE</code> operation on the table, rather than performing an <code>INSERT</code>. The argument is a comma-separated list of the columns which are matched against the corresponding FME attributes' values to specify which rows are to be deleted when their values match the other attribute values.</p> <p>For example: <code>infx_delete_key_columns ID,NAME</code></p> <p>would delete those rows in the table whose values match the attribute values passed in through this <code>DEF</code> line. The FME attribute is always matched against the Informix Spatial column with the same name. Also, the target table is always the feature type specified in the <code>DEF</code> line. Each column listed with the <code>infx_delete_key_columns</code> directive must be defined with a type on the <code>DEF</code> line, in addition to the columns whose values will be updated by the operation. This cannot be used with <code>infx_update_key_columns</code>. Also, the keys cannot be of type</p>

Parameter	Contents
	BLOB, CLOB, or LONG_VARCHAR.
infx_geometry_column	This parameter can be used to specify the name of the spatial layer (geometry column name). If it is not specified the default name of "geometry" will be used for the spatial layer.
infx_multi_geometry	This specifies whether the Informix types for point, linestring and polygon should be written as multi-geometries or single geometries. If YES, the table created has multi-geometries (that is, the geometry column type will be e.g. ST_MULTIPPOINT, and the features are coerced into multi-geometries if they are not already). If NO, the geometry column of the created table is singular (that is, ST_POINT), and multi-geometries are split. FIRST_FEATURE allows this setting to be based on the first feature in the table.
infx_offset_x	The x-offset value for the dataset; defaults to 0.
infx_offset_y	The y-offset value for the dataset; defaults to 0.
infx_scale_x	The x and y scale value for the dataset; defaults to 1.

TRANSACTION_INTERVAL

This statement informs the FME about the number of features to be placed in each transaction before a transaction is committed to the database.

If the **INFXSPATIAL_TRANSACTION_INTERVAL** statement is not specified, then a value of 1000 is used as the transaction interval.

Parameter	Contents
<transaction_interval>	The number of features in a single transaction.

Default: 1000

Example:

INFXSPATIAL_TRANSACTION_INTERVAL 5000

Workbench Parameter: *Transaction Interval*

ABORT_ON_BAD_DATA

Required/Optional: *Optional*

Some features' geometries may fail Informix Spatial's check constraints based on the offset, scale, and coordinate system values. These features, as well as others with out-of-range or invalid attribute values, will be rejected and cannot be written to the database. If the value of this directive is YES then the translation will be aborted immediately after encountering such a problem. If this directive is set to NO then the translation will continue but the features with rejected geometry will not be written to the database.

Values: YES | NO

Default: NO

Example:

INFXSPATIAL_ABORT_ON_BAD_DATA YES

Workbench Parameter: *Abort Translation on Bad Data*

Feature Representation

Features read from Informix Spatial consist of a series of attribute values and geometry. The feature type of each Database feature is as defined on its **DEF** line.

Features written to the database have the destination table as their feature type, and attributes as defined by on the **DEF** line.

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), the Informix Spatial module adds the format-specific attributes described below:

Attribute Name	Contents
infx_type	The type of geometric entity stored within the feature. The valid values for the object model are listed below: infx_null infx_point infx_linestring infx_polygon infx_geometry

Features read from, or written to, Informix Spatial also have an attribute for each column in the database table. The feature attribute name will be the same as the source or destination column name. The attribute and column names are not case-sensitive.

No Coordinates

infx_type: infx_null

Features with no coordinates are tagged with this value when reading from or writing to Informix Spatial.

Points

infx_type: infx_point

All Informix Spatial point and multipoint features are read as **infx_point**. The only difference being the geometry type of feature, which will be set to **fme_aggregate** if it is a multipoint and **fme_point** if it is a point.

Lines

infx_type: infx_line

All Informix Spatial linestring and multilinestring features are read as **infx_line**. The only difference is the geometry type of feature, which will be set to **fme_aggregate** if it is a multilinestring and **fme_line** if it is a linestring.

Polygons

infx_type: infx_polygon

All Informix Spatial polygon and multipolygon features are read as **infx_polygon**. The only difference is the geometry type of feature, which will be set to **fme_aggregate** if it is a multipolygon and **fme_polygon** if it is a polygon. Polygon features include donut polygons with one or more holes.

The following table summarizes all of the infx_type values that are possible with Informix Spatial geometry, and provides a description of each representation.

infx_type	Informix Spatial type	Representation
infx_null	N/A	No geometry
infx_point	POINT	Single point geometry. fme_geometry = fme_point fme_type = fme_point
	MULTIPOINT	Aggregate containing one or more points. fme_geometry = fme_aggregate fme_type = fme_point
infx_line	LINestring	Single line geometry. fme_geometry = fme_line fme_type = fme_line
	MULTILINestring	An aggregate of linestrings. fme_geometry = fme_aggregate fme_type = fme_line
infx_polygon	POLYGON	A single polygon or donut geometry. fme_geometry = fme_polygon or fme_donut fme_type = fme_polygon
	MULTIPOLYGON	An aggregate of simple polygons or donut polygons. fme_geometry = fme_aggregate fme_type = fme_polygon
infx_geometry	GEOMETRY	An arbitrary geometry.

IDRISI Vector Format Reader/Writer

The IDRISI Vector Format Reader and Writer modules allow the Feature Manipulation Engine (FME) to read and write IDRISI vector files. IDRISI files use a published binary format. The IDRISI data and the documentation file structure are described in the *IDRISI Guide to GIS and Image Processing, Volume 1*.

Overview

IDRISI is a geographic information and image processing software system that is widely used by universities worldwide. It is a two-dimensional (2D) system which allows user-defined attributes stored in an Access database to be linked to its data file.

IDRISI features contain a feature geometry and an id number. An IDRISI file consists of at least two physical files, having the following filename extensions:

Filename Extension	Contents
.vct	Vector data file
.vdc	Vector documentation file

If an Access database is available, the following files should also be available:

Filename Extension	Contents
.vix	Vector link information file
.mdb, .accdb	Access database file
.adc	Attribute documentation file

These extensions are added to the basename of the IDRISI file.

The IDRISI reader and writer support the storage of point, line, area (polygon), and text geometric data in the **.vct** files.

The FME considers an IDRISI dataset to be a collection of IDRISI files in a single directory.

IDRISI Quick Facts

Format Type Identifier	IDRISI
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	Directory or File
Feature Type	File base name
Typical File Extensions	.vlx, .vct, .vdc (.mdb, .accdb, .adc)
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	idrisi_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	yes		z values	no
none	no			

Reader Overview

The IDRISI reader first scans the directory it is given for the IDRISI files defined in the mapping file.

For each IDRISI file that it finds, it checks to see if that file is requested by looking at the list of **IDs** specified in the mapping file. If a match is made or no **IDs** were specified in the mapping file, the IDRISI file is opened. The IDRISI reader then extracts features from the file one at a time, and passes them on to the rest of the FME for further processing. If an additional Access database is available, the user-defined attributes will also be passed to the FME. When the file is exhausted, the IDRISI reader move on to the next file in the directory.

Optionally, a single IDRISI file can be specified in the mapping file. If this is the case, only that IDRISI file is read.

Reader Directives

The suffixes listed are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the IDRISI reader is `IDRISI`.

DATASET

Required/Optional: *Required*

The value for this directive is the directory containing the IDRISI files to be read, or a single IDRISI file. A typical mapping file fragment specifying an input IDRISI dataset looks like:

```
IDRISI_DATASET /usr/data/idrisi/input
```

Workbench Parameter: *Source IDRISI Vector Format File(s)*

DEF

Required/Optional: *Required*

Each IDRISI file must be defined before it can be read. The definition specifies the base name of the file, and the names and the types of all attributes. The syntax of an IDRISI `DEF` line is:

```
<ReaderKeyword>_DEF <baseName> \  
    [<attrName> <attrType>]+
```

The following table shows the attribute types supported.

Field Type	Description
char(<width>)	Character fields store fixed-length strings. The width parameter controls the maximum number of characters that can be stored by the field. No padding is required for strings shorter than this width. For the IDRISI database, the width is limited to a maximum of 255.
date	Date fields store date as character strings with the format <code>YYYYMMDD</code> .
double	Float fields store 64-bit floating point values. There is no ability to specify the precision and width of the field.
integer	Integer fields store 32-bit signed integers.
logical	Logical fields store TRUE/FALSE data. Data read or written from and to such fields must always have a value of either true or false.

IDs

Required/Optional: *Optional*

This optional specification limits the available and defined IDRISI files read. If no `IDs` are specified, then all defined and available IDRISI files are read.

The syntax of the `IDS` directive is:

```
<ReaderKeyword>_IDS <baseName> \  
    <baseName1> ... \  
    <baseNameN>
```

The basenames must match those used in **DEF** lines.

The example below selects only the **roads** IDRISI file for input during a translation:

```
IDRISI_IDS roads
```

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The **COORDINATE_SYSTEM** directive, which specifies the coordinate system associated with the data to be read, must always be set if the **SEARCH_ENVELOPE_COORDINATE_SYSTEM** directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the **SEARCH_ENVELOPE_COORDINATE_SYSTEM** to the reader **COORDINATE_SYSTEM** prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the **SEARCH_ENVELOPE** directive.

Values

YES | NO (default)

Mapping File Syntax

<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The IDRISI writer creates and writes feature data to IDRISI files in the directory specified by the **DATASET** directive. As with the reader, the directory must exist before the translation occurs. Any old IDRISI files in the directory having the same name as files being written are overwritten with the new feature data. As features are routed to the IDRISI writer, the IDRISI writer determines the file into which the features are written to and outputs them accordingly. Many IDRISI files can be written during a single FME session.

In IDRISI, each vector file can have only one type of feature. Any feature not matching the feature type as specified in the mapping file will not be written to file. In addition, any user-defined attributes are written to the database only if an Access database is available.

Writer Directives

The directives processed by the IDRISI writer are listed below. The suffixes shown are prefixed by the current <writerKeyword> in a mapping file. By default, the <writerKeyword> for the IDRISI writer is **IDRISI**.

DATASET

Required/Optional: *Required*

The value for this directive is the directory containing the IDRISI files to be written to. A typical mapping file fragment specifying an output IDRISI dataset looks like:

```
IDRISI_DATASET /usr/data/idrisi/output  
IDEX_DATASET c:\index\map.idx
```

Workbench Parameter: *Destination IDRISI Vector Format Directory*

DEF

Required/Optional: *Required*

Each IDRISI file must be defined before it can be written. The definition specifies the base name of the file, and the names and the types of all attributes. The syntax of a IDRISI **DEF** line is:

```
<WriterKeyword>_DEF <baseName> \  
    [<attrName> <attrType>]+
```

The attribute types supported are the same as those listed under the reader section.

The following table shows the **DEF** line directives that are supported by IDRISI. They are prefixed by the keyword "**IDRISI_**".

Keyword Suffix	Value	Required/Optional
TYPE	The feature type of the file. In IDRISI, each file can contain only one feature type. The allowable types are idrisi_point , idrisi_line , idrisi_area and idrisi_text .	Required
FILE_TITLE	Contains the descriptive name of the file. It is the name which is displayed at the top of the data file. If not specified, this defaults to "unknown".	Optional
REF_SYSTEM	The name of the geographic reference system used with the file. This may be Plane, or Lat/Long and so forth. If not specified, this defaults to "unknown".	Optional
REF_UNITS	The unit of measure used by the reference system. The recognized units are meters, feet, miles, kilometers, degrees and radians. This defaults to "meters" if not specified.	Optional
UNIT_DIST	The scaling factor of the map in relation to the ground. This should be 1 in most cases and this defaults to 1 if not specified.	Optional
POSN_ERROR	The degree of accuracy of the position of the feature in the vector file. If not specified, this defaults to "unknown".	Optional
RESOLUTION	The typical distance between points of a feature in the vector file. If not specified, this defaults to "unknown".	Optional
VALUE_UNITS	The unit of measure of the values used in the vector file. If not specified, this defaults to "unspecified".	Optional
VALUE_ERROR	The degree of error in the data values. For qualitative data, this is recorded as a proportional error value. For quantitative data,	Optional

Keyword Suffix	Value	Required/Optional
	this should be recorded as a RMS error value.	
FLAG_VALUE	Any value in the vector file, which is not a data value, but has special meaning. This entry should remain blank if such value does not exist.	Optional
FLAG_DEF	The definition of the above FLAG_VALUE. This field should remain blank if FLAG_VALUE does not exist.	Optional
LEGEND_CAT {<number>}	The legend categories which shows up in the legend box when the vector file is displayed. The <number> can be any positive integer which is larger than 0.	Optional
COMMENT {<number>}	Any additional information about the data. The <number> starts from 1 and increments by one for each additional comment line.	Optional
LINEAGE {<number>}	Any information regarding the history of how the data is recorded. The <number> starts from 1 and increments by one for each additional lineage line.	Optional
completeness {<number>}	The degree of how well the values describe the subject matter indicated. The <number> starts from 1 and increments by one for each additional lineage line.	Optional
consistency {<number>}	The logical consistency of the file. The <number> starts from 1 and increments by one for each additional consistency line.	Optional

The following mapping file fragment defines two IDRISI files. Note that all but the **IDRISI_TYPE** are optional. There is no need to specify all of them.

The first file provides a simple example as to how some of the **DEF** lines are used. The second file gives an example which has two additional user defined attributes. Note that user-defined attributes are only allowed on Microsoft Windows platforms.

```
IDRISI_DEF landcover \
IDRISI_FILE_TITLE "ExampleFileTitle1" \
IDRISI_TYPE idrisi_area \
IDRISI_REF_SYSTEM "ExampleRefSystem" \
IDRISI_REF_UNITS kilometers \
IDRISI_LEGEND_CAT{-20} "ExampleLegendCat-20" \
IDRISI_LEGEND_CAT{1} "ExampleLegendCat1" \
IDRISI_LEGEND_CAT{-1} "ExampleLegendCat-1" \
IDRISI_COMMENT{1} "ExampleComment1" \
IDRISI_COMMENT{2} "ExampleComment2" \
```

```
IDRISI_COMMENT{3} "ExampleComment3"

IDRISI_DEF roads
IDRISI_FILE_TITLE "ExampleFileTitle2" \
IDRISI_TYPE idrisi_line \
IDRISI_REF_SYSTEM "ExampleRefSystem" \
IDRISI_REF_UNITS kilometers \
IDRISI_LEGEND_CAT{-20} "ExampleLegendCat-20" \
IDRISI_LEGEND_CAT{1} "ExampleLegendCat1" \
IDRISI_LEGEND_CAT{-1} "ExampleLegendCat-1" \
color char(20) \
style logical
```

COMPRESS_AT_END

Required/Optional: *Optional*

This statement instructs FME to compact the database after all writing has been done. This makes use of the existing MDB database option to compact. The compact operation compresses the output database to a small size on disk.

Example:

```
COMPRESS_AT_END Yes
```

Workbench Parameter: *Compress Database When Done*

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

IDRISI features consist of geometry and attributes. The attribute names are defined in the DEF line and there is a value for each attribute in each IDRISI feature. In addition, each IDRISI feature contains several special attributes to hold the type of the geometric entity and its display parameters. All IDRISI features contain an **idrisi_type** attribute, which identifies the geometric type. Depending on the geometric type, the feature contains additional attributes specific to the geometric type. These are described in subsequent sections. All features contain an **idrisi_id**.

Points

idrisi_type: idrisi_point

IDRISI point features specify a coordinate in addition to its ID value. There are no attributes specific to this type.

Attribute Name	Contents
idrisi_type	The IDRISI geometric type of this entity. Range: idrisi_point idrisi_line idrisi_area idrisi_text Default: No default
idrisi_id	The ID value of the symbol. IDRISI IDs are used to link vector graphics with user-defined attributes. Range: Any real > 0. Default: 1 and increments by 1 for each additional feature

Lines

idrisi_type: idrisi_line

IDRISI line features specify linear features defined by an array of x and y coordinates. There are no attributes specific to only this type of element.

Regions

idrisi_type: idrisi_area

IDRISI area features specify area (polygonal) features. The areas that make up a single feature may or may not be disjoint, and may contain polygons that have holes. The first and last coordinates must be the same in order for it to be a region.

There are no attributes specific to this type of element.

Text

idrisi_type: idrisi_text

IDRISI text features are used to specify annotation information. Each text feature has a location defined by a single point geometry, and can have its text string, style, justification, and rotation angle set independently.

The following table lists the special FME attribute names used to control the IDRISI text settings.

Attribute Name	Contents
idrisi_text_string	This is the text string that is the label for the feature. Range: 1 - 256 characters Default: Empty string
idrisi_text_size	This the font size of the label for the feature. Range: > 0 Default: 1/10th of the height of the map or 10
idrisi_style	The style code of the text string. This controls the color of the text string. Range: Any integer > 0 Default: 1
idrisi_rotation	The rotation of the text, as measured in degrees counterclockwise from the horizontal. Range: 0.0 - 360.0 Default: 0.0
idrisi_justification_x	The justification of the text in the X direction. Values approaching 1.00 shift the text to the left of the location point, and values the closer to 0.00 shift the text to the right. Range: 0.00 - 1.00 0.00 Left margin 0.50 Centre 1.00 Right margin Default: 0.00

Attribute Name	Contents
idrissi_justification_y	<p>The justification of the text in the Y direction. Values approaching 1.00 shift the text upward relative to the location point, and values the closer to 0.00 shift the text downward.</p> <p>Range: 0.00 - 1.00 0.00 Bottom 0.50 Centre 1.00 Top</p> <p>Default: 0.00</p>

Industry Foundation Class STEP Files (IFC) Reader

The IFC Reader allows the Feature Manipulation Engine (FME) to read Industry Foundation Classes (IFC) Standard for the Exchange of Product model data Files (STEP-Files). IFC is a vendor neutral Building Information Modeling data model and the ISO 10303 STEP-File standard is its primary exchange format.

The FME format keyword is IFC.

Overview

The IFC specification is promoted and published by International Alliance for Interoperability (IAI). The IFC model specification is published in the ISO 10303 EXPRESS data modelling language. Data is exchanged through the ISO 10303 STEP-File plain-text format.

The IFC Reader module support IFC specification version 2x, 2x2, and 2x3.

An IFC dataset is a collection of entity instances of entity classes. All class instances that descend from *IfcRoot* are mapped to features in FME. There are three fundamental class types in the IFC model:

- Descendants of *IfcObject* stands for all physically tangible and existing items, and conceptual items, such as processes and resources. *IfcProduct* is a subtype of *IfcObject* and descendants of *IfcProduct* are the only classes that may have a geometric representation.
- Descendants of *IfcRelationship* describe relationships between objects.
- Properties are descendants of *IfcPropertyDefinition* and are characteristics that may be assigned to objects.

Instances are uniquely identified by an instance name inside a STEP-File dataset. Instance names appear in attributes that reference other instances.

The attributes of IFC classes are fixed. However, objects can be extended by properties. Properties may have predefined structures defined by the IFC model, or they can be dynamically defined inside a dataset.

A collection of properties forms a property set, and the set is assigned to an object through a relationship instance. In FME, dynamic property set instances are identified by the *Name* attribute, and they form feature types according to this attribute. Properties assigned to these dynamic property sets appear as attributes of feature types.

IFC Quick Facts

Format Type Identifier	IFC
Reader/Writer	Reader
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	IfcRoot classes
Typical File Extensions	IFC
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	Never
Enhanced Geometry	Yes
Geometry Type	ifc_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	yes		polygon	yes
circular arc	yes		raster	no
donut polygon	yes		solid	yes
elliptical arc	yes		surface	yes
ellipses	yes		text	yes
line	yes		z values	yes
none	yes			

Reader Overview

The IFC reader will automatically detect the IFC specification version of the data file by analyzing its header.

To generate the source schema, the IFC Reader will scan through the source file. Each non-abstract entity descendant from IfcRoot will appear as a source feature type. The source schema also has feature types for history and identification related entities: IfcOwnerHistory, IfcPerson, IfcApplication, IfcPersonAndOrganization, and IfcOrganization. The source schema will only contain feature types for entities that are instantiated in the source data file.

Reader Directives

The directives that are processed by the IFC reader are listed below. The suffixes shown are prefixed by the current `<ReaderKeyword>_` in a mapping file. By default, the `<ReaderKeyword>` for the IFC reader is `IFC`.

DATASET

Required/Optional: *Required*

The value for this directive is the path to the source data file. If the data file does not exist or if the file is unrecognized by the reader, then the process will fail.

SPLIT_REPRESENTATIONS

Required/Optional: *Optional*

This directive specifies whether IfcProduct objects associated with multiple IfcShapeRepresentation objects will be read as a single FME feature. If the value is `NO` and the IfcProduct object is associated with multiple IfcShapeRepresentation objects, then the geometry of the feature will be a collection of all the geometric representations. If the value is `YES`, then the IfcProduct will be split among multiple FME features with each feature geometrically represented by a single IfcShapeRepresentation object. All the split features will have the same attributes that are on the IfcProduct object. The default value of this directive is `YES`.

Default value:

IFC_SPLIT_REPRESENTATIONS YES

Workbench Parameter: *Split multiple representations*

SUBTRACT_OPENINGS

Required/Optional: *Optional*

This directive specifies whether the reader will subtract IfcOpeningElement representations from IfcProduct representations that are related together by an IfcRelVoidsElement object. If the value is `YES` then the IfcOpeningElement objects will have no geometry, and IfcProduct representations will have openings as determined by the IfcRelVoidsElement relationship. If the value is `NO` then the opening will not be calculated, and IfcOpeningElement objects will retain their representations. The default value of this directive is `YES`.

IFC_SUBTRACT_OPENINGS YES

Workbench Parameter: *Subtract Openings*

IFCSPACE_GEOMETRY

Required/Optional: *Optional*

This directive specifies whether the reader will preserve or remove the representations of IfcSpace features. IfcSpace geometries are virtual areas or volumes that provide for certain functions within a building. When physical entities are most important, visualizing these volumes of space may not be desirable.

If the value is `YES` then IfcSpace features will have their defined representations. If the value is `NO`, then IfcSpace features will have no geometry. The default value for this directive is `NO`.

IFC_IFCSPACE_GEOMETRY NO

Workbench Parameter: *Read IfcSpace geometries*

CONTEXT_TYPES

Required/Optional: *Optional*

This directive specifies which geometric representation will be processed by the reader according to the associated IfcRepresentationContext object's ContextType attribute. If this directive is not specified, then all representations will be processed. The format for values for this directive is a comma-delimited list of ContextType values. If at least one ContextType value is specified for this directive, then the reader will only process the representations that are associated with IfcRepresentationContext objects that have ContextType values that appear in the list specified for this directive. However, if a '!' character appears by itself in the comma-delimited string, then the reader will not process

the representations associated with representation contexts that matches a value in the list. If the only value for this directive is '!', then all context types will be processed.

For example, this mapping file statement will direct the reader to only read geometric representations that are associated with 'Design' and 'Sketch' representation contexts:

```
IFC_CONTEXT_TYPES Design,Sketch
```

This mapping file statement will direct the reader to read all geometric representations that are not associated with the 'Sketch' representation context:

```
IFC_CONTEXT_TYPES !,Sketch
```

Workbench Parameter: *Representation context types to read*

REPRESENTATION_IDENTIFIERS

Required/Optional: *Optional*

This directive specifies which geometric representation will be processed by the reader according to the IfcRepresentation object's RepresentationIdentifier attribute. If this directive is not specified, then all representations will be processed. The format for values for this directive is a comma-delimited list of RepresentationIdentifier values. If at least one RepresentationIdentifier value is specified for this directive, then the reader will only process the representations that have RepresentationIdentifier values that appear in the list specified for this directive. However, if a '!' character appears by itself in the comma-delimited string, then the reader will not process the representations that matches a value in the list. If the only value for this directive is '!', then all representation identifiers will be processed.

For example, this mapping file statement will direct the reader to only read geometric representations with 'Axis' and 'Body' representation identifiers:

```
IFC_REPRESENTATION_IDENTIFIERS Axis,Body
```

This mapping file statement will direct the reader to read all geometric representations except for the ones with an 'Axis' representation identifier:

```
IFC_REPRESENTATION_IDENTIFIERS !,Axis
```

Workbench Parameter: *Representation identifiers to read*

REPRESENTATION_TYPES

Required/Optional: *Optional*

This directive specifies which geometric representation will be processed by the reader according to the IfcRepresentation object's RepresentationType attribute. If this directive is not specified, then all representations except for the BoundingBox representation type will be processed. The format for values for this directive is a comma-delimited list of RepresentationType values. If at least one RepresentationType value is specified for this directive, then the reader will only process the representations that have RepresentationType values that appear in the list specified for this directive. However, if a '!' character appears by itself in the comma-delimited string, then the reader will not process the representations that matches a value in the list. If the only value for this directive is '!', then all representation types will be processed.

Representations can have multiple representation types: specifically, the MappedRepresentation type can coexist with other representation types. In this case, all representation types applicable to the representation must be specified for that representation to be read.

For example, this mapping file statement will direct the reader to only read geometric representations with 'Brep' and 'SweptSolid' representation types:

```
IFC_REPRESENTATION_TYPES Brep,SweptSolid
```

This mapping file statement will direct the reader to read all geometric representations except for the ones with an 'BoundingBox' representation type. This is the default value for this directive and this value will be used if the directive was not specified in the mapping file:

```
IFC_REPRESENTATION_TYPES !,BoundingBox
```

Workbench Parameter: *Representation types to read*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Feature Representation

IFC features will have attributes that corresponds to the explicit attributes listed in the EXPRESS entity definitions. Derived attributes and inverse attributes will not appear on features. However, some inverse attributes are mapped to format specific attributes for convenience and are listed in the following table.

For IfcProduct objects, the attributes ObjectPlacement and Representation are mapped to the format specific attributes ifc_object_placement and ifc_representation, respectively.

Attributes with attribute type instance are references to other instances. These values will always start with a '#' character followed by a number. They will match the value of the ifc_instance_name attribute of some feature.

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the following format-specific attributes:

Attribute Name	Contents
ifc_instance_name	This attribute contains the unique instance name that corresponds to the feature.
ifc_entity_type	This attribute contains the name of the entity of which the feature is an instance. The value is equivalent to the feature type name for most features.
ifc_representation_identifier	This attribute contains the value of the RepresentationIdentifier attribute of the IfcRepresentation object by which this feature is represented. This attribute is only applicable when the feature has only one representation. (See the ifc_representations attribute)
ifc_context_type	This attribute contains the value of the ContextType attribute of the IfcRepresentationContext object with which the

Attribute Name	Contents
	feature's representation is associated. This attribute is only applicable when the feature has only one representation. (See the <code>ifc_representations</code> attribute)
<code>ifc_context_dimension</code>	This attribute contains the value of the <code>CoordinateSpaceDimension</code> attribute of the <code>IfcGeometricRepresentationContext</code> object with which the feature's representation is associated.
<code>ifc_context_precision</code>	This attribute contains the value of the <code>Precision</code> attribute of the <code>IfcGeometricRepresentationContext</code> object with which the feature's representation is associated.
<code>ifc_is_decomposed_by{}</code>	This list attribute contains the instance names of objects that decomposes the feature according to an <code>IfcRelDecomposes</code> relationship instance.
<code>ifc_decomposes</code>	This attribute contains the instance name of the object that the feature decomposes according to an <code>IfcRelDecomposes</code> relationship instance.
<code>ifc_contains{}</code>	This list attribute contains the instance names of objects that is spatially contained in the feature according to an <code>IfcRelContainedInSpatialStructure</code> relationship instance.
<code>ifc_is_contained_in</code>	This attribute contains the instance name of the object in which the feature is spatially contained according to an <code>IfcRelContainedInSpatialStructure</code> relationship instance.
<code>ifc_representations{}</code> . type	This structured list attribute contains the IFC geometry type of each representation associated with the feature.
<code>ifc_representations{}</code> . identifier	This structured list attribute contains the representation identifier (see <code>ifc_representation_identifier</code>) of each representation associated with the feature.
<code>ifc_object_placement</code>	This attribute contains the instance name of the <code>IfcObjectPlacement</code> object which determines the placement of the feature's representation in world coordinates. The IFC Reader will have processed this object when generating the geometry for the feature.
<code>ifc_representation</code>	This attribute contains the instance name of the <code>IfcRepresentation</code> object which determines the representation of the feature. The IFC Reader will have processed this object when generating the geometry for the feature.

Material Associations

If the feature is associated with an IfcMaterial object through an IfcRelAssociatesMaterial relationship object, then the feature will have the following attributes according to the type of material description.

Association of a single material

Attribute Name	Contents
ifc_material.name	A string value of the name of the material.

Association of a list of materials

Attribute Name	Contents
ifc_material.material{}.name	A string value of the name of a material in the material list.

Association of material layers

Attribute Name	Contents
ifc_material.layer{}.name	A string value of the name of a material layer in the layer set.
ifc_material.layer{}.thickness	A real number value of the thickness of a material layer in the layer set.
ifc_material.layer{}.is_ventilated	A boolean value of either 'T' or 'F' that determines whether the material layer is ventilated.
ifc_material.layer{}.layer_set_name	An optional string value of the name of the layer set.

Property Sets

If the feature is a property set object, then its attribute will be IfcProperty objects. The IfcProperty.Name attribute determines the name of the attribute on the feature. The encoding of various types of IfcProperty classes to feature attributes is described in the following section:

IfcPropertySingleValue

The value of the attribute on the feature is the value of the IfcPropertySingleValue.NominalValue attribute of the instance.

IfcPropertyEnumeratedValue

The value of the attribute on the feature is the comma-delimited string of the multiple values of the IfcPropertyEnumeratedValue.EnumerationValues attribute of the instance.

IfcPropertyBoundedValue

The value of the attribute on the feature is the values of the IfcPropertyBoundedValue attributes LowerBoundValue and UpperBoundValue of the instance arranged in the form '<LowerBoundValue>,<UpperBoundValue>'.

IfcPropertyListValue

The value of the attribute on the feature is the comma-delimited string of the multiple values of the IfcPropertyListValue.ListValues attribute of the instance.

IfcComplexProperty

The IfcComplexProperty object contains multiple IfcProperty objects. Each of the IfcProperty objects will be encoded by the method described above, but their attribute names on the feature will be prefixed by the value of the IfcProperty.Name attribute of the IfcComplexProperty instance.

The following example shows a IFC data file snippet and the resulting FME feature schema:

```
#1=IFCPROPERTYSET($,$,'Example','',(#2));
#2=IFCCOMPLEXPROPERTY('Color',$,'color',(#3,#4,#5));
#3=IFCPROPERTYSINGLEVALUE('Red',$,IFCINTEGER(0),$);
#4=IFCPROPERTYSINGLEVALUE('Green',$,IFCINTEGER(0),$);
#5=IFCPROPERTYSINGLEVALUE('Blue',$,IFCINTEGER(0),$);

+++++
Feature Type: Example'
Attribute(string)      : Color.Red' has value 0'
Attribute(string)      : Color.Green' has value 0'
Attribute(string)      : Color.Blue' has value 0'
=====
```

IfcConnectionGeometry

IfcConnectionGeometry objects describe the geometric and topological constraints that facilitate the connection of two objects. IfcConnectionGeometry objects connect geometries of features, and such geometries with connections will have the following geometry traits:

Attribute Name	Contents
ifc_connects_relating_element	A comma delimited list of instance names of instances that are relate to the geometry.
ifc_connects_related_element	A comma delimited list of instance names of instances to which the geometry relates.

Geometry Representation

The geometry of IFC features is identified by the **ifc_type** attribute. The valid values for this attribute are listed and described in the following sections.

Points

ifc_type: ifc_point

The geometry is represented by a single point.

Lines

ifc_type: ifc_line

The geometry is represented by a path. Path segments can be linear or elliptical.

Currently, IfcTrimmedCurve is not supported.

These attributes are common to features with ifc_type of ifc_line, but the attributes may also appear in features with different ifc_type values.

Attribute Name	Contents
ifc_curve_style_name	This attribute specifies the name of the specific curve style assigned to the geometry.
ifc_curve_style_width	This attribute specifies the width of the curve.

Attribute Name	Contents
ifc_curve_style_color	This attribute specifies the color of the curve as a string in the format "r,g,b" where 'r', 'g', and 'b' are the red, green, and blue components respectively specified in the interval between 0 and 1.
ifc_curve_style_color.name	This attribute specifies the name of the color assigned to the curve.
ifc_curve_style_font_name	This attribute specifies the name of the curve font.
ifc_curve_style_font_pattern.visible	This attribute specifies the length of the visible portion of the curve. The curve will alternate between visible and invisible segments accordingly.
ifc_curve_style_font_pattern.invisible	This attribute specifies the length of the invisible portion of the curve. The curve will alternate between visible and invisible segments accordingly.
ifc_curve_style_font_scaling	This attribute specifies the scaling factor applied to the curve font.

Polygon

ifc_type: ifc_polygon

The geometry is represented by an enclosed area.

These attributes are common to features with ifc_type of ifc_polygon, but the attributes also may appear in features with different ifc_type values.

Attribute Name	Contents
ifc_fill_area_style_name	This attribute specifies the name of the specific area fill style assigned to the geometry.
ifc_fill_area_style_color	This attribute specifies the color of the area fill as a string in the format "r,g,b" where 'r', 'g', and 'b' are the red, green, and blue components respectively specified in the interval between 0 and 1.
ifc_fill_area_style_color.name	This attribute specifies the name of the color assigned to the fill area.
ifc_fill_area_style_external_reference.location	This attribute specifies the location of the external resource item.
ifc_fill_area_style_external_reference.item_reference	This attribute specifies the external resource identifier.
ifc_fill_area_style_external_reference.name	This attribute specifies the optional name of the external resource item.

Surface

ifc_type: ifc_surface

The geometry is represented by a 3D surface. The surface may consist of multiple faces. Faces in IFC are double-sided; as a result, back-face culling should not be performed on IFC data.

These attributes are common to features with ifc_type of ifc_surface or ifc_solid, but the attributes may also appear in features with different ifc_type values.

Attribute Name	Contents
ifc_surface_style_name	This attribute specifies the name of the specific surface style assigned to the geometry.
ifc_surface_style_side	This attribute specifies the side of the surface geometry to which this style is assigned.
ifc_surface_style_shading_color	This attribute specifies the shading color of the surface.
ifc_surface_style_diffuse_transmission_color	This attribute specifies the diffuse transmission color of the surface.
ifc_surface_style_diffuse_reflection_color	This attribute specifies the diffuse reflection color of the surface.
ifc_surface_style_transmission_color	This attribute specifies the transmission color of the surface.
ifc_surface_style_reflectance_color	This attribute specifies the reflectance color of the surface.

Solid

ifc_type: ifc_solid

The geometry is represented by a 3D solid. The outer boundary of the solid is represented by an enclosed surface. Faces in IFC are double-sided; as a result, back-face culling should not be performed on IFC data.

Currently, for SweptSolid geometries, only IfcExtrudedAreaSolid is supported. The IfcRelVoidsElement relationship is not supported: IfcWall geometries may not have proper recesses and openings.

Text

ifc_type: ifc_text

The geometry is represented by a text literal.

These attributes are common to features with ifc_type of ifc_text, but the attributes may also appear in features with different ifc_type values.

Attribute Name	Contents
ifc_text_style_name	This attribute specifies the name of the specific text style assigned to the geometry.
ifc_text_style_font_name	This attribute specifies the name of the text font.
ifc_text_style_external_reference	This attribute specifies an externally defined text style.
ifc_text_style_foreground_color	This attribute specifies the foreground color of the text.
ifc_text_style_background_color	This attribute specifies the background color of the text.

Bounding Box

ifc_type: ifc_bounding_box

The geometry is represented by a 3D box which represents the bounding box of the physical representation of the feature. With the default setting for the `REPRESENTATION_TYPES` directive, bounding boxes will not be output by the IFC Reader.

Collection

ifc_type: ifc_collection

The geometry is represented by collection of geometry. Each member of the collection may have any geometric representation. This geometry type most frequently arises when the IFC feature has multiple geometric representations.

Intergraph GeoMedia Access and SQL Server Warehouse Reader/Writer

Format Notes: This format is not supported by FME Base Edition.

The GeoMedia Access Warehouse Reader/Writer module provides the Feature Manipulation Engine (FME) with access to Intergraph GeoMedia Access and SQL Server Warehouses, which store both spatial and attribute data.

Note: The following information applies both to Access and SQL Server warehouse reading and writing unless otherwise specified.

FM0 Quick Facts

Format Type Identifier	FM0
Reader/Writer	Both
Licensing Level	Professional
Dependencies	Writer: GeoMedia installed
Dataset Type	File
Feature Type	Table name
Typical File Extensions	.mdb, .accdb
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Optional
Schema Required	Yes
Transaction Support	No
Enhanced Geometry	Yes
Geometry Type	fm0_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	no		polygon	yes
circular arc	yes		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	yes
line	yes		z values	yes
none	yes			

FM0_SQL Quick Facts

Format Type Identifier	FM0_SQL
Reader/Writer	Reader
Licensing Level	Professional
Dependencies	None
Dataset Type	Database
Feature Type	Table name
Typical File Extensions	N/A
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Optional
Schema Required	Yes
Transaction Support	No
Enhanced Geometry	Yes
Geometry Type	fm0_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	no		polygon	yes
circular arc	yes		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	yes
line	yes		z values	yes
none	yes			

Overview

GeoMedia warehouses store both geometry and attributes for features in the form of columns within the tables of a database. Tables can be divided into two groups based on content. The first group contains meta-information about the formatting of the data, including coordinate systems, tables aliases, modification logs, a data table list and a field level index. The second group is the list of tables that actually contain the geometry features and their attributes. For example, a single Microsoft Access **.mdb** or **.accdb** file or a single SQL Server database contains all the necessary information for an image.

In order to retrieve information from a GeoMedia warehouse, an Open DataBase Connectivity (ODBC) source must be set up. Depending on the source dataset format, users can specify a filename, database name or a valid existing ODBC data source name. If the source format is of type "GeoMedia Access warehouse" then either a filename or DSN can be used. If the source format is of type "GeoMedia SQL Server warehouse" then either a database name with associated parameters or DSN can be used. Once the GeoMedia Reader has all the required information, it then dynamically creates a temporary ODBC source (when a filename or database name is supplied) to connect to that database.

Point, line, area, arc, and text primitive geometric data can be stored in the tables produced by GeoMedia, as well as composites (aggregates), boundaries (donuts) and collections (aggregates) of those types. A given data table holding multiple types of geometries can be read by the reader, but the writer only produces tables containing one specific geometry type each, including boundaries, composites or collections of that type. The result is that one table containing many types of features may be converted into several tables – one for each type of feature by the writer. This is especially true of the collection type in GeoMedia which is very generally a collection of any primitive type and has no corresponding equivalent in FME. Syntactically, the type is simply appended to the table name separated by an underscore (for example, `tableofManyTypes_area`).

The key table for determining the names of the spatial tables is called the **GFeatures** table. This table contains the list of names of the geometry and attribute data tables. This information can also be retrieved from the **Field-Lookup** table which also contains the specific fields of each geometry table. Before either of these lookups can happen, however, you have to find the **GFeatures** and **FieldLookup** tables. Table names can be aliased and there is only one table that must have a constant name (the **GAliasTable**). From here, you can look up the given names of the other metadata tables in the specific database you are viewing.

GeoMedia warehouses hold three-dimensional geometries. A geometry table may contain any mix of attributes the user has specified, but must contain a column containing the actual geometry object. This column is a blob type and is simply an encoded block of binary data. Each geometry blob type is encoded in a unique way and varies in length.

GeoMedia can store text in two variations: plain text and rich text. Since FME supports plain text only, the GeoMedia reader will convert all rich text to plain text and set the text size to either the default (1 ground unit) or to the user-supplied size in ground units using the **TEXT_SIZE_GROUND_UNITS** keyword. If the text being read is in rich text format, it also sets the attribute `fm0_rtf_text_string` to the original formatted string.

The GeoMedia writer will first check to see whether or not the attribute `fm0_rtf_text_string` is set. If it is set, then the formatted string will be used to write out as rich text. If the attribute is not set, then the GeoMedia writer by default will write plain text unless the **PLAIN_TEXT** keyword is set to **NO**. In this case, the GeoMedia writer will write rich text using either a default font size of 10 or a user-supplied font size using **FONT_SIZE** keyword. Allowed font size is between 1 to 1024 inclusive. At this time the font size for rich text is the only supported style for writing.

Reading is performed by parsing the tables and respective binary BLOBs directly from the Microsoft Access database or the SQL Server database. Therefore, the GeoMedia Access Warehouse Reader does not require GeoMedia to be installed in order to run. The GeoMedia Access Warehouse Writer, however, uses the GeoMedia COM objects to create and write the tables and BLOBs, and therefore cannot be used without a licensed GeoMedia installation. The GeoMedia SQL Server Warehouse Reader/Writer does not require the installation of GeoMedia but does require access to Microsoft SQL Server.

Coordinate system support exists for both reading and writing: in the best case, a GeoMedia warehouse that contains a defined coordinate system can be read and converted to any of the supported writer formats in FME which also support coordinate systems, with the same coordinate system name. If the coordinate system is not specifically identified to have the exact same defined name within FME, the attributes of the coordinate system are still transferred, producing an identical coordinate system in all but name. The reverse is also true: the writer can interpret any given FME coordinate system and convert it to either a named GeoMedia coordinate system or an equivalent created coordinate system.

One issue involves the type of projection used in a coordinate system definition which GeoMedia calls the Base Storage Type. This type can be set to one of projected, geographic or geocentric. Projected types are the usual case and are handled normally by FME, though it is notable that the storage center values from the GeoMedia coordinate system become built into the coordinates and are cleared in translated warehouses. All geographic types are represented in the Lat/Long coordinate system but remain identical in appearance. The storage center values here will represent how to convert the coordinates to radians since they will be provided in degrees, as is consistent with the way GeoMedia itself handles geographic types. Finally, the geocentric type is not supported by FME.

Native spatial indexing in GeoMedia is supported by both the reader and the writer. However some conditions apply. The reader will preserve spatial indexes read from a GeoMedia warehouse and a writer will automatically create new indexes (when creating new feature tables) based on the data, as long as the following is true:

The **CREATE_SPATIAL_INDEX** keyword is not set to **NO**,

The registry key **HKEY_LOCAL_MACHINE\SOFTWARE\Intergraph\Applications** has a string value called **DefaultJCache** which must be set to the correct version of GeoMedia (for example "GeoMedia Professional_04.00"), and

The **autodt.ini** file must contain a valid datum mapping between the current coordinate system datum and the WGS84 datum. (See the file for more detail. It is probably located in a directory such as **C:\Program Files\GeoMedia Professional\Program\cssruntm\cfg\autodt.ini**).

When appending to an existing warehouse, the creation of a Spatial Index depends on whether the **SpatialKeyFieldName** property is set for the geometry column and the column itself exists. If the property is set and the column exists, then a spatial key will be automatically created for the geometry, regardless of the **CREATE_SPATIAL_INDEX** keyword setting.

For the most part, spatial index creation should happen automatically for most known coordinate systems since the default for the writer creating them is **yes**, and the registry key mentioned above should be set when GeoMedia is installed. Additionally, data tables can be created with either primary or secondary indexes by the GeoMedia Access Warehouse Writer.

When translations are run with enhanced geometry handling turned ON, it enables the Geomedia and Geomedia SQL readers to read complex geometries like paths, and enables the FME to store them. The Geomedia and Geomedia SQL writers have been enabled to write FME features with enhanced geometries as well. Altogether, the addition of the enhanced geometry model support increases accuracy of geometric representation in Geomedia-to-Geomedia translations, as well as the creation and interpretation of more accurate features when translating to or from other FME formats

Reader Overview

The GeoMedia Warehouse Reader produces FME features for all data held in the Microsoft Access **.mdb** or **.accdb** files or an MS SQL Server database, with the exception of image (coverage) data. The reader opens the connection to the source dataset and reads the **GAliasTable** to determine the proper table names to be used. Next it reads the table of **GFeaturesTable** type to determine the list of tables that contain geometry data. This list of data tables to be read is modified by the specified **ID** and **DEF** lines specified in the mapping file or on the command line. Each geometry table is then read and its features are processed and returned one at a time. When a table is exhausted, the reader starts on the next data table in the list until all tables are read. Issues with reading in a table may result in a specific feature being skipped and sometimes an entire table depending on the severity of the error, but the reader will always try to perform as much translation as possible.

Geometries from GeoMedia do not map exactly to FME geometries. This will have the following effects on the resulting FME features:

- Collections map to one aggregate feature for each FME **fm0_type** depending on which types exist in the collection.
- Multilevel composites may be flattened out to simpler first- or second-level nesting.
- Because GeoMedia is not strict in its typing, the reader can produce some nonsensical features that may be skipped, e.g., a line aggregate containing points.

Reader Directives – all GeoMedia Warehouses

DATASET

Required/Optional: *Required*

The value of this keyword depends on the format of the source dataset. For GeoMedia Access warehouse, it is either the filename of MS Access database or an existing ODBC data source name; for GeoMedia SQL Server warehouse, it is either the database name or an existing ODBC data source name.

When specifying data source name for GeoMedia SQL Server warehouse, a username and password is also required. For GeoMedia Access warehouse, a password is required only if the source warehouse is password-protected.

Range: Valid File Name or ODBC source for Access warehouses

Default: None

Example:

A typical mapping file fragment specifying an input GeoMedia dataset looks like:

```
FM0_DATASET D:\warehouses\featuredata.mdb
```

or

```
FM0_DATASET Access_ODBC_Source
```

```
FM0_SQL_DATASET myDatabase
```

or

```
FM0_SQL_DATASET sql_Server_ODBC_Source
```

Workbench Parameter: *Source Intergraph GeoMedia Access Warehouse File(s) or Source Intergraph GeoMedia SQL Server Warehouse Dataset*

DEF

Required/Optional: *Optional*

Each GeoMedia table may optionally be defined before it is read. The definition specifies the name of the table, the type of geometry on each row, the names and types of all attributes and possibly an optional **WHERE** clause, or even an entire SQL statement with which to query the table. The syntax of a GeoMedia **DEF** thus may appear in one of two forms.

The first form allows specification of a **WHERE** clause:

```
<ReaderKeyword>_DEF <tableName> \  
[SQL_WHERE_CLAUSE <whereClause>] \  
    FM0_GEOMETRY fm0_point|fm0_arc|fm0_line|fm0_area|  
    fm0_text|fm0_none \  
    [<attrName> <attrType>]+
```

Note that the Reader uses only the **tableName** and possible SQL modifications, ignoring the other geometry and attribute information, which is relevant only to the writer. An example of an SQL **WHERE** clause that could be used is

```
FM0_DEF States \  
    SQL_WHERE_CLAUSE "id > 35"\  
    fm0_type fm0_area \  
    id integer \  
    name char(20)
```

Note: The value for SQL_WHERE_CLAUSE should always be enclosed within double quotation marks when specified on DEF lines.

where **"id"** is a column of the table **"States"**. The user is responsible for ensuring the column is contained in the table, and this **WHERE** clause is only appended to the SQL statement **select * from <tableName>**. The word **WHERE** is not included in the **WHERE** clause, but is appended automatically when a clause is specified.

The second form of a DEF line involves specification of an entire SQL statement:

```
<ReaderKeyword>_DEF <tableName> \  
[SQL_STATEMENT <sqlStatement>] \  
    FM0_GEOMETRY fm0_point|fm0_arc|fm0_line|fm0_area|  
    fm0_text|fm0_none \  
    [<attrName> <attrType>]+
```

The specified SQL statement is used to place the query against the database. It is again the responsibility of the user to ensure its correspondence with the correct table and columns. An example SQL statement might be

```
SELECT GEOMETRY FROM POINTS WHERE ID=0
```

The following table shows the attribute types that are supported.

Field Type	Description
char(<length>)	Character fields store fixed-length strings. The length parameter controls the maximum characters that can be stored by the field. When a character field is written, it is right-padded with blanks, or truncated, to fit the width. When a character field is retrieved, any padding blank characters are stripped away.
date	Date fields store dates as character strings with the format YYYYMMDD . Note that <fieldname>.full contains the time as well and is of the format YYYYMMDDHHMMSS .
smallint	Integer fields store whole numbers. This one is 2 bytes or 16 bits long.
integer	Integer fields store whole numbers. This one is 4 bytes or 32 bits long.
float	Real fields store decimal numbers. This one is 4 bytes or 32 bits long.
double	Real fields store decimal numbers. This one is 8 bytes or 64 bits long.
number (<width>,<decimals>)	Fields created with this option will be converted to smallint, integer or double depending on the value of width and decimal parameters. The width parameter is the total number of characters allocated to the field, including the decimal point. The decimals parameter controls the precision of the data and is the number of digits to the right of the decimal. If the decimal is zero and width is less than 5, then the field type will be changed to smallint . If the decimal is zero and width is greater than 5 and less than 10, then the field type will be changed to integer . For all other cases, the field type will be treated as double .

Range: YES | NO

Default: NO

IDs

Required/Optional: *Optional*

This optional specification is used to identify a specific set of tables to be read from the data source. If nothing is specified, all tables are returned. This feature is useful only if a data source contains many tables and for efficiency you want to read only one table, rather than all of them.

Range: Valid table name in the data source

Default: None

Workbench Parameter: *Feature Types to Read*

PASSWORD

For a password-protected GeoMedia Access warehouse, this is a required parameter; otherwise, it is optional. For a GeoMedia SQL Server warehouse, this is a required parameter and will specify the password required to log in to the database specified by **DATASET** keyword.

Workbench Parameter: *Password*

TEXT_SIZE_GROUND_UNITS

Specifies the text size in ground units.

Range: Any positive number less than 2,147,483,647

Default: 1

Workbench Parameter: *Text Size In Ground Units*

USE_ORIENTED_POINTS

Required/Optional: *Optional*

This directive specifies how the GeoMedia oriented points will be read. If **YES** is specified, then all GeoMedia oriented points will be returned with orientation information. If **NO** is specified, then all GeoMedia-oriented points will be returned as normal points without the orientation information. The default value of this keyword is **NO**.

Workbench Parameter: *Use oriented points*

RETRIEVE_ALL_SCHEMAS

Required/Optional: *Optional*

This specification is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

When set to 'Yes', indicates to the reader to return all the schemas of the tables in the source database.

If this specification is missing then it is assumed to be 'No'.

Range: YES | NO

Default: NO

RETRIEVE_ALL_TABLE_NAMES

Required/Optional: *Optional*

This specification is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

Similar to **RETRIEVE_ALL_SCHEMAS**: this optional specification is used to tell the reader to only retrieve the table names of all the tables in the source database. If **RETRIEVE_ALL_SCHEMAS** is also set to Yes, then **RETRIEVE_ALL_SCHEMAS** is chosen. If this value is not specified, then it is assumed to be No.

Range: YES | NO

Default: NO

Reader Directives – GeoMedia Access Warehouse

In addition to the reader directives that apply to all GeMedia warehouses, these directives are specific to GeoMedia Access Warehouses. For GeoMedia SQL Server Warehouse-specific directives, see Reader Directives – GeoMedia SQL Server Warehouse.

The directives listed below are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the GeoMedia Warehouse Reader is `FM0`.

Reader Directives – GeoMedia SQL Server Warehouse

In addition to the reader directives that apply to all GeMedia warehouses, these directives are specific to GeoMedia SQL Server Warehouses. For GeoMedia Access Warehouse-specific directives, see Reader Directives – GeoMedia Access Warehouse.

The directives listed below are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the SQL Warehouse Reader is `FM0_SQL`.

SQL_SERVER

Required/Optional: *Required*

Specifies the name of the server hosting the MS SQL Server that stores the GeoMedia warehouse.

Range: String

Workbench Parameter: *Server*

USER_NAME

Required/Optional: *Required*

This is required only for GeoMedia SQL Server Warehouse. The username for the database must be supplied here, either through the command-line interface or the user interface settings for translation.

Workbench Parameter: *User Name*

WHERECLAUSE

Required/Optional: *Optional*

This optional keyword specifies a WHERE clause which is applied to the columns of a table to limit the resulting features.

The main difference between this `WHERE` clause and the one specified on DEF lines is that this `WHERE` clause will be applied to all the source feature tables, and the one specified on the DEF lines applies to one particular table. In the case when both the WHERE clauses are specified, then the `DEF` line WHERE clause takes precedence.

Workbench Parameter: *Where Clause*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The GeoMedia Access Warehouse Writer writes features to a Microsoft Access database identified by the **DATASET** keyword. If the database does not exist, an empty database file is copied and used as a template.

The GeoMedia SQL Server Warehouse Writer writes to existing Microsoft SQL Server databases. If the database does not exist, the translation will be terminated. If the metadata tables and/or data tables do not exist in the database specified in the **DATASET** keyword, they will be created provided user has enough permissions to create and write to tables to the database. If the metadata tables exist, then information about newly created data table(s) will be appended to them. When writing to GeoMedia Access warehouses, if data tables exist and **OVERWRITE** is **NO**, then the data is appended to the table without trying to match the schema. When writing to GeoMedia SQL Server warehouses, if data tables exist and **fm0_truncate_table** is **NO**, then data is appended to the table without trying to match the schema. If **fm0_truncate_table** is **YES** and data tables exist, then all the existing data in the table will be deleted before writing.

As features are routed to either GeoMedia Warehouse Writer by the FME, they determine the layer (feature type) they are in and write the features to the corresponding table. Only one Microsoft Access file database is written during a single FME session, but many tables can be created within the database. Similarly, the SQL Server writer writes to one database, but may create many tables with that one database. The exception to this rule occurs when either of the writer types (MS Access or MS SQL Server) are used in conjunction with the FME Multi-writer and thus could write to several databases or files in a given FME session.

Additional Information About the MS SQL Server Writer

The GeoMedia SQL Server Warehouse Writer does not require GeoMedia to be installed. However, for successful writing, users must have sufficient permissions for creating and deleting tables, and inserting and updating rows in existing tables. All metadata and feature tables will be created with "dbo" ownership, which ensures that any user with permissions has access to the tables. A table with "dbo" ownership means that it is owned by the database administrator.

When writing to GeoMedia SQL Server warehouse, the writer will create four additional columns for storing the extents of geometry. These columns are required by GeoMedia. The names for these columns are derived from the name of the geometry column in the table being written. For example, if the name of the geometry column is **Geometry**, then the four derived column names will be **Geometry_xlo**, **Geometry_ylo**, **Geometry_xhi** and **Geometry_yhi**.

In situations where you are reading from a GeoMedia SQL Server warehouse table and writing to the same or different GeoMedia SQL warehouse, the names of extent columns read from the source warehouse may clash with the derived ones for writing if the geometry column name is same. Such a translation will fail with the error:

```
"SQL Server returned following error message(s):  
- [Microsoft][ODBC SQL Server Driver][SQL Server]Column names in each table must be  
unique.  
Column name 'Geometry_XLO' in table '<table-name>' is specified more than once.  
** AND ** [Microsoft][ODBC SQL Server Driver][SQL Server]Statement(s) could not be  
prepared."
```

There are two possible solutions to avoid such errors:

1. Remove the offending extent column names from the destination DEF lines. In workbench this could be accomplished by deleting the attributes names from the destination feature type.

or

2. Change the destination geometry column name. In Workbench, this can be easily done from the Parameters tab on the **Destination Feature Type Properties** dialog or in in mapping files by setting the **fm0_geome-try_column** attribute on the destination **DEF** line to the desired name.

DEF Line options

Creating Indexes

Data tables can be created with either primary or secondary indexes by the GeoMedia Warehouse Writer by appending an indexing suffix on the DEF line of the value of the field to be indexed. To create a primary index on a field, add the suffix

```
,primary
```

to the value. To create a secondary index, add

```
,indexed
```

Note: FME will accept primaryindex as a suffix for backwards compatibility, but primary is the recommended suffix.

Note that there can only be one primary index per table. If one is not specified, the default **PRIMARYINDEX** column will be created and indexed as the primary index. Alternatively, users can also provide a different name for **PRIMARYINDEX** column using the **default_primary_index_column** (or **default_primary_index_column** in workbench) option on the **DEF** line. If a column is specified as "primary" indexed then it will ignore the **fm0_primary_index_column** option for that table. If no column is specified "primary", and the **fm0_primary_index_column** has the same value as one of the attribute columns, that particular attribute column will be erased and replaced by a primary index.

Note: GeoMedia SQL Server writer allows to create index using multiple columns. This can be done by adding the appropriate suffix next to the column definition.

The following example creates a primary index on the field **ID** and a secondary index on the field **Number** for the table **mytable**.

```
FM0_DEF mytable \  
  FM0_GEOMETRY fm0_point \  
    ID integer,primary \  
    Name char(255) \  
    Number float,indexed
```

Example using the **fm0_primary_index_column** option

```
FM0_DEF mytable \  
  FM0_GEOMETRY fm0_point \  
  fm0_primary_index_column MyIndex \  
    ID integer,primary \  
    Name char(255) \  
    Number float,indexed
```

Specifying Primary Geometry Column Name

Data tables can be created with a user-specified Geometry column name by supplying its name on the **DEF** lines.

```
FM0_DEF mytable \  
  FM0_GEOMETRY fm0_point \  
    fm0_geometry_column MyGeometryColumn \  
    ID integer \  
    Name char(255) \  
    Number float
```

Specifying Whether to Truncate Tables

This option applies only to SQL Server writing. When writing features to data tables, users can specify whether to append to the existing table or delete all the features from the existing table before writing, using the 'fm0_truncate_table' option. If this option is **YES** then all the features(rows) in the existing data table will be deleted and new features written.

```

FMO_DEF mytable \
    FMO_GEOMETRY fm0_point \
    fm0_truncate_table YES \
    ID integer \
    Name char(255) \
    Number float

```

Creating Clustered or Non-clustered Indexes

This option applies only to SQL Server writing. When writing features to data tables, users can specify whether to create clustered index or not. In SQL Server database only one column per table can have clustered index. Using the `fm0_clustered_index` option which can have one of three values `{PRIMARY, EXTENTS, NONE}`, users can specify if the clustered index will be on the primary key column, extent columns or none of the columns. If `PRIMARY` is specified then the clustered index on the primary key column will be created. If `EXTENTS` is specified then a clustered index on the four extent columns (`<geom>_xlo`, `<geom>_ylo`, `<geom>_xhi` and `<geom>_yhi`) will be created and the primary key column will have non-clustered index. If `NONE` is specified, then all indexes are created as non-clustered.

```

FMO_DEF mytable \
    FMO_GEOMETRY fm0_point \
    fm0_clustered_index PRIMARY \
    ID integer \
    Name char(255) \
    Number float

```

Writer Directives – all GeoMedia Warehouses

The following directives apply to both GeoMedia Access Warehouses and GeoMedia SQL Server Warehouses. The directives listed below are prefixed by the current `<WriterKeyword>` in a mapping file. By default, the `<WriterKeyword>` for GeoMedia Access Warehouse is `FMO` and the `<WriterKeyword>` for GeoMedia SQL Server Warehouse is `FMO_SQL`.

The remaining writer-specific directives are discussed in `Writer Directives – GeoMedia Access Warehouse` and in `Writer Directives – GeoMedia SQL Server Warehouse`.

DATASET

The `DATASET` directive operates in the same manner as it does for the reader.

DEF

Required/Optional: *Required*

Each GeoMedia table must be defined before it is written to. The definition specifies the name of the table, the type of geometry on each row, and the names and types of all attributes.

The syntax of a GeoMedia SQL Server DEF line is:

```

<writerKeyword>_DEF <tableName> \
    fm0_type fm0_point|fm0_arc|fm0_line|fm0_area|
    fm0_text|fm0_collection|fm0_none \
    [fm0_geometry_column <column name>] \
    [fm0_primary_index_column <column name>] \
    [fm0_drop_table (yes|no)] \
    [fm0_truncate_table (yes|no)] \
    [<attrName> <attrType>]+

```

The table definition allows control of the table that will be created. If the fields and types are listed, the types must match those in the database. Fields which can contain NULL values do not need to be listed - these fields will be filled with NULL values.

If the table does not exist, then the field names and types are used to first create the table. In any case, if a `<fieldType>` is given, it may be any field type supported by the target database.

The configuration parameters present on the definition line are described in the following table:

Parameter	Contents
tableName	The name of the table to be written. If a table with the specified name exists, it will be overwritten if the fm0_drop_table DEF line parameter is set to YES , or it will be truncated if the fm0_truncate_table DEF line parameter is set to YES . Otherwise the table will be appended. Valid values for table names include any character string devoid of SQL-offensive characters and less than 30 characters in length.
fm0_geometry_column	This specifies the name of the column used to store the geometry. Default: Geometry
fm0_primary_index_column	This specifies the name of the column used to store the default primary index. Default: PRIMARYINDEX
fm0_drop_table	This specifies that if the table exists by this name, it should be dropped and replaced with a table specified by this definition. Default: NO
fm0_truncate_table	This specifies that if the table exists by this name, it should be cleared prior to writing. Default: NO
fieldName	The name of the field to be written. Valid values for field name include any character string devoid of SQL-offensive characters and less than 30 characters in length.
fieldType	See the Attribute Types section below.

Attribute Types

The following table shows the attribute types that are supported.

Field Type	Description
char(<length>)	Character fields store fixed-length strings. The length parameter controls the maximum characters that can be stored by the field. When a character field is written, it is right-padded with blanks, or truncated, to fit the width. When a character field is retrieved, any padding blank characters are stripped away.

Field Type	Description
date	Date fields store dates as character strings with the format YYYYMMDD. Note that <fieldname>.full contains the time as well and is of the format YYYYMMDDHHMMSS.
smallint	Integer fields store whole numbers. This one is 2 bytes or 16 bits long.
integer	Integer fields store whole numbers. This one is 4 bytes or 32 bits long.
float	Real fields store decimal numbers. This one is 4 bytes or 32 bits long.
double	Real fields store decimal numbers. This one is 8 bytes or 64 bits long.
number (<width>,<decimals>)	Fields created with this option will be converted to smallint, integer or double depending on the value of width and decimal parameters. The width parameter is the total number of characters allocated to the field, including the decimal point. The decimals parameter controls the precision of the data and is the number of digits to the right of the decimal. If the decimal is zero and width is less than 5, then the field type will be changed to smallint. If the decimal is zero and width is greater than 5 and less than 10, then the field type will be changed to integer. For all other cases, the field type will be treated as double.

WAREHOUSE_VERSION

The value of this keyword implies compatibility with warehouses created by GeoMedia – the value does not correspond to GeoMedia versions.

- **GeoMedia Access Warehouse:**

Required/Optional: *Optional*

Access warehouses created with GeoMedia version 4 are different from Access warehouses created with GeoMedia version 5 because of the changes to metadata tables. FME supports creating Access warehouses which are compatible with warehouses created by all the GeoMedia versions from 4 to 6. If **OVERWRITE_DATAFILE** is **NO**, then the writer will determine the version of the existing warehouse and write to it regardless of the setting of **WAREHOUSE_VERSION**. If the warehouse does not exist, the writer will create a new warehouse whose version will be determined by **WAREHOUSE_VERSION** keyword setting.

Range: 4, 5 or 6

Default: 5

By default, GeoMedia Access warehouse version 5 is created for a new warehouse or when overwriting an existing warehouse.

Workbench Parameter: <WorkbenchParameter>

- **GeoMedia SQL Server Warehouse:**

Required/Optional: *Optional*

With GeoMedia version 5.2, there have been changes to the metadata tables for SQL Server warehouses. FME can create new warehouses or write to existing SQL Server warehouses. When writing to an SQL Server warehouse, there is no option to overwrite, so if a warehouse already exists, then the value of this keyword will be ignored and data will be written to correspond to the warehouse version that exists. This keyword only applies when creating new SQL Server warehouses.

Range: 5, 5.2 or 6

Default: 5

Example:

```
FMO_WAREHOUSE_VERSION 5
```

or

```
FMO_SQL_WAREHOUSE_VERSION 5.2
```

Workbench Parameter: <WorkbenchParameter>

PLAIN_TEXT

Required/Optional: *Optional*

Note: This directive applies only to writing features of FME type `fme_text`.

By default, the GeoMedia Access Warehouse Writer will format text objects as rich text format (RTF) to insert the text size as part of the object, since text size cannot be set any other way during translation. In some cases, however, plain text may be desired instead of the default RTF text formatting.

Range: YES or NO

Default: YES

Example:

```
FMO_SQL_PLAIN_TEXT YES
FMO_PLAIN_TEXT NO
```

Workbench Parameter: <WorkbenchParameter>

FONT_SIZE

Required/Optional: *Optional*

This directive allows you to specify the font size in points.

Range: 1 - 1024

Default: 10

Example:

```
FMO_FONT_SIZE 12
```

Workbench Parameter: <WorkbenchParameter>

Writer Directives – GeoMedia Access Warehouse

These directives apply only to GeoMedia Access Warehouses. For other GeoMedia directives, see [Writer Directives – GeoMedia SQL Server Warehouse](#) and [Writer Directives – all GeoMedia Warehouses](#).

The directives listed below are prefixed by the current `<WriterKeyword>` in a mapping file. By default, the `<WriterKeyword>` for the GeoMedia Warehouse Writer is `FMO`.

OVERWRITE_DATAFILE

Required/Optional: *Optional*

When writing to a Microsoft Access database, the default action performed depends on the existence of the file or ODBC data source defined in `FMO_DATASET`. If the file does not exist, a template warehouse is used as a base and the tables and features to be written are appended to it. If the file exists, the tables and features are appended to it. This is the default behavior and is equivalent to setting the `OVERWRITE_DATAFILE` to `NO`. Setting this directive to `YES` means the template warehouse is used regardless of whether or not the data file exists, and anything existing previously in the warehouse named by `FMO_DATASET` is lost.

Range: YES | NO

Default: NO

Workbench Parameter: `<WorkbenchParameter>`

CREATE_SPATIAL_INDEX

When creating a new warehouse or overwriting an existing warehouse, native GeoMedia spatial index creation is performed automatically by the FME GeoMedia Access Warehouse Writer as long as the input data comes from a known coordinate system that has a corresponding datum mapping in the `autodt.ini` file under the GeoMedia install directory. If a spatial index is not desired for the new warehouse, its creation can be turned off by setting this keyword to `NO`.

When appending to an existing warehouse, creation of spatial index depends on whether the `SpatialKeyFieldName` property is set for the geometry column and the spatial key column actually exists. If the property is set and the column exists, then a spatial key will be automatically created for the geometry, regardless of the `CREATE_SPATIAL_INDEX` keyword setting.

Note: When using GeoMedia 5.0 (05.00.23.04) the GeoMedia Hotfix 05.0023.50 must be applied for spatial indexes to be properly created. This applies to both GeoMedia and GeoMedia Professional. The issue is corrected in GeoMedia 5.1. Recall that a valid source coordinate system is still required in all cases.

Example:

```
FMO_CREATE_SPATIAL_INDEX YES
```

Workbench Parameter: `<WorkbenchParameter>`

MODIFICATION_LOG

When writing to a Microsoft Access database, any changes to feature tables or metadata tables are written to `ModificationLog` and `ModifiedTables` metadata tables if modification logging is enabled. By default, modification logging is disabled. When writing large number of features, modification logging can add to the size of database.

Example:

```
FMO_MODIFICATION_LOG NO
```

Workbench Parameter: `<WorkbenchParameter>`

MDB_VERSION

This keyword allows you to specify the Microsoft Access file version.

Example:

```
FMO_MDB_VERSION 2002
```

Workbench Parameter: `<WorkbenchParameter>`

Writer Directives – GeoMedia SQL Server Warehouse

These specific directives apply only to GeoMedia SQL Server Warehouses. For other GeoMedia directives, see [Writer Directives – GeoMedia Access Warehouse](#) and [Writer Directives – all GeoMedia Warehouses](#).

The directives listed below are prefixed by the current `<WriterKeyword>` in a mapping file. By default, the `<WriterKeyword>` for the GeoMedia SQL Server Warehouse is `FM0_SQL`.

SQL_SERVER

Required/Optional: *Required*

Specifies the name of the server hosting the MS SQL Server that stores the GeoMedia warehouse.

Range: String

Workbench Parameter: *Server*

USER_NAME

Required/Optional: *Required*

This is required only for GeoMedia SQL Server Warehouse. The username for the database must be supplied here, either through the command-line interface or the user interface settings for translation.

Workbench Parameter: *User Name*

PASSWORD

This is required only for GeoMedia SQL Server Warehouse. It will specify the password required to log in to the database specified by `DATASET` keyword.

Workbench Parameter: *Password*

START_TRANSACTION

This statement tells the writer when to start actually writing features into the database. The writer does not write any features until the feature is reached that belongs to `<last successful transaction> + 1`. Specifying a value of zero causes every feature to be output. Normally, the value specified is zero – a non-zero value is only specified when a data load operation is being resumed after failing partway through.

Parameter: `<last successful transaction>`

The transaction number of the last successful transaction. When loading data for the first time, set this value to 0.

Example:

```
FM0_SQL_START_TRANSACTION 0
```

Workbench Parameter: *Start transaction at*

TRANSACTION_INTERVAL

This statement informs the FME about the number of features to be placed in each transaction before a transaction is committed to the database. If the `TRANSACTION_INTERVAL` statement is not specified, then a value of 1000 is used as the transaction interval.

Example:

```
FM0_SQL_TRANSACTION_INTERVAL 1500
```

Workbench Parameter: *Transaction interval*

IMMEDIATE_WRITE

This statement instructs the FME to immediately write each row of data to the database, rather than batching up writes into bulk arrays. Bulk arrays are normally preferred, as they require fewer queries sent to the database in order to store the data.

Example:`FM0_SQL_IMMEDIATE_WRITE NO`Workbench Parameter: *Immediate write***Feature Representation**

In addition to the generic FME feature attributes that FME Workbench adds to all features (see [About Feature Attributes](#)), this format adds the format-specific attributes described in this section.

GeoMedia features consist of geometric shapes which have their associated attributes as part of their definition. All GeoMedia features contain an `fm0_type` attribute, which identifies the geometric type. The feature contains additional attributes specific to its geometric type. Additional attributes are described in subsequent sections. The common attributes of all GeoMedia features are shown below. The table name is used as the feature type.

Attribute Name	Contents
<code>fm0_type</code>	<p>The GeoMedia geometric type of this entity.</p> <p>Range:</p> <p><code>fm0_point </code> <code>fm0_line </code> <code>fm0_arc </code> <code>fm0_area </code> <code>fm0_text </code> <code>fm0_collection </code> <code>fm0_none</code></p> <p>Default: No default</p>

The blob breakdowns for the composite (aggregate), boundary (donut) and collection (aggregate) features are not mentioned below with the simple features, each of which has its corresponding attributes and blob data parsed.

Note: Geometries from FME do not map exactly to GeoMedia geometries. Usually, however, this is not an issue because although some of the internals of the geometries may be changed, their appearance in GeoMedia is still the same.

Points**fm0_type:** `fm0_point`

GeoMedia point features specify a single set of coordinates, which is converted, and are devoid of any additional geometric attributes.

fm0_type: `fm0_oriented_point`

GeoMedia-oriented point features specify a single set of coordinates along with a rotation vector. This rotation attribute is stored as a rotation angle by the reader. Either a rotation angle or the individual components of a rotation vector may be passed to the writer.

Attribute Name	Contents
<code>fm0_rotation</code>	<p>This attribute specifies an optional rotation for the shape where the clockwise direction is positive.</p> <p>Range: -360 to 360 degrees</p> <p>Default: 0</p>

Attribute Name	Contents
fm0_orientation_i fm0_orientation_j fm0_orientation_k	These attributes specify the components of the optional 3D rotation vector for a point. They may be used as a preferential alternative to the fm0_rotation attribute. The vector components are written directly to GeoMedia and typically form a unit length vector.

Lines

fm0_type: *fm0_line*

GeoMedia line features consist of a list of two or more points. No additional attributes are required to control GeoMedia lines.

Arcs

fm0_type: *fm0_arc*

GeoMedia arcs consist of a start and end point as well as a normal vector and radius. From these points and their values, the center point of an arc, as well as the start and sweep angles, can be calculated and used to render the arc in FME format. Note that the normal vector serves to identify an arc as being drawn either clockwise or counter-clockwise. Similarly, a positive radius indicates an arc of greater than 180 degrees, while a negative radius indicates an arc of less than 180 degrees.

Attribute Name	Contents
fm0_primary_axis <i>Applicable only with classic geometry.</i>	The length of the semi-major axis in ground units (x-axis). Range: Any real number > 0 Default: No default
fm0_secondary_axis <i>Applicable only with classic geometry.</i>	The length of the semi-minor axis in ground units (y-axis). Range: Any real number > 0 Default: No default
fm0_start_angle <i>Applicable only with classic geometry.</i>	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of start_angle. Default: 0
fm0_sweep_angle <i>Applicable only with classic geometry.</i>	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of sweep_angle. Range: 0.0..360.0 Default: No default
fm0_rotation <i>Applicable only with classic geometry.</i>	This attribute specifies and optional rotation for the shape where the clockwise direction is positive. Range: -360 to 360 degrees Default: 0

Areas

fm0_type: fm0_area

GeoMedia area features specify various polygon features. An area is simply a closed line and can be an individual area, a donut, or an aggregate of areas. As with lines, there are no additional attributes for area features.

Text

fm0_type: fm0_text

GeoMedia text is simple and straightforward, offering no font, color or style attributes. Text attributes include the string to be written, a rotation and a possible size.

Note: Although the reader attempts to provide a reasonable value for the text size, it may be necessary to adjust the text size during a Workbench or mapping file translation. Similarly, when writing, the result may have to be adjusted through the GeoMedia product for better results.

Attribute Name	Contents
fm0_justification	The alignment of the text around the origin (not present in Reader) Range: 0..2, 4..6, 8..10 0 centered vertically, centered horizontally 1 centered vertically, left of the origin 2 centered vertically, right of the origin 4 above the origin, centered horizontally 5 above the origin, left of the origin 6 above the origin, right of the origin 8 below the origin, centered horizontally 9 below the origin, left of the origin 10 below the origin, right of the origin Default: 9
fm0_text_size	The size of the text in ground units (not present in Writer) Range: Any real number ≥ 0 Default: None
fm0_text_font_size	The size of the text in points (not present in Reader). If this is not specified in the writer, the value from the FONT_SIZE writer keyword is used. Range: Any real number between 0 and 1024. Default: None
fm0_text_string	The text string to be displayed. Range: Any character string Default: None
fm0_rtf_text_string	This attribute is used in both the reader and writer. If this attribute is set when reading, the original

Attribute Name	Contents
	<p>RTF text string from the source dataset will be included. If this attribute is set when writing, the RTF text string will be written. In this case, the keyword PLAIN_TEXT will be overridden by this attribute. This option is provided for two main reasons: When translating from GeoMedia to GeoMedia if the source dataset contains RTF text strings, then they will be translated without any loss of formatting information; and it also gives the user the ability to supply custom formatted RTF text strings per feature, written out to the destination dataset.</p> <p>Range: Any RTF character string.(No syntax verification done by the writer)</p> <p>Default: None</p>
<p>fm0_rotation</p> <p>Deprecated – applicable only with classic geometry.</p>	<p>This attribute specifies an optional rotation for the shape where the clockwise direction is positive.</p> <p>Range: -360 to 360 degrees</p> <p>Default: 0</p>

None

fm0_type: fm0_none

Features with no coordinates are tagged with this type when reading or writing to or from GeoMedia.

```
<ReaderKeyword>_DEF <tableName> \
[SQL_STATEMENT <sqlStatement>] \
  FMO_GEOMETRY fm0_point|fm0_arc|fm0_line|fm0_area|
  fm0_text|fm0_none
[<attrName> <attrType>]+
```

Collections

fm0_type: fm0_collection

These are GeoMedia features that are heterogeneous aggregates of simple types.

For example, an aggregate of points, lines, and polygons can comprise a collection. There are no additional attributes required for collections.

Troubleshooting

Common issues that arise when using the GeoMedia Access Warehouse Reader and Writer are sometimes a matter of knowledge about how the product works, as well as its limitations.

Spatial Indexes

To create spatial indexes in GeoMedia Professional 5, you will need to install Hot Fix 05.00.23.50, available on Intergraph's website at:

www.intergraph.com/gis/support/GMProHotFix5.asp

Text Size

This can be an awkward issue in writing. Often a dataset will appear with very small text that you cannot see until you zoom in closely, and other times the text seems too large for the data. It is suggested that for reading, use the **TEXT_SIZE_GROUND_UNITS** keyword to set the size appropriate to the bounds of your source dataset. For writing, set the **PLAIN_TEXT** to **NO** and provide a suitable font size using **FONT_SIZE** keyword.

“Class not found” Errors

This is a message from GeoMedia that is passed back through FME. There are two basic causes:

The GeoMedia Access Warehouse Writer requires GeoMedia to be installed in order to run; if it is not installed, then this message may result.

If GeoMedia is installed, then there is likely more than one copy or version of GeoMedia installed on the machine and the installations are in conflict. This is possible due to the registration scheme of the GeoMedia products. The solution recommended by Intergraph is to uninstall all GeoMedia products, clean the registry of anything related to GeoMedia (especially the **HKEY_LOCAL_MACHINE\Software\Intergraph\Applications** key), and then reinstall the single version of GeoMedia you want to use.

Translation Errors

Translation errors can occur if the destination dataset is set to create a file in a directory that does not exist. Since the GeoMedia Warehouse Writer is a file-based writer, it requires that the path in which the destination file is to be created must already exist.

Translation Errors in Workbench

There is a known issue with the GeoMedia SQL Server warehouse writer and Workbench Feature Type Properties.

The Database User field should be left blank. Entering a username in the field may cause the FME translation to fail. However, even if the translation is successful, GeoMedia will not be able to read the resulting table.

Intergraph MGE Reader/Writer

Format Notes:

This format is not supported by FME Base Edition.

The **Intergraph MGE Reader/Writer** is nearly identical to the (Bentley) **MicroStation GeoGraphics Reader/Writer**. The only difference is that by default, the <ReaderKeyword> for the GeoGraphics reader is *GG*.

This chapter contains information that is applicable to both formats.

The Intergraph Modular GIS Environment (MGE) Reader and Writer modules provide the FME with the ability to read and write design files and their associated databases.

Overview

MGE uses standard MicroStation *elements* to represent the graphical portions (geometry) of the geographic map objects. These elements are identical to the design file elements used by the Design File Reader/Writer. Any geometry enhancements made to the Design File Reader/Writer benefit the MGE Reader/Writer.

The non-graphical aspects of a map object are defined by linking an MGE *feature* to elements in a database. Features are linked to elements in the database via pairs of entity and **mslink** numbers. Feature definitions are stored in a special table within the project's database. Each row of the feature table defines a feature, providing a name, feature code, feature type, display attributes (to be applied to elements linked to that feature), and possibly a pointer to a table of non-graphical attributes to be associated with each instance of the feature.

Tip: A feature in Intergraph MGE is closer in nature to a feature type in FME terminology. In this chapter, we refer to FME's features using the term "FME features" to differentiate it from MGE's features.

The following table describes the general database tables used. Other tables not listed are non-graphical user attribute tables. The **mscatalog** and feature tables are mandatory for the database used.

Table Name	Description
mscatalog	The key table to determine where there are more attributes to the feature in the other tables.
feature	This table holds all the feature types and the general feature information.
maps	The table contains the corresponding map information associated with the feature.
category	This table holds information about the categories associated with the feature.
ugfeature, ugcaterory, ugjoin_cat, ugmap, ugcommand, ugtable_cat	Extended tables of the Intergraph Design format table that hold additional information for MGE.

The MGE Reader and Writer use symbolic names for the design element types rather than numeric values. This greatly simplifies element type specification. The following table maps the design element type number to its corresponding FME feature *igds_type* value that is used by the MGE Reader and Writer. Subsequent sections describe the handling of each of these element types in detail.

IGDS Element Type	FME igds_type
2	igds_cell
3	igds_point
4, 12	igds_line
6, 14	igds_shape
7	igds_text_node
11, 12	igds_curve
15	igds_ellipse
16	igds_arc
17	igds_text
7, 17	igds_multi_text
2, 6, 14	igds_solid

MGE Quick Facts

Format Type Identifier	MGE
Reader/Writer	Both
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	GeoGraphics Feature Name
Typical File Extensions	.dgn, .cad
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	Yes
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Enhanced Geometry	Yes
Geometry Type	igds_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	yes		polygon	yes
circular arc	yes		raster	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
donut polygon	yes		solid	no
elliptical arc	yes		surface	no
ellipses	yes		text	yes
line	yes		z values	yes
none	no			

Reader Overview

The MGE Reader extracts all elements from the design files, one at a time, via the help of the Design File Reader. For each element, it finds every attached feature and passes each on to the rest of the FME for processing, along with any corresponding non-graphical attributes. Complex elements are extracted as single FME features. If the element has any attribute linkages attached to it, these are read in and added as attributes to the FME feature created.

When the MGE Reader encounters an element type it does not recognize, it is processed as an “unlinked” element type.

Reader Directives

The following directives are processed by the MGE reader. The suffixes shown will be prefixed by the current **<ReaderKeyword>** in the mapping file. By default, the **<ReaderKeyword>** for the MGE reader is MGE. Please see the *Bentley MicroStation Design Reader/Writer* for other keyword suffixes.

DATASET

Required/Optional: *Required*

Contains the directory name of the input MGE files.

SERVER_TYPE

Required/Optional: *Required*

Contains the server type for the input data.

SERVER_NAME

Required/Optional: *Required*

Contains the server name for the input data.

USER_NAME

Required/Optional: *Optional*

Contains the user name for the database. May be required by the database being used.

Workbench Parameter: *Database Username*

PASSWORD

Required/Optional: *Optional*

Contains the password for the database. May be required by the database being used.

Workbench Parameter: *Database Password*

DATABASE_NAME

Required/Optional: *Optional*

Contains the database name. May be required by the database being used.

DEF

Required/Optional: *Required*

The `<ReaderKeyword>_DEF` must be used to define a feature before elements of that feature may be read. It allows complete specification of a MGE feature. The definition of a feature consists of:

- A name for the feature.
- A set of database attributes associated with each element attached to the feature. All database attributes of a given feature are stored in a single table; elements attached to that feature contain linkages into this attribute table.
- A set of properties which apply to all map objects belonging to that feature. (These properties, described in greater detail below, define the graphic specifications applied to elements attached to the feature, as well as specifying the feature's feature code, category, and other feature-specific attributes.)

The syntax for the definition line is:

```
<ReaderKeyword>_DEF "<featureName>" \  
  [<attrName> <attrType>]* \  
  [<featPropertyName> "<featPropertyValue>"]*
```

The feature name must follow the conventions for MGE feature names. The letter case of alphabetic characters in the names is insignificant, as all names are converted to lowercase for internal use by the MGE Reader.

There may be zero or more attributes associated with a feature. Like feature names, the case of attribute names is insignificant. (Attribute names are conventionally specified in lowercase to more easily distinguish them from the type properties.) The following table shows the supported attribute types:

Field Type	Description
char(<width>)	Character fields store fixed length strings. The width parameter controls the maximum of characters that can be stored by the field. No padding is required for strings which are shorter than this width.
date	Date fields store dates as character strings with the format YYYYMMDD.
number(<width>,<decimals>)	Number fields store single and double precision floating point values. The width parameter is the total number of characters allocated to the field, including the decimal point. The decimals parameter controls the precision of the data, and is the number of digits to the right of the decimal.
logical	Logical fields store TRUE/FALSE data. Data read or written from/to such fields must always have a value of either true or false.

The MGE reader only requires the specification of those attributes which are of interest to the translation process; there is no need to mention any attributes which are not used. Further, every attribute which is specified in the mapping file is verified against those associated with the feature being defined, to ensure that it exists and that its type is compatible.

There are a number of properties that may be specified to define the feature of interest. Every specified property must agree with the existing MGE feature, or an error will result. No property specifications are required when

reading from a MGE project; however, in order for feature attributes not belonging to the design file to be written, a GG_ATTR_TABLE needs to be defined (the default table name is the feature name).

UNITS

Required/Optional: *Optional*

The <ReaderKeyword>_UNITS directive controls the conversion between UORs in the design file and FME coordinates. There are three possibilities, outlined in the table below. If no UNITS directive is specified, then GG_MASTER_UNITS is the default.

GG_UNITS Value	Description
GG_MASTER_UNITS	The UORs read from the design file will be converted into master units , according to the conversion factor defined in MicroStation's terminal control block (TCB), before being stored in an FME feature. This is the default.
GG_SUB_UNITS	The UORs read from the design file will be converted into subunits , according to the conversion factor defined in MicroStation's TCB, before being stored in an FME feature.
GG_UORS	The UORs read from the design file will be stored directly in an FME feature, with no conversion.

FEATURE_TABLE_NAME

The <ReaderKeyword>_FEATURE_TABLE_NAME directive identifies the name of the feature table. Use this directive only if you have a feature table with the name other than **feature**.

Workbench Parameter: *Feature table name*

CATEGORY_TABLE_NAME

The <ReaderKeyword>_CATEGORY_TABLE_NAME directive identifies the name of the feature table. Use this directive only if you have a category table with the name other than **category**.

Workbench Parameter: *Category table name*

Writer Overview

The MGE Writer writes all elements to the current master file. It extracts the conversion parameters required to translate coordinates from feature coordinate units to internal Units of Resolution (UORs). It also determines the dimension of the master file.

When writing to GG/MGE, one of the Data Source types from ODBC, MDB or ORACLE has to be selected. Against the Database Server Name, an empty database has to be specified which the writer uses to write all the tables like mscatalog, feature, etc. For instance, with Access (MDB) as the data source, an empty database **.mdb** or **.accdb** file would suffice. When an existing database is used, new records are appended to the existing tables. The username and password are optional and may or may not apply in every case.

A cell library file may optionally be used by the MGE Writer. Cell libraries contain named symbol definitions which can be used to depict point features. If a cell library is specified, the MGE Writer reads in all the cell definitions for later when cell features are output. The MGE Writer can use either 2 or 3 dimension cell libraries, and will automatically convert the cell definitions to be of the correct dimension for output.

The MGE Writer then outputs each FME feature it is given. Most often, a single FME feature corresponds to a single design element. However, some of the IGDS element types cause several elements to be output as a complex unit

(with the complex bit turned on). This occurs when a multi-line text object, a cell, or a closed shape or linear feature with more than 101 coordinates is output. The MGE Writer hides all of the details of complex element output.

Tip: Since coordinates in design files are ultimately stored as integer UORs, it is possible for precision to be lost or overflow to occur when they are output. Care must be taken to ensure that the conversion parameters in the seed file preserve the data precision and range.

WAREHOUSE_VERSION

Required/Optional: *Optional*

Range: 4, 5 or 6

Default: 5

SERVER_TYPE

Required/Optional: *Required*

Contains the server type for the output data.

SERVER_NAME

Required/Optional: *Required*

Contains the server name for the output data.

USER_NAME

Required/Optional: *Optional*

Contains the user name for the database. May be required by the database being used.

Workbench Parameter: *Database Username*

PASSWORD

Required/Optional: *Optional*

Contains the password for the database. May be required by the database being used.

Workbench Parameter: *Database Password*

DATABASE_NAME

Required/Optional: *Optional*

Contains the database name. May be required by the database being used.

Workbench Parameter: *Output Access Database File*

UNITS

Required/Optional: *Optional*

Specifies how FME feature coordinates will be interpreted and converted into UORs. See the documentation under the MGE Reader for details.

Workbench Parameter: *Output Units*

IMMEDIATE_WRITE

Required/Optional: *Optional*

Specifies if the database is written immediately when needed (yes) or not (no).

Workbench Parameter: *Immediately Write Database Records*

TRANSACTION_INTERVAL

Required/Optional: *Optional*

The number of features that are to be in a single transaction before the FME performs a commit operation when writing to the database.

Workbench Parameter: *Transaction Interval*

DEF

Required/Optional: *Required*

Defines an MGE feature. Each feature must be defined before it can be written. The definition specifies the characteristics which make up the MGE feature. Additionally, it specifies the non-graphical attributes which will appear in the correlation section for the feature. There may be many **DEF** lines, one for each file to be read.

The following table summarizes the supported feature properties:

- Entries under the **Workbench Parameter Name** column are the keyword descriptions as seen in Workbench.
- Entries under the **Mapping File Property Name** column correspond to the actual DEF line parameters used in the mapping file.

Note: Regardless of type, all feature properties' values are specified in quotation marks.

Workbench Property Name	Mapping File Property Name	Description	Type	Required/Optional
Feature Code	GG_FEAT_CODE	A set of dot-separated integers which define numerically the feature hierarchy structure.	char(10)	Optional
Category Number	GG_CATEGORY	The name of the category containing the feature; this must be one of the categories defined on the MGE project.	char(32)	Optional
User Attribute Table Name	GG_ATTR_TABLE	The name of the database table defining the non-graphical attributes for the feature. If there are no such features, this value should be the null string (" "). This is required in order to write out the attributes that are not part of the design file. The default is the feature name.	char(32)	Required
Element Type	GG_ELEMENT_TYPE	The type of elements tagged with this feature.	integer	Optional
Element Lock	GG_ELOCK	The strength of enforcement of the above element type (0 =>	char(12)	Optional

Workbench Property Name	Mapping File Property Name	Description	Type	Required/Optional
		interest - attach to any element type; 1 => similar - may attach only to "comparable" element type; 2 => exact - element type must exactly match GG_ELEMENT_TYPE).		
Geometry Type	GG_FEAT_TYPE	The geometry type of the feature type. (-=> undefined - default; 1 => point; 2 => line; 3 => area boundary; 4 => area centroid; 5 => label).	integer	Optional
Feature Level	GG_LEVEL	The level number applied to elements attached to this feature.	integer	Optional
Feature Style or Line Code	GG_STYLE	The style or line code applied to elements attached to this feature.	integer	Optional
Feature Weight	GG_WEIGHT	The weight applied to elements attached to this feature.	integer	Optional
Feature Color	GG_COLOR	The color applied to elements attached to this feature.	integer	Optional
Feature Angle	GG_ANGLE	Angle at which feature is set.	float	Optional
Feature Height	GG_HEIGHT	Height of feature, applied to attached text and node elements.	float	Optional
Feature Width	GG_WIDTH	Width of feature, applied to attached text and node elements.	float	Optional
Line Spacing	GG_LINE_SPACING	Line spacing of text nodes attached to feature.	float	Optional

Workbench Property Name	Mapping File Property Name	Description	Type	Required/Optional
Line Length	GG_LINE_LENGTH	Line length of text nodes attached to feature.	integer	Optional
Feature Font	GG_FONT	Font used for attached text and node elements.	integer	Optional
Feature Symbol	GG_SYMBOL	Feature symbol used to form symbol text.	char(1)	Optional
Feature Justification	GG_JUSTIFICATION	Direction at which feature text is justified.	integer	Optional
Active Stream Delta	GG_STREAM_DELTA	Active stream delta.	float	Optional
Active Stream Tolerance	GG_STREAM_TOL	Active stream tolerance.	float	Optional
Feature Snap Type	GG_SNAP_TYPE	Type of feature snap.	integer	Optional
Feature Snap Tolerance	GG_SNAP_TOL	Feature snap tolerance.	integer	Optional
Database Linkage Mode	GG_NEW_DUP	Database linkage mode (-1 => NO_LINK; 0=> NEW_LINK; 1=>DUP_LINK).	integer	Optional
Feature Class	GG_CLASS	Feature class.	integer	Optional
Feature Priority	GG_FEAT_PRIO	Priority of feature relative to other features in design file.	integer	Optional
Database Links to Infomode	GG_INFO_MODE	Sets database links to infomode.	integer	Optional
Displayable Attribute Type	GG_DAS_TYPE	Displayable attribute type.	integer	Optional
Display Priority	GG_DISPLAY_PRIO	Priority of display for elements with multiple feature tags.	float	Optional

FEATURE_TABLE_NAME

Required/Optional: *Optional*

Specifies the name of the feature table to be written. This defaults to the name **feature**.

Workbench Parameter: *Feature table name*

CATEGORY_TABLE_NAME

Required/Optional: *Optional*

Specifies the name of the category table to be written. This defaults to the name **category**.

Workbench Parameter: *Category table name*

LINKAGE_TYPE

Required/Optional: *Optional*

Specifies the type of database linkages that will be attached to features written to MGE layers. The value is a character string. If this directive is not specified, the MGE writer defaults to creating database linkages of type "dmrs".

Workbench Parameter: *Linkage Type*

MANGLE_DBCS_TEXT

Required/Optional: *Optional*

Controls whether or not double-byte-character set text is mangled when text strings are written. Microstation uses special header bytes to single DBCS text. If this directive is set to **YES** in the mapping file, then these special bytes will be output when a DBCS text string is encountered. The default value is **NO**. Note that the IGDS reader automatically de-mangles DBCS text.

Workbench Parameter: *Mangle DBCS Text*

COMPRESS_AT_END

Required/Optional: *Optional*

Tells the writer to compact the **.mdb**, **.accdb** Access database file. This compresses the file size after all the writing is done. This makes use of the existing MDB database option to compact. The compact operation compresses the output database to a small size on disk.

Range: YES | NO

Default: NO

Workbench Parameter: *Compress Database When Done*

SPLIT_BIG_DGN7_FILES

Required/Optional: *Optional*

Note: This directive applies to the V7 writer only.

Allows user to split Version 7 DGN files bigger than 32 MB. Note that this directive can be manually set to **no** in the mapping file.

Range: YES | NO

Default: YES

Workbench Parameter: *Split Files > 32 MB*

MDB_VERSION

This statement instructs FME to set the version of the output Microsoft Access file version. Access file versions 97 and 2000 are the supported types. By default, an Access 2000 file is created.

Example:

```
MDB_VERSION 97
```

Workbench Parameter: *MS Access Version*

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Each design element may be attached to one or more MGE features. The FME feature consists of a design element, a single MGE feature, and the attributes from the row of the feature's attribute table which corresponds to the element. Special FME feature attributes are used to hold design element parameters. The MGE Writer will use these attribute values as it fills in an element structure during output. The MGE Reader will set these attributes in the FME feature it creates for each element it reads.

Tip: By using a common value for graphic group value, several otherwise separate elements may be tied together into a logical super-element for later processing by application programs.

The FME considers the MGE feature name to be the *FME feature type* of an element in a MGE design file. Each MGE element, regardless of its geometry type, shares a number of other parameters, as described in the following table. Please see the Design File Reader/Writer feature representation for the parameters specific descriptions to each of the supported element types.

ISO 8211 Reader

Format Notes: This format is not supported by FME Base Edition.

The ISO8211 Reader provides FME with access to data in an ISO/IEC 8211:1994 formatted files. These files are called ISO8211 files.

Overview

ISO 8211 is a format for the structured and self-described transfer of data. It is the underlying encoding format used for the SDTS and S-57 file formats, as well as being used for some other purposes. While data in an ISO 8211 formatted file may be spatial, that can't be directly deduced from the ISO 8211 formatting information. This reader produces features with attributes, but no geometry.

More information on the ISO 8211 format can be found at:

<http://user.icx.net/~brooks/iso8211.html>

ISO 8211 Quick Facts

Format Type Identifier	ISO8211
Reader/Writer	Reader
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	DDF
Typical File Extensions	.ddf
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Not applicable
Transaction Support	No
Geometry Type	iso8211_type
Encoding Support	No

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	point	no
circles	no	polygon	no
circular arc	no	raster	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
donut polygon	no		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	no		z values	n/a
none	yes			

Reader Overview

The FME considers a single ISO 8211 formatted file, usually with the **.DDF** extension, to be a data set. Each record in the file is read as a feature.

Reader Directives

The suffixes listed are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the ISO 8211 reader is **ISO8211**.

DATASET

Required/Optional: *Required*

The file name of the ISO 8211 formatted file to be read often has the extension **.DDF** as shown in this example:

```
ISO8211_DATASET PALO_ALTO\SC01LE01.DDF
```

Workbench Parameter: *Source ISO8211 File(s)*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Feature Representation

Features read from the database consist of a series of attribute values. They have no geometry. The feature type of each feature is **DDF**.

ISO 8211 records, which are translated into FME features, consist of a list of fields. Fields consist of subfields, which can repeat within a field. Fields and subfields have names.

Each subfield value is translated into an FME attribute, and its name is generated by appending the subfield name to the field name separated with an underscore. Repeated subfields are translated into FME array syntax.

Example:

An ISO8211 record with three fields – **ENID**, **LINE** and **SADR** – might be translated as follows. In this case, the **ENID** field has two subfields, **MODN** and **RCID**.

The **LINE** field has three subfields **MODN**, **OBRP** and **RCID**. The **SADR** field has two subfields **X** and **Y**, but they are repeating so they are presented in array syntax.

```
String Attribute: ENID_MODN' is N001'  
String Attribute: ENID_RCID' is 10'  
String Attribute: LINE_MODN' is LE01'  
String Attribute: LINE_OBRP' is LE'  
String Attribute: LINE_RCID' is 10'  
String Attribute: SADR_X{0}' is 57367669'  
String Attribute: SADR_X{1}' is 57367659'  
String Attribute: SADR_Y{0}' is 414608954'  
String Attribute: SADR_Y{1}' is 414610051'
```

Each record in an ISO 8211 file can potentially have different sets of fields selected from a set of fields defined in the header of the file. When a field appears, it will always have the same set of subfields.

JSON (JavaScript Object Notation) Reader/Writer

Format Notes:

This format is not supported by FME Base Edition.

JSON (JavaScript Object Notation) is a simple structured text format. It is designed to be easy for both humans and computers to produce and consume, and to be easily integrated into JavaScript applications.

Overview

JSON is centred around the concept of an *Object* and an *Array*. An object is a set of name/value pairs, while an array is a list of values. A value can be an object, an array, a string, a number (integral, decimal, or exponential), a boolean, or the literal value *null*. An object key must always be a string.

A JSON object is a pair of braces containing key/value pairs separated by commas, with a colon between each key and value. The keys are an unordered set, which means that each of the keys in an object should be unique, and the order that the keys appear in the object is ignored. An object can have any number of key/value pairs, including zero. A sample JSON object is:

```
{
  "key with string value":"this is a string value",
  "key with exponential value":-59.45E-4,
  "key with null value":null,
  "key with boolean value":true,
  "key with array value":[ false, 12, 56.82, { "key":"value" } ],
  "key with object value":{}
}
```

A JSON array is an ordered set of values separated by commas. An array can have any number of values, including zero. A sample JSON array is:

```
[
  12,
  "a string value",
  56.3e6,
  null,
  false,
  [ 1, 2, 3, {} ]
]
```

Coordinate systems are supported in JSON through the use of a *json_ogc_wkt_crs* key and OGC WKT text. Quotations in the WKT text should be escaped with backslashes.

```
{
  "json_featuretype":"SampleJSONFeature",
  "json_ogc_wkt_crs":"GEOCS[\"WGS84 Lat/Long's, Degrees, -180 ==> \
+180\",DATUM[\"WSG_1984\",SPHEROID[\"world Geodetic System of \
1984\",6378137,298.257223563],AUTHORITY[\"EPSG\",\"6326\"]], \
PRIMEM[\"Greenwich\",0],UNIT[\"degree\",0.0174532925199433], \
AUTHORITY[\"EPSG\",\"4326\"]]"
  "json_geometry":"LINESTRING(45.3 56.89, 85.63 96.73, 12.61 91.38)"
}
```

Further information on JSON can be found at <http://www.json.org>.

JSON Quick Facts

Format Type Identifier	JSON
Reader/Writer	Reader/Writer
Licensing Level	Professional
Dependencies	None
Dataset Type	File/URL
Feature Type	Varies – schema is dependent on the source dataset.
Typical File Extensions	.json
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	No
Transaction Support	No
Geometry Type	json_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	yes		z values	no
none	yes			

Reader Overview

The JSON reader is capable of reading JSON text structured in several different ways. The only requirement it places on the JSON text is that every FME feature must be represented by a single JSON object or array.

If the feature is represented by a JSON object, the user can specify the object key whose value will become the FME feature type, as well as the key whose value contains the feature geometry. All other key/value pairs in the JSON object are turned into attribute names/values. If the value of a key is a nested JSON object or array, then the actual JSON text becomes the value of the corresponding attribute.

If the feature is represented by an array, each value in the array will become an attribute value. The attribute names will be generic names attribute0, attribute1, attribute2, etc, corresponding to the position of the value within the array. As in the object case, if an array element is a nested JSON object or array, the corresponding attribute value will be the actual JSON text. A feature constructed in this manner will have no geometry or coordinate system, and will have the default JSON feature type.

The JSON objects and arrays that are to be turned into FME features are specified by a JSON query. More information on JSON queries can be found in the JSONQueryFactory documentation.

The reader can read from a local or network file, or a remote URL accessible via http or ftp. The reader can access these URLs directly, or via a proxy server.

Geometry

The JSON reader supports two different geometry formats: GeoJSON and OGC-WKT, and the user specifies the type of geometry that the reader should attempt to use. The reader also allows for attribute-only datasets which contain no geometry. More information on the GeoJSON geometry objects can be found in the GeoJSON reader/writer documentation.

Example 1 (GeoJSON) :

```
{
  "json_featuretype": "SampleJSONFeature",
  "json_geometry":
  {
    "type": "LineString",
    "coordinates": [ [45.3,56.89], [85.63,96.73], [12.61,91.38] ]
  }
}
```

Example 2 (OGC-WKT):

```
{
  "json_featuretype": "SampleJSONFeature",
  "json_geometry": "LINESTRING(45.3 56.89, 85.63 96.73, 12.61 91.38)"
}
```

Coordinate Systems

The JSON reader currently supports coordinate systems in OGC WKT format as described in the overview.

FME Feature Attributes

Feature attributes are set to the value of a JSON object key, or to the value of an element in a JSON array. If this value is a string, it will be set as a UTF-16 encoded string attribute on the FME feature. If the value is an integer, real number or boolean, it will be set as the corresponding attribute type.

If the value is a nested object or array, then the actual JSON text of the object or array will become the attribute value. This allows particular portions of the object or array to be retrieved using the JSONExtractor transformer (which uses the JSONQueryFactory FME factory).

Reader Directives

The suffixes shown are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the JSON reader is `JSON`.

DATASET

Required/Optional: Required

The location of the JSON file to be read. This can be the path to a local or network file, or a URL.

Examples:

```
JSON_DATASET c:\json_sample.json
JSON_DATASET \\path\to\network\file.json
```

JSON_DATASET [http://search-
h.yahooapis.com/ImageSearchService/V1/imageSearch?appid=YahooDemo&query=FME&output=json](http://search.yahooapis.com/ImageSearchService/V1/imageSearch?appid=YahooDemo&query=FME&output=json)

SCHEMA_SCAN_QUERY

Required/Optional: Optional

Specifies where in the JSON text the reader should look for objects representing FME features. If no value is provided, then the default query `json[*]` is used. This default query will work for a dataset that is an array of JSON objects, each of which represents a single feature. More information on JSON queries can be found in the `JSONQueryFactory` documentation.

Example:

```
JSON_SCHEMA_SCAN_QUERY json["searchResults"][*]
```

Workbench Parameter: *JSON Query for feature objects*

FEATURE_TYPE_KEY

Required/Optional: Optional

This directive specifies the JSON object key whose value will become the feature type of the FME feature produced from the object. If no value is provided, then a default value of `json_featuretype` will be used.

Example:

```
JSON_FEATURE_TYPE_KEY json_featuretype
```

Workbench Parameter: *Feature Type key name*

GEOMETRY_KEY

Required/Optional: Optional

This directive specifies the JSON object key whose value contains the geometry of the FME feature produced from the object. If no value is provided, then a default value of `json_geometry` is used.

Example:

```
JSON_GEOMETRY_KEY json_geometry
```

Workbench Parameter: *Geometry key name*

GEOMETRY_FORMAT

Required/Optional: Optional

This directive specifies the geometry format that the reader should use when converting the value of the `GEOMETRY_KEY` directive into FME geometry. Possible values are `GeoJSON`, `OGC-WKT` and `None`. If no value is provided, then a default value of `GeoJSON` is used.

Example:

```
JSON_GEOMETRY_FORMAT OGC-WKT
```

Workbench Parameter: *Geometry format*

DELETE_DOWNLOAD_FILE

Required/Optional: Optional

If the value of this directive is 'Yes' then when the reader has finished reading downloaded JSON text, it will delete the file that the JSON text was downloaded to. The default value is 'No'. The value of this directive is only meaningful if the dataset is a URL.

Example:

```
JSON_DELETE_DOWNLOAD_FILE No
```

Workbench Parameter: *Delete downloaded file*

PROXY_URL

Required/Optional: Optional

Specifies a proxy server that the reader will be use when accessing a URL dataset. The port number of the proxy server can be set in the URL, or by using the **PROXY_PORT** directive.

Example:

```
JSON_PROXY_URL www.someproxy.net  
JSON_PROXY_URL www.someproxy.net:8080
```

Workbench Parameter: *Http Proxy URL*

PROXY_PORT

Required/Optional: Optional

Specifies the port number of the proxy server indicated by the **PROXY_URL** directive. This directive should only be used if the port number was not indicated in the **PROXY_URL** directive. This directive is ignored if the **PROXY_URL** directive has no value.

Example:

```
JSON_PROXY_PORT 8080
```

Workbench Parameter: *Http Proxy Port*

PROXY_USERNAME

Required/Optional: Optional

Specifies the username to use when accessing a password protected proxy server. This directive is ignored if any of the **PROXY_URL**, **PROXY_PASSWORD** or **PROXY_AUTH_METHOD** directives have no value.

Example:

```
JSON_PROXY_USERNAME someusername
```

Workbench Parameter: *Http Proxy Username*

PROXY_PASSWORD

Required/Optional: Optional

Specifies the password to use when accessing a password protected proxy server. This directive is ignored if any of the **PROXY_URL**, **PROXY_USERNAME** or **PROXY_AUTH_METHOD** directives have no value.

Example:

```
JSON_PROXY_PASSWORD password1234
```

Workbench Parameter: *Http Proxy Password*

PROXY_AUTH_METHOD

Required/Optional: Optional

Specifies the authentication method to use when accessing a password protected proxy server. This directive is ignored if any of the **PROXY_URL**, **PROXY_USERNAME** or **PROXY_PASSWORD** directives have no value. Acceptable values for this directive are 'Basic' or 'Digest'.

Example:

```
JSON_PROXY_AUTH_METHOD Basic
```

Workbench Parameter: *Http Proxy Authentication Method*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The JSON writer will output features as an array of JSON objects. The user can specify the key in the JSON object whose value will contain the FME feature type, as well as the key whose value will contain the feature geometry. Feature attributes are written out as key/value pairs in the JSON object.

Coordinate Systems

The JSON writer currently supports coordinate systems in OGC WKT format as described in the overview. If no coordinate system is specified, no coordinate system will be output.

Geometry

The JSON writer supports the same two different geometry formats as the JSON reader: GeoJSON and OGC-WKT, and the user specifies the type of geometry that the writer should use. The writer also allows the user to specify that all feature geometry should be ignored, so that only feature attributes will be written.

Writer Directives

The suffixes shown are prefixed by the current `<WriterKeyword>` in a mapping file. By default, the `<WriterKeyword>` for the JSON writer is `JSON`.

DATASET

Required/Optional: Required

The file to which the should output the JSON text. If the file does not exist it will be created.

Example:

```
JSON_DATASET c:\data.json
```

Workbench Parameter: *Destination JSON File*

FEATURE_TYPE_KEY

Required/Optional: Optional

This directive specifies the JSON object key whose value will contain the feature type of the FME feature represented by a JSON object. If no value is provided, then a default value of `json_featuretype` will be used.

Example:

```
JSON_FEATURE_TYPE_KEY json_featuretype
```

Workbench Parameter: *Feature Type key name*

GEOMETRY_KEY

Required/Optional: Optional

This directive specifies the JSON object key whose value will contain the geometry of the FME feature represented by a JSON object. If no value is provided, then a default value of `json_geometry` is used.

Example:

```
JSON_GEOMETRY_KEY json_geometry
```

Workbench Parameter: *Geometry key name*

GEOMETRY_FORMAT

Required/Optional: Optional

This directive specifies the geometry format that the writer should use to populate the value of the object key specified by the `GEOMETRY_KEY` directive. Possible values are `GeoJSON`, `OGC-WKT` and `None`. If no value is provided, then a default value of `GeoJSON` is used.

Example:

```
JSON_GEOMETRY_FORMAT OGC-WKT
```

Workbench Parameter: *Geometry format*

WRITE_NULL_ATTRIBUTE_VALUES

Required/Optional: Optional

This directive specifies whether or not the JSON object representing an FME feature should include a key for every attribute specified by the feature schema, or only those attributes for which the FME feature has a value. Possible values for this directive are `Yes` and `No`. If the value is `No`, then the JSON object will only contain keys for which the FME feature has an attribute value. If the value of the directive is `Yes`, then the output JSON objects will contain keys for every attribute in the feature type schema, and keys for which an FME feature has no attribute value will have a null JSON value. The default value of this directive is `No`.

Example:

```
JSON_WRITE_NULL_ATTRIBUTE_VALUES Yes
```

Workbench Parameter: *Write 'null' for attributes with no value*

WRITER_CHARSET

Required/Optional: Optional

The character set encoding in which the JSON text will be written. Possible values for this directive are `UTF-8`, `UTF-16`, `UTF-16BE`, `UTF16-LE`, `UTF-32`, `UTF-32BE` and `UTF-32LE`. If no character set is specified, the JSON text will be written in the `UTF-8` character set.

Example:

```
JSON_WRITER_CHARSET UTF-16
```

Workbench Parameter: *Output Character Set*

WRITE_BOM

Required/Optional: Optional

The value of this directive specifies whether or not the JSON writer should preface the JSON text with a byte order marker to indicate the endianness of the Unicode text. Possible values for this directive are `Yes` and `No`. The default value is `No`.

Example:

`JSON_WRITE_BOM` Yes

Workbench Parameter: *Byte Order Marker*

JSONP_FUNC_NAME

Required/Optional: Optional

The value of this directive specifies the JSONP javascript function name that the user wants to wrap the JSON file with. JSONP (JSON with Padding) is developed as a standard for grabbing JSON from external domains, that works well with AJAX calls.

The default value is null. If no value is set or the default is set, then the JSON writer will output a JSON file without the JSONP padding.

Example:

`JSONP_FUNC_NAME` getFeatures

Workbench Parameter: *JSONP function call name, if JSONP mode used*

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Geometry

The geometry of JSON features may be identified by the `json_type` attribute. The valid values for this attribute are:

<code>json_type</code>	Description
<code>json_no_geom</code>	FME Feature with no geometry.
<code>json_point</code>	Point feature.
<code>json_line</code>	Linear feature.
<code>json_polygon</code>	Simple polygon or donut feature.
<code>json_rectangle</code>	Simple rectangular polygon feature.
<code>json_collection</code>	Feature with multiple geometries.

No Geometry

json_type: `json_no_geom`

Features with their `json_type` attribute set to `json_no_geom` do not contain any geometry data.

Points

json_type: `json_point`

Features with their `json_type` set to `json_point` are single coordinate features or an aggregate of single points.

Lines

json_type: `json_line`

Features with their `json_type` set to `json_line` are polyline features or an aggregate of polylines.

Areas

json_type: `json_polygon`

Features with their `json_type` set to `json_polygon` are polygon features which may or may not have interior boundaries, or an aggregate of such polygons.

Boxes

json_type: `json_rectangle`

Features with their `json_type` set to `json_rectangle` are simple rectangular closed polygons. Features with this type will not have any interior boundaries.

Aggregates

json_type: `json_collection`

Features with their `json_type` set to `json_collection` are a heterogeneous collection of multiple geometries.

Landmark Z-Map Writer

Format Notes: This format is not supported by FME Base Edition.

Overview

The Landmark Z-Map module provides the FME with the ability to write geometric data files which are compatible with Landmark Graphics' Z-MAP Plus™ software.

Z-Map files store both feature geometry and attribution. A Z-Map file has the following file name extension:

File Name Extension	Contents
.dat	Z-Map data file

The extension is added to the base name of the Z-Map file. The base name is the feature name.

Z-Map Quick Facts

Format Type Identifier	ZMAP
Reader/Writer	Writer
Licensing Level	Professional
Dependencies	None
Dataset Type	Writer: Directory
Feature Type	File name base
Typical File Extensions	.dat
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Not applicable
Schema Required	Yes
Transaction Support	No
Encoding Support	No
Geometry Type	ZMAP_TYPE

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	point	yes
circles	no	polygon	yes
circular arc	no	raster	reader: yes writer: no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
donut polygon	no		solid	no
elliptical arc	no		surface	no
ellipses	no		text	yes
line	yes		z values	no
none	yes			

Band Interpretations	Real64
Palette Key Interpretations	not applicable
Palette Value Interpretations	not applicable
Nodata Value	None
Cell Origin (x, y)	0.5, 0.5
Rotation Support	No
GCP Support	No
World File Support	No
TAB File Support	No

Writer Overview

The Z-Map writer creates and writes feature data to Z-Map files into the directory specified by the **DATASET** keyword. Existing Z-Map files with the same name specified are overwritten with the new feature data.

The Z-Map writer does not currently support raster features.

Writer Directives

The directives processed by the Z-Map writer are listed below. The suffixes shown are prefixed by the current **<writerKeyword>** in a mapping file. By default, the **<writerKeyword>** for the Z-Map writer is **ZMAP**.

DATASET

Required/Optional: *Required*

This is the name of a directory containing one or more Z-Map files. The default extension for Z-Map files is **.dat**.

An example of the **DATASET** keyword in use is:

```
ZMAP_DATASET /usr/data/zmap/input
```

Workbench Parameter: *Destination Landmark Z-MAP Directory*

DEF

Required/Optional: *Required*

The **<writerKeyword>_DEF** must be used to define a feature before elements of that feature may be written. The syntax for the definition line is:

```
<writerKeyword>_DEF <featureName> \  
    [<attrName> <attrType>]*
```

The following table shows the supported attribute types:

Field Type	Description
<code>char(<width>)</code>	Character fields store fixed length strings. The width parameter controls the maximum of characters that can be stored by the field. No padding is required for strings which are shorter than this width.
<code>date</code>	Date fields store dates as character strings with the format YYYYMMDD.
<code>number(<width>)</code>	Number fields store single and double precision floating point values. The width parameter is the total number of characters allocated to the field, including the decimal point. These numbers are stored as characters.
<code>logical</code>	Logical fields store TRUE/FALSE data. Data read or written from/to such fields must always have a value of either true or false.
<code><code_type>, <width></code>	The <code>code_type</code> parameter is the Z-Map data type code number. The width parameter controls the field width.

The following is an example DEF line with the `<writerKeyword>` as ZMAP:

```
ZMAP_DEF roads \
    "well Name" char(31) \
    Operator 20,47
```

Feature Representation (Z-Map Writer)

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Each written Z-Map element, regardless of its geometry type, shares attributes in the following table. Subsequent subsections will describe parameters specific to each of the supported element types.

Attribute Name	Contents
<code>zmap_type</code>	The Z-Map geometric type of this entity. Range: <code>zmap_point</code> <code>zmap_line</code> <code>zmap_text</code> Default: No default
<code>LATITUDE</code>	The latitude of the first point. This only exists when the coordinate system is Latitude/Longitude. Range: Any real

Attribute Name	Contents
	Default: No default
LONGITUDE	The longitude of the first point. This only exists when the coordinate system is Latitude/Longitude. Range: any real Default: No default
X (EASTING)	The x-coordinate. This only exists when the coordinate system is not Latitude/Longitude. Range: any real Default: No default
Y (NORTHING)	The y-coordinate. This only exists when the coordinate system is not Latitude/Longitude. Range: Any real Default: No default

Points

zmap_type: zmap_point

There are no specific attributes for this type.

Lines

zmap_type: zmap_line

Attribute Name	Contents
SEG I.D.	The fault or line identifier (all vertices for the same line have the same identifier). The values are generated by the writer. Range: Any real number. Default: increments from 1

Text

zmap_type: zmap_text

Attribute Name	Contents
TEXT_ANGLE	The text rotation angle. Range: Any real number. Default: 0
TEXT_SIZE	The text character size in inches. Range: Any real number. Default: 0
CHARACTER_TEXT	The text label. Range: Maximum 47 characters Default: Blank

Landmark Zycor Graphics File (ZGF) Reader

Format Notes:

This format is not supported by FME Base Edition.

The Landmark Zycor Graphics File (ZGF) Reader allows the Feature Manipulation Engine (FME) to read Zycor Graphics Files (ZGFs). The ZGF format is used by Zycor programs to store 2D graphics information.

Overview

ZGF is a two-dimensional (2D) system with no provision for storing user-defined attributes for the geometric data. A ZGF file has the following file name extension:

File Name Extension	Contents
.zgf	Vector geometric data

The extension is added to the basename of the ZGF file.

A ZGF contains one or more Major Graphics Units (MGUs), also called pictures. A picture is typically a basemap or contour map, although nothing prevents other types of pictures from being stored. Each picture is composed of one or more Logical Graphics Blocks (LGBs), also called segments. LGBs contain things such the map border, a title block, a contour, a posted well, etc. LGBs contain one or more graphic elements. There are six kinds of graphic elements: poly-line, polygon, text, marker, clip window, and application data elements.

ZGF Quick Facts

Format Type Identifier	ZGF
Reader/Writer	Reader
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	Geometry based name
Typical File Extensions	.zgf
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Never
Schema Required	No
Transaction Support	No
Geometry Type	zgf_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	no		solid	no
elliptical arc	no		surface	no
ellipses	no		text	yes
line	yes		z values	no
none	yes			

Reader Overview

The ZGF reader extracts features from a file one at a time, and passes them on to the rest of the FME for further processing. The reader finishes when it reaches the end of the file.

Each feature returned by the ZGF reader has its FME feature type set to one of the following: `zgf_polyline`, `zgf_marker`, `zgf_text`, `zgf_clipwindow`, `zgf_polygon` or `zgf_applicationdata`.

Reader Directives

The directives processed by the ZGF reader are listed below. The suffixes shown are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the ZGF reader is `ZGF`.

DATASET

Required/Optional: *Required*

The value for this keyword is the file containing the ZGF dataset to be read. A typical mapping file fragment specifying an input ZGF file looks like:

```
ZGF_DATASET /usr/data/zgf/zgffile.zgf
```

Workbench Parameter: *Source Landmark Zycor ZGF File(s)*

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the `mitab.dll` in the FME home directory to `mapinfo.dll`.

The syntax of the `MAPINFO_SEARCH_ENVELOPE` directive is:

```
MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

```
MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

All ZGF features contain the `zgf_type` attribute. The value for this attribute is set identically to the value of the feature's feature type.

Attribute Name	Contents
<code>zgf_type</code>	Specifies what type of data the feature represents. Range: <code>zgf_polyline</code> <code>zgf_marker</code> <code>zgf_text</code> <code>zgf_clipwindow</code> <code>zgf_polygon</code>

Attribute Name	Contents
	zgf_applicationdata Default: No default

Furthermore, all ZGF features contain the following attributes, which are obtained from the MGU containing the feature:

Attribute Name	Contents
zgf_mgu_name	The name of the MGU containing the feature. Range: Maximum of 76 characters Default: Blank
zgf_mgu_type	The type of picture contained by the MGU. Range: Maximum of 4 characters Default: Blank
zgf_mgu_status	An integer denoting the status of the MGU. A non-zero value may indicate that the MGU should not be visible. Range: 32-bit integer Default: No default.

ZGF features also receive some attributes from the LGB containing the feature, as follows:

Attribute Name	Contents
zgf_lgb_type	An integer code describing what is contained by the LGB, such as the map border, a title block, a contour, a posted well, etc. Range: 32-bit integer Default: No default
zgf_lgb_ignore	An integer indicating whether the LGB should be ignored. Range: < 0 - ignore the LGB >= 0 - do not ignore the LGB Default: No default

Polyline

zgf_type: zgf_polyline

ZGF line features specify linear features defined by a sequence of x and y coordinates.

These attributes are specific to line features.

Attribute Name	Contents
zgf_line_type	An integer code describing the appearance of the line. Range: 1 - normal line

Attribute Name	Contents
	2 - dashed line 3 - bold line 4 - hachured line 5 - hachured line 6 - double dashed line 7 - triple dashed line Default: No default
zgf_smooth	The smoothing flag.
zgf_continue	This flag specifies whether or not the polyline was constructed from multiple smaller segments, as stored in the ZGF file. Range: 0 - stored as a single, standalone line 1 - stored as multiple line segments Default: 0
zgf_line_width	The width of the line. Range: 32-bit real Default: No default
zgf_first_dash_length zgf_second_dash_length zgf_third_dash_length	For dashed lines, these denote the length of the dashes. Range: 32-bit real Default: No default
zgf_first_dash_gap zgf_second_dash_gap zgf_third_dash_gap	For dashed lines, these denote the length of the gaps. Range: 32-bit real Default: No default
zgf_hachure_direction zgf_hachure_spacing zgf_hachure_length	These are used to specify the presentation properties of hachure lines.
zgf_marker_number	For normal or boldface lines, if zgf_marker_number is not 0, it specifies the symbol that should be drawn at each point on the line. Range: 32-bit integer Default: No default
zgf_marker_height	If zgf_marker_number is specified, zgf_marker_height denotes the height of this marker. Range: 32-bit real Default: No default
zgf_color_index	An index to the color table owned by the feature's

Attribute Name	Contents
	containing MGU. Used to find the feature's color. Range: 0 - 255 Default: No default
zgf_units	Indicates the natural coordinate space for the feature. Note that all coordinates are converted to user space by the ZGF reader. Range: 1 - user (also called application, engineering, or scaled) 2 - plotter (also called unscaled) 3 - normalized (all coordinates between 0.0 - 1.0, inclusive) Default: No default

Marker

zgf_type: zgf_marker

The zgf_marker features indicate a point at which a well symbol or symbols should be drawn.

The following table lists the attributes common to all zgf_marker features.

Attribute Name	Contents
zgf_marker_number	This is a number specifying what sort of symbol to draw at the point. Range: 32-bit integer Default: No default
zgf_marker_height	This specifies the height of this marker. Range: 32-bit real Default: No default
zgf_id	A marker record in a ZGF file may actually contain several points. These are split into separate point features by the ZGF Reader. Thus, this is a unique ID given to each marker record, so that points that were originally from the same marker record will have the same zgf_id value. Range: 32-bit integer Default: No default
zgf_color_index	See the Polyline section.
zgf_units	See the Polyline section.

Text

zgf_type: zgf_text

Text features describe a text string located at a particular position.

They have the following attributes:

Attribute Name	Contents
zgf_text_string	The text string. Range: Strings Default: No default
zgf_text_height	This specifies the height of the text. Range: 32-bit real Default: No default
zgf_angle	The text's angle of rotation in degrees. Range: 0.00 - 360.00 Default: No default.
zgf_font	The font. Range: 1 - Zycor original 2 - Roman Simplex 3 - Roman Complex 4 - Italics 5 - Duplex Default: No default.
zgf_justification	Justification of the text string. Range: 1 - lower left 2 - lower right 3 - center Default: No default.
zgf_color_index	See the Polyline section.
zgf_units	See the Polyline section.

Clip Window

zgf_type: zgf_clipwindow

Clip Window features describe a rectangular area to which other features should be clipped. Note that the ZGF Reader does not actually do any clipping, it simply reads these as rectangular features of type zgf_clipwindow.

The following table lists the attributes common to all zgf_clipwindow features.

Attribute Name	Contents
zgf_mode	Specifies whether or not the clip window is "on", i.e. that other features should be clipped to this window. Range: 0 - off (don't clip)

Attribute Name	Contents
	1 - on (clip) Default: No default
zgf_units	See the Polyline section.

Polygon

zgf_type: zgf_polygon

ZGF polygon features specify solid-filled area (polygonal) features.

The following table lists the attributes common to all zgf_polygon features.

Attribute Name	Contents
zgf_color_index	See the Polyline section.

Application Data

zgf_type: zgf_applicationdata

ZGF application data features contains non-graphical, application specific data. They do not have any geometry.

The following table lists the attributes common to all zgf_applicationdata features.

Attribute Name	Contents
zgf_id	Identification number. Range: 32-bit integer Default: No default.
zgf_data	The application data. Range: Strings Default: No default.

LandXML Reader

The LandXML Reader allows the Feature Manipulation Engine (FME) to read LandXML (Extensible Markup Language) documents.

LandXML is an XML-based survey exchange format. This document assumes that the reader has a good understanding of the LandXML format.

More information is available at <http://www.landxml.org>.

Overview

Current support for LandXML is limited to the following LandXML element types.

- <Alignment>
- <Alignments>
- <Application>
- <CgPoint>
- <CgPoints>
- <CoordinateSystem>
- <FeatureDictionary>
- <LandXML>
- <Parcel>
- <Parcels>
- <PlanFeature>
- <PlanFeatures>
- <Project>
- <Surface>
- <Surfaces>
- <Survey>
- <Units>

This provides support for a restricted subset of LandXML version 1.1. Further support is planned for the future.

Each of the above elements in a LandXML document will result in one or more features being produced.

LandXML Quick Facts

Format Type Identifier	LANDXML
Reader/Writer	Reader
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	.xml
Typical File Extensions	.xml
Automated Translation Support	limited
User-Defined Attributes	Yes
Coordinate System Support	limited
Generic Color Support	No
Spatial Index	Never
Schema Required	No
Transaction Support	No
Geometry Type	xml_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	no		polygon	yes
circular arc	limited		raster	no
donut polygon	yes		solid	no
elliptical arc	limited		surface	yes
ellipses	no		text	no
line	yes		z values	yes
none	no			

Reader Overview

The LandXML reader module produces an FME feature for each supported element type in the file.

Automatic coordinate system support is limited to CoordinateSystem elements which contain OGCWKT coordinate system definitions or recognized EPSG codes. In the case where neither is true, the coordinate system attributes will be available within FME/Workbench for further examination, but geometry found in the LandXML file will not have a coordinate system set. It is of course possible to set the coordinate system based on the values found in the CoordinateSystem feature manually.

CoordGeom Elements

The geometry of an Alignment, Parcel, or PlanFeature makes use of a CoordGeom element to store a part of its geometry. A CoordGeom element consists of any number of linear elements or curve/spiral elements. The LandXML Reader currently has limited support for Curve and Spiral elements. Curve elements which are arc types, defined by three

points (start, center, end) will be read. Chord Curve elements are not read, and no Spiral elements are read. If a Curve or spiral type is not read it will result in an interpolation between segments of a CoordGeom element. A warning will be issued if a spiral element or an unsupported Curve element is encountered. Support for these elements will be added in future versions of the reader. It is recommended that if the data contains a spiral or curve element as part of a CoordGeom element that the geometry of the output be considered faulty (since when a spiral or curve is skipped, the intervening space will be linearly interpolated from the surrounding segments. But if linear interpolation produced correct geometry, then there would have been no need for a curve or spiral component).

Reader Directives

The directives processed by the LandXML reader are listed below. The suffixes shown are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the LandXML reader is `LANDXML`.

DATASET

Required/Optional: *Required*

An example of the `DATASET` directive in use is:

```
LANDXML_DATASET /usr/data/input.xml
```

Workbench Parameter: *LandXML Document*

SPLIT_COLLECTIONS

Required/Optional: *Required*

Default: YES

Many formats are not capable of handling features with heterogeneous aggregates (e.g. an aggregate composed of a point and a polygon). In order to more easily support automatic translations, by default these collections will be deaggregated. In a non-automated translation you will usually want this set to 'NO'. This keyword can only be set at schema-generation time, as it alters the schema features that would be read.

An example of the `SPLIT_COLLECTIONS` keyword in use is:

```
LANDXML_SPLIT_COLLECTIONS NO
```

Workbench Parameter: *Split Collections*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

*** Workbench Parameter**

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

*** Workbench Parameter**

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

*** Workbench Parameter**

Additional Attributes to Expose

Feature Representation

LandXML is an XML-based format, so all features have both XML specific format attributes, as well as LandXML-specific format attributes.

This table lists the LandXML-specific format attributes.

Attribute Name	Contents
landxml_landxml_type	The LandXML element type that this feature belongs to.
landxml_retwall_offset_above	On a Surface Feature, determines how to interpret the vertical offset on the geometry.
landxml_retwall_offset_right	On a Surface Feature, determines how to interpret the horizontal offset on the geometry.
landxml_element_id	An attribute which identifies this feature. It is used to track the hierarchy of elements within a LandXML document.
landxml_parent_id	An attribute which identifies the parent of this feature. It is used to track the hierarchy of elements within a LandXML document.
landxml_children_id	A comma-separated value attribute which identifies the children of this feature (by landxml_element_id value). It is used to track the hierarchy of elements within a LandXML document.
landxml_equipment_type	This attribute identifies the type of the equipment used for a survey. Only found on Survey feature types. Current possible values are "equipment_laser_details", "equipment_instrument_details", "equipment_gps_reciever_details", "equipment_gps_antenna_details".

In particular, landxml_element_id, landxml_parent_id, and landxml_children_id all work together to provide information on the hierarchy of elements within the original LandXML document. These attributes are *bookkeeping* attributes, in that they store information on the *placement* of elements within the original LandXML document, but are not properties of the elements themselves. That is, there is no element or attribute that corresponds to the landxml_parent_id attribute, but rather the placement of the element itself is recorded in these format attributes.

As an example of these attributes in action, consider the following LandXML fragment

```
<Parcels name="AirPorts">
  <Parcel name="AirPort_01">
    <Parcels name="Runways_01"/>
  </Parcel>
  <Parcel name="AirPort_02">
    <Parcels name="Runways_02">
      <Parcel name="Runway_02_01"/>
    </Parcels>
  </Parcel>
</Parcels>
```

Here we see a Parcels feature, which contains two Parcel features, each of which in turn contains a single Parcels feature, and one of those embedded Parcels contains a single Parcel feature. The following features would be produced (with geometry and irrelevant attributes omitted for clarity).

```

Parcels
  name : Airports
  landxml_element_id : 1
  landxml_children : 2,4

Parcel
  name : AirPort_01
  landxml_element_id : 2
  landxml_parent_id : 1
  landxml_children : 3

Parcels
  name : Runways_01
  landxml_element_id : 3
  landxml_parent_id : 2

Parcel
  name : AirPort_02
  landxml_element_id : 4
  landxml_parent_id : 1
  landxml_children : 5

Parcels
  name : Runways_02
  landxml_element_id : 5
  landxml_parent_id : 3
  landxml_children : 6

Parcel
  name : Runway_02_01
  landxml_element_id : 6
  landxml_parent_id : 5

```

So within FME, these IDs can be used to reconstruct the original structure of the document. Within a reader, the `landxml_element_id` will be unique to each feature.

Note that while it is possible to create cyclic containment hierarchies (i.e. with a `Parcels` feature referencing a `Parcel` feature as its parent, which in turn references that `Parcels` feature as *its* parent). These will cause the translation to fail with a warning that a cyclic relationship was detected. Self loops (in which a feature refers to itself as its parent) will be detected and the parent reference changed to the default parent in the document.

Of the feature types currently read, only `Parcel/Parcels` and `CgPoints` elements can create a cycle. A `Parcels` element is a container for `Parcel` elements, and a `Parcel` element can contain a `Parcels` element. In the case of `CgPoints`, a `CgPoints` element can contain other `CgPoints` elements.

Geometry Representations

In general, the geometry will be identified by the `xml_type` attribute as defined in the documentation for the XML (Extensible Markup Language) Reader/Writer.

Click here for more information on See "Geometry Traits" and See "LandXML Features ".

Geometry Traits

In a number of LandXML feature types (`Alignment`, `PlanFeature`, `Parcel`, `Surface` elements), the geometry of a feature can be quite complicated and make use of a number of different elements, each with its own purpose. Since FME currently restricts a feature to having only one geometric element, the LandXML reader constructs an aggregate out of the different geometry properties that a given feature has. Then, in order to make it easier to examine a given geometry property, we have assigned to each geometry a geometry trait (essentially an attribute on a geometry) which identifies it.

Trait Name	Contents
<code>landxml_geometry_type</code>	A string identifying the geometry in question.

An example taken from an Alignment element shows the `landxml_geometry_type` trait in question.

```
+++++
Geometry Type: IFMEAggregate
Number of Geometry Traits: 1
GeometryTrait(string): 'landxml_geometry_type' has value alignment_geometry'
Number of Geometries:
-----
Geometry Number: 0
  Geometry Type: IFMEPoint
  Number of Geometry Traits: 1
  GeometryTrait(string): 'landxml_geometry_type' has value start'
  Coordinate Dimension: 2
  (1283.49421251,1309.71022416)
-----
Geometry Number: 1
  Geometry Type: IFMEPath
  Number of Geometry Traits: 1
  GeometryTrait(string): 'landxml_geometry_type' has value alignment_boundary'
  Number of Segments: 13
  [rest of geometry omitted]
+++++
```

Geometry Traits are also used on features produced by the LandXML reader in order to tie properties of a geometry tightly to that geometry. For example, it is possible to have a set of BreakLine elements on a Surface SourceData element. Each BreakLine element can have a set of properties (called 'Feature' elements in LandXML). By coupling the name-value pairs directly on the geometry, it is easy to determine which feature-property refers to which BreakLine.

The LandXML Writer does not currently support writing Curve or Spiral types. These geometries will be stroked into lines before writing.

LandXML Features

In addition to a large set of predefined attributes, LandXML has a mechanism for user-defined attributes called *Features* (not to be confused with FME features).

While LandXML Features include a set of related attributes (a name, a DocFileReference, etc.), they can also include simply name-value pairs. In order to capture the related attributes as a group, list attributes are used. In particular, the structure of a LandXML Feature encompasses the following attributes:

- `Feature{}.Code`
- `Feature{}.Source`
- `Feature{}.Property{}.Label`
- `Feature{}.Property{}.Value`
- `Feature{}.DocFileRef{}.Name`
- `Feature{}.DocFileRef{}.Location`
- `Feature{}.DocFileRef{}.Filetype`
- `Feature{}.DocFileRef{}.FileFormat`

Due to LandXML's structure, these Feature attributes may be stored on either a Feature, or as traits on a geometry.

MapInfo MIF/MID Reader/Writer

The MapInfo Data Interchange Format (MIF) Reader/Writer allows FME to read and write MapInfo® import and export files.

MIF is a published ASCII format used by the MapInfo product for input and export. The *MapInfo Reference Manual* describes the MIF format and all constants it uses for color, style, symbol, and fill patterns.

MapInfo Interchange Format Files are often called MIF or MIF/MID files.

Overview

MapInfo is a two-dimensional (2D) system with no provision for transferring elevation data for each vertex in a MapInfo feature. However, point features can define an elevation attribute to store their elevation.

MIF files store both feature geometry and attribution. A logical MIF file consists of two physical files, having the following file name extensions:

File Name Extension	Contents
.mif	Vector geometric data
.mid	Attributes for the geometric data

These extensions are added to the basename of the MIF file.

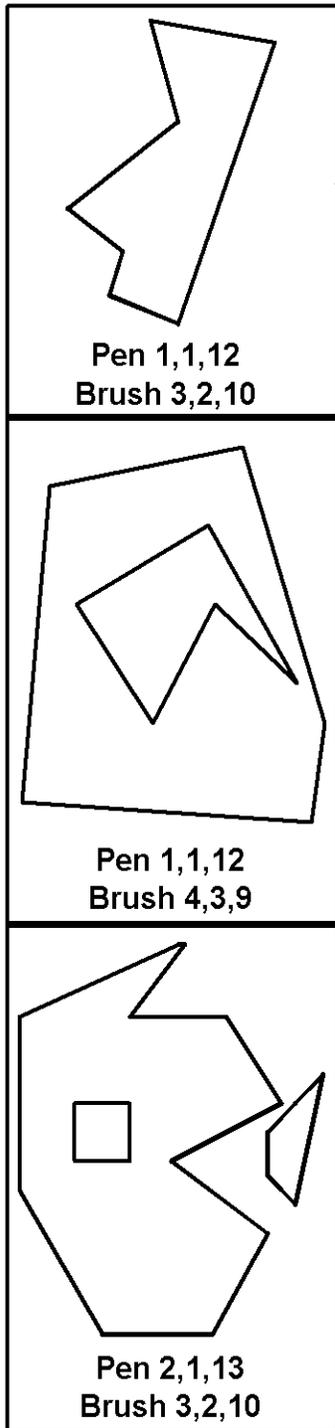
The MapInfo reader and writer support the storage of point, line, polyline, arc, ellipse, rectangle, rounded rectangle, region (polygon), and text geometric data in .mif files. The MIF format also stores features with no geometry. Features having no geometry are referred to as having a geometry of *none*.

Each geometric entity present in a .mif file has display properties such as pen and brush width, pattern, and color. In addition, each entity has a row of attributes stored in an associated .mid file. A single .mif file contains many different types of geometry however, the associated attribute in the .mid file must have the same number and type of fields for each entity in the .mif file. The order of the entries in the two files is synchronized. For example, the second geometric entity in the .mif file has the attributes held in the second row of the .mid file.

The number and type of attributes associated with each entity is specified by the user. There must be at least one attribute field in the .mid file.

The following example shows a MIF file containing three *region* entities in it. Note that the second polygon contains a hole, and the third polygon is an aggregate of two disjoint polygons, one of which contains a hole. Each geometric entity in turn corresponds with one record in the attribute table.

.mif Geometry File



.mid Attribute File

DEPTH	LAKE_NAME
10	Smokey Lake
15	Beaver Hill Lake
25	Swan Lake

The diagram shows a table with two columns: DEPTH and LAKE_NAME. Three arrows originate from the .mif Geometry File and point to the rows in this table. The top arrow points to the first row (DEPTH 10, LAKE_NAME Smokey Lake), the middle arrow points to the second row (DEPTH 15, LAKE_NAME Beaver Hill Lake), and the bottom arrow points to the third row (DEPTH 25, LAKE_NAME Swan Lake).

FME considers a MIF dataset to be a collection of MIF files in a single directory. The attribute definitions for each MIF file must be defined in the mapping file before it can be read or written.

When translations are run with enhanced geometry handling turned ON, it enables the MIF reader to read complex geometries like heterogeneous aggregates, and enables the FME to store them.

Note, however, that when enhanced geometry handling is turned ON, the reader's BREAK_COLLECTION directive will be overridden and set to NO.

MIF Quick Facts

Format Type Identifier	MIF
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	Directory or File
Feature Type	File base name
Typical File Extensions	.mif (.mid)
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	Yes
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Enhanced Geometry	Yes
Geometry Type	mif_type
Encoding Support	Yes

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	yes		polygon	yes
circular arc	yes		raster	no
donut polygon	yes		solid	no
elliptical arc	yes		surface	no
ellipses	yes		text	yes
line	yes		z values	no
none	yes			

Reader Overview

The MIF reader first scans the directory it is given for the MIF files defined in the mapping file. For each MIF file that it finds, it checks to see if that file is requested by looking at the list of IDs specified in the mapping file. If a match is made or no IDs were specified in the mapping file, the MIF file is opened. The MIF reader then extracts features from the file one at a time, and passes them on to the rest of the FME for further processing. When the file is exhausted, the MIF reader starts on the next file in the directory.

Optionally a single MIF file can be provided as the dataset. In this case, only that MIF file will be read.

Reader Directives

The directives that are processed by the MIF reader are listed below. The suffix shown is prefixed by the current <ReaderKeyword> in a mapping file. By default, the <ReaderKeyword> for the MIF reader is MIF.

DATASET

Required/Optional: *Required*

The value for this keyword is the directory containing the MIF files to be read, or a single MIF file. A typical mapping file fragment specifying an input MIF dataset looks like:

```
MIF_DATASET /usr/data/mapinfo/92i080
```

Workbench Parameter: *Source MapInfo MIF/MID File(s)*

DEF

Required/Optional: *Optional*

The definition specifies the base name of the file, and the names and the types of all attributes. The syntax of a MIF DEF line is:

```
<ReaderKeyword>_DEF <baseName> \  
[<attrName> <attrType>]+
```

The file names of the physical MIF files is constructed by using the directory specified by the DATASET keyword, the basename specified on the MIF DEF lines, and the .mif (geometry) and .mid (attributes) extensions.

MIF files require at least one attribute to be defined. The attribute definition given must match the definition of the file being read. If it does not, translation is halted and the true definition of the MIF file's attributes gets logged to the log file. There are no restrictions on the field names of MIF attributes.

Tip: MapInfo decimal fields are analogous to DataBase Format (DBF) number fields. MapInfo also provides float, integer, and smallint field types for storing numeric values.

The following table shows the attribute types supported.

Field Type	Description
char(<width>)	Character fields store fixed length strings. The width parameter controls the maximum number of characters that can be stored by the field. No padding is required for strings shorter than this width.
date	Date fields store dates as character strings with the format YYYYMMDD.
datetime	Datetime fields store dates as character

Field Type	Description
	strings with the format YYYYMMDDHHMMSS.FFF
decimal(<width>, <decimals>)	Decimal fields store single and double precision floating point values. The width parameter is the total number of characters allocated to the field, including the decimal point. The decimals parameter controls the precision of the data and is the number of digits to the right of the decimal.
float	Float fields store floating point values. There is no ability to specify the precision and width of the field.
integer	Integer fields store 32 bit signed integers.
logical	Logical fields store TRUE/FALSE data. Data read or written from and to such fields must always have a value of either true or false.
smallint	Small integer fields store 16 bit signed integers and therefore have a range of -32767 to +32767.
time	Time fields store times as character strings with the format HHMMSS.FFF

The following mapping file fragment defines two MIF files. Notice that neither definition specifies the geometric type of the entities it will contain since MIF files may contain any of the valid geometry types.

```
MIF_DEF landcover \
    area decimal(12,3) \
    landcoverType char(11) \
    perimeter float
MIF_DEF roads \
    numberOfLanes smallint \
    roadType char(5) \
    underConstruction logical \
    divided logical \
    travelDirection char(6)
```

IDs

Required/Optional: *Optional*

This optional specification limits the available and defined MIF files read. If no IDs are specified, then all defined and available MIF files are read.

The syntax of the IDs keyword is:

```
<ReaderKeyword>_IDs <baseName1> \
<baseName2> \
<baseNameN>
```

The basenames must match those used in DEF lines.

Workbench Parameter: *Feature Types to Read*

Example: The example below selects only the roads MIF file for input during a translation:

```
MIF_IDS roads
```

BREAK_COLLECTION (applicable only with classic geometry)

Required/Optional: *Optional*

This directive specifies how the MIF collections are processed. If no BREAK_COLLECTION is specified, then all MIF collections are broken down into their component parts before being returned to FME. If a MIF-to-MIF translation is being performed, then this may be set to NO to preserve the collections as single features.

Note that when FME_GEOMETRY_HANDLING is set to YES, this directive will be overridden and set to NO.

Workbench Parameter: *<WorkbenchParameter>*

Example:

This example shows how collections may be preserved:

```
MIF_BREAK_COLLECTION NO
```

ENCODING

This directive is applicable only if you are working with foreign (non-English) character sets.

For example, if your source data contains foreign characters, using this directive along with the encoding value ensures that the original data is preserved from the reader to the writer.

Required/Optional

Optional

Values

Values supported by MapInfo 10:

SJIS, CP437, CP850, CP852, CP855, CP857, CP860, CP861, CP863, CP864, CP865, CP869, CP932, CP936, CP950, CP1250, CP1251, CP1253, CP1254, CP1255, CP1256, ISO8859-1, ISO8859-2, ISO8859-3, ISO8859-4, ISO8859-5, ISO8859-6, ISO8859-7, ISO8859-8, ISO8859-9

Mapping File Syntax

```
<ReaderKeyword>_ENCODING <encoding>
```

*** Workbench Parameter**

Character Encoding (optional)

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The MIF writer creates and writes feature data to MIF files in the directory specified by the DATASET keyword. As with the reader, the directory must exist before the translation occurs. Any old MIF files in the directory are overwritten with the new feature data. As features are routed to the MIF writer, the MIF writer determines the file into which the features are written and outputs them accordingly. Many MIF files can be written during a single FME session.

The version of MIF files produced depends on the data being written. FME automatically writes the lowest possible version that still supports the data. For example, if time or datetime attributes are being written, or the coordinate system is "Krovak S-JTSK", then the version will be set to at least 900; otherwise it will be lower if the data can be supported in a lower version.

When the MIF writer receives a feature with an `fme_color` or `fme_fill_color` attribute, the writer will honor the color values. The only exception is when native MapInfo color settings are also present, in which case the native settings will take precedence.

Writer Directives

The directives that are processed by the MIF writer are listed below. The suffixes shown are prefixed by the current <WriterKeyword> in a mapping file. By default, the <WriterKeyword> for the MIF writer is MIF.

DATASET

Required/Optional: *Required*

The value for this keyword is the directory containing the MIF file(s) to be written.

Workbench Parameter: *Destination MapInfo Directory*

DEF

Required/Optional: *Required*

The MIF writer processes this directive as described in the *Reader Directives* section.

COORDSYS_STATEMENT

Required/Optional: *Required*

The value for this directive is the coordinate system statement that should be written to the header of the produced MIF files. Normally, FME examines the coordinate system information present on the features written to the MIF files, and generates a coordinate system statement based on this information. However, in certain circumstances it is necessary to override this and force a particular coordinate system to be output into the file. This is typically done to force the units of a *non-earth* projection to something other than the default, which are metres.

The syntax of this line is the same as the line defined for the CoordSys line in the MapInfo MIF/MID documentation. For example, to force a non-earth inches coordinate system, this line would be present in the mapping file:

```
MIF_COORDSYS_STATEMENT CoordSys NonEarth Units \"in\"
```

The FME appends bounds information to this statement when it is written to the MIF file. Notice that the quotes must be escaped, as they are required when the coordinate system statement is written to the MIF file.

Workbench Parameter: *Coordinate System Statement*

BOUNDS

This directive allows explicit setting of the bounds of the output features. Because MIF has limited precision available for the storage of coordinates, defining a tight bound on the range of the data can preserve more accuracy. When this directive is specified, the coordinate system string written to the top of the MIF file will contain this bounds specification. The syntax of this directive is:

```
MIF_BOUNDS<xmin> <ymin> <xmax> <ymin>
```

Workbench Parameter: *Bounds Min X, Bounds Min Y, Bounds Max X, and Bounds Max Y*

FILENAME_PREFIX

The value for this directive is prepended to every output file that is created by the writer.

For example, to have the word *temp* appear on the front of every file name, this line would be present in the mapping file:

```
MIF_FILENAME_PREFIX temp
```

Workbench Parameter: *<WorkbenchParameter>*

WRITE_REGION_CENTROIDS

To direct the Writer to output region centroids, the syntax of this directive is:

```
WRITE_REGION_CENTROIDS yes
```

Workbench Parameter: *Generate and Write Region Centroids*

ENCODING

This directive is applicable only if you are working with foreign (non-English) character sets.

For example, if your data contains foreign characters, using this directive along with the encoding value ensures that the original characters are preserved.

Required/Optional

Optional

Values

Values supported by MapInfo 10:

SJIS, CP437, CP850, CP852, CP855, CP857, CP860, CP861, CP863, CP864, CP865, CP869, CP932, CP936, CP950, CP1250, CP1251, CP1253, CP1254, CP1255, CP1256, ISO8859-1, ISO8859-2, ISO8859-3, ISO8859-4, ISO8859-5, ISO8859-6, ISO8859-7, ISO8859-8, ISO8859-9

Mapping File Syntax

```
<WriterKeyword>_ENCODING <encoding>
```

✳ Workbench Parameter

Character Encoding (optional)

Feature Representation

MIF features consist of geometry and attributes. The attribute names are defined in the DEF line and there is a value for each attribute in each MIF feature. In addition, each MIF feature contains several special attributes to hold the type of the geometric entity and its display parameters. All MIF features contain the *mif_type* attribute, which identifies the

geometric type. All MIF features may contain either or both of the `fme_color` and `fme_fill_color` attributes, which store the color and fill color of the feature respectively.

In addition to the generic FME feature attributes that FME Workbench adds to all features (see [About Feature Attributes](#)), this format adds the format-specific attributes described in this section.

Attribute Name	Contents
<code>mif_type</code>	The MIF geometric type of this entity. Range: <code>mif_point </code> <code>mif_polyline </code> <code>mif_region </code> <code>mif_text </code> <code>mif_ellipse </code> <code>mif_arc </code> <code>mif_rectangle </code> <code>mif_rounded_rectangle </code> <code>mif_collection </code> <code>mif_none</code> Default: No default
<code>fme_color</code>	A normalized RGB triplet representing the color of the feature, with format <code>r,g,b</code> . Range: 0,0,0 to 1,1,1 Default: No default
<code>fme_fill_color</code>	A normalized RGB triplet representing the fill color of the feature, with format <code>r,g,b</code> . Range: 0,0,0 to 1,1,1 Default: No default

Points

mif_type: `mif_point`

MIF point features specify a single x and y coordinate in addition to any associated user-defined attributes. An aggregate of point features may also be read or written – this corresponds to the MIF MULTI_POINT primitive type.

A MIF point also specifies a symbol. The symbol is defined by a symbol number, a color, and a size. If no symbol is defined for a point entity, the previous symbol is used.

The table below lists the special FME attribute names used to control the MIF symbol settings.¹

Attribute Name	Contents
<code>mif_symbol_color</code>	The color of the symbol. MapInfo colors are defined in relative concentrations of red, green, and blue. Each color

¹MapInfo symbols cannot be rotated. However, some third-party add-ons to MapInfo rotate symbols based on a user-defined rotation attribute.

Attribute Name	Contents
	ranges from 0 to 255, and the color value is calculated according to the formula: $(red * 65536) + (green * 256) + blue$ Range: 0...2 ²⁴ - 1 Default: 0 (black)
mif_symbol_shape	The number of the symbol. See the <i>MapInfo Reference Manual</i> for a list of the available symbols. Range: 31...67 Default: 35 (a star)
mif_symbol_size	The point size of the symbol. Note that this size is <i>not</i> scaled depending on the zoom level. Range: Any integer number > 0 Default: 10

Font Points

mif_type: mif_font_point

MIF font point are very similar to MIF points, but allow a symbol based on a TrueType font to be specified. In addition to the font, may specify rotation, color, shape number, size, and style.

The table below lists the special FME attribute names used to control the MIF font point settings:

Attribute Name	Contents
mif_symbol_color	The color of the symbol calculated according to the formula: $(red * 65536) + (green * 256) + blue$ Range: 0...2 ²⁴ - 1 Default: 0 (black)
mif_symbol_shape	The number of the shape within the TrueType font to be used as the symbol. Range: Integer Default: No default
mif_symbol_size	The point size of the symbol. Range: Integer Default: 12
mif_symbol_font	The name of the TrueType font to be used for the symbol. Range: String Default: No default

Attribute Name	Contents
mif_symbol_angle	The rotation angle for the symbol, measured in degrees counterclockwise from horizontal. Range: -360.0..360.0 Default: 0
mif_symbol_style	The display style for the symbol. Range: 0 (Plain text) 1 (Bold text) 16 (Black border around symbol) 32 (Drop Shadow) 256 (White border around symbol) Default: 0

Custom Points

mif_type: mif_custom_point

MIF custom points are very similar to MIF points, but allow a bitmap image to be specified as the symbol to be drawn. In addition to the image, color, size, and style may be specified.

The table below lists the special FME attribute names used to control the MIF custom point settings:

Attribute Name	Contents
mif_symbol_color	The color of the symbol calculated according to the formula: $(red * 65536) + (green * 256) + blue$ Whether or not the color is used depends on the setting of the style attribute. Range: 0... $2^{24} - 1$ Default: 0 (black)
mif_symbol_file_name	The name of the bitmap file found in the MapInfo Cust-Symb directory. Range: String Default: No default
mif_symbol_size	The point size of the symbol. Range: Integer Default: 12
mif_symbol_style	The display style for the symbol. Range: 0 (White pixels in the image are transparent, allowing whatever is beneath to show through. Non-white pixels are drawn in the same color as they are in the bitmap.) 1 (White pixels in the image are drawn as white. Non-white pixels are drawn in the same color as they are in

Attribute Name	Contents
	<p>the bitmap.)</p> <p>2 (White pixels in the image are transparent. Non-white pixels are drawn in the color specified by mif_symbol_color.)</p> <p>3 (White pixels in the image are drawn in white. Non-white pixels are drawn in the color specified by mif_symbol_color)</p> <p>Default: 0</p>

Multipoints

mif_type: mif_point, mif_font_point, mif_custom_point

MIF multipoint feature specify a number of individual sets of points each defined by an x and y coordinate. All the points share the same attributes and geometry. This is supported as a homogeneous aggregate feature composed of points, font points or custom points.

The MIF multipoint uses the same attribute names control settings as the points, font points and custom point.

Polylines

mif_type: mif_polyline

MIF polyline features specify linear features defined by a sequence of x and y coordinates. Each polyline has a pen style associated with it specifying the color, width, and pen pattern of the line. A polyline may also specify that it is a smoothed line, in which case MapInfo uses a curve fitting algorithm when rendering the line¹. If no pen style is defined, the previous style is used.

Tip: MapInfo MIF supports a special type for two point lines. The FME transparently converts such MIF lines into polylines, both as it reads MIF files and as it writes them.

The table below lists the special FME attribute names used to control the MIF polyline settings.

Attribute Name	Contents
mif_pen_color	<p>The color of the polyline. MapInfo colors are defined in relative concentrations of red, green, and blue. Each color ranges from 0 to 255, and the color value is calculated according to the formula:</p> $(\text{red} * 65536) + (\text{green} * 256) + \text{blue}$ <p>Range: 0...2²⁴ - 1 Default: 0 (black)</p>
mif_pen_pattern	<p>The pattern used to draw the line. See the <i>MapInfo Reference Manual</i> for a list of the available patterns.</p> <p>Range: 1...77 Default: 2</p>
mif_pen_width	<p>The width of the line rendered for the polyline feature. This is measured as a thickness in pixels. A width of 1 is always drawn as a hairline. A width of 0 should be considered to be a line with no width, or a line with no style, or</p>

¹MapInfo renders smoothed polylines substantially slower than unsmoothed polylines.

Attribute Name	Contents
	invisible, and should not normally be used. If an invisible line is necessary, it should be created by setting the pattern to 1 (None). If a hairline is desired, the pen should be created by setting the width to 1. The width can be specified as a point width, in which case this formula is used: $\text{penwidth} = (\text{point width} * 10) + 10$ Range: 0...7 (pixel width) 11...2047 (point width) Default: 1
mif_smooth	Controls whether or not the polyline will be smoothed when rendered. Range: true false Default: false

Regions

mif_type: mif_region

MIF region features specify area (polygonal) features. The areas that make up a single feature may or may not be disjoint, and may contain polygons that have holes. Each region has a pen style associated with it to control the color, width, and pen pattern used when its boundary is drawn. In addition, a region may set its brush pattern, foreground, and background color to control how its enclosed area will be filled. If no pen or brush style is defined for a region entity, the previous style is used. The following table lists the special FME attribute names used to control the MIF region settings.

Attribute Name	Contents
mif_brush_pattern	The pattern used to fill the area the region contains. See the <i>MapInfo Reference Manual</i> for a list of the available brush patterns. Range: 1...71 Default: 2 (solid)
mif_brush_foreground	The foreground color used when the region is filled. MapInfo colors are defined in relative concentrations of red, green, and blue. Each color ranges from 0 to 255, and the color value is calculated according to the formula: $(\text{red} * 65536) + (\text{green} * 256) + \text{blue}$ Range: $0 \dots 2^{24} - 1$ Default: 0 (black)
mif_brush_background	The background color used when the region is filled. (-1 specifies transparent color) Range: $-1 \dots 2^{24} - 1$ Default: 16777215 (white)
mif_pen_color	The color of the boundary of the region. Range: $0 \dots 2^{24} - 1$ Default: 0 (black)

Attribute Name	Contents
mif_pen_pattern	The pattern used to draw the region's boundary. See the <i>MapInfo Reference Manual</i> for a list of the available patterns. Range: 1...77 Default: 2
mif_pen_width	The width of the line rendered for the region's boundary. This is measured as a thickness in pixels. A width of 1 is always drawn as a hairline. A width of 0 should be considered to be a line with no width, or a line with no style, or invisible, and should not normally be used. If an invisible line is necessary, it should be created by setting the pattern to 1 (None). If a hairline is desired, the pen should be created by setting the width to 1. Range: 0...35 Default: 1
mif_center_xcoord	The centroid x coordinate. Range: Any real number Default: 0
mif_center_ycoord	The centroid y coordinate. Range: Any real number Default: 0

Text

mif_type: mif_text

MIF text features are used to specify annotation information. Each text feature can have its font, color, spacing, justification, and rotation angle set independently. The following table lists the special FME attribute names used to control the MIF text settings.

Attribute Name	Contents
mif_rotation	The rotation of the text, as measured in degrees counterclockwise from horizontal. Range: -360.0..360.0 Default: 0
mif_text_fontbgcolor	The background color used when the text is drawn. Range: 0...2 ²⁴ - 1 Default: 16777215 (white)
mif_text_fontfgcolor	The foreground color used when the text is drawn. MapInfo colors are defined in relative concentrations of red, green, and blue. Each color ranges from 0 to 255, and the color value is calculated according to the formula:

Attribute Name	Contents
	<p>(red*65536) + (green*256) + blue Range: 0...2²⁴ - 1 Default: 0 (black)</p>
mif_text_fontname	<p>The name of the font used to draw the text. The font named must be available on the destination computer system. Range: Default: Arial</p>
mif_text_fontstyle	<p>The style code of the text. This flag controls whether the text is bold, underlined, italic, etc. See the <i>MapInfo Reference Manual</i> for a list of style codes and their meanings. The basic range of possible flag settings are listed below. Combinations of various values are also allowed. For example, a value of 6 indicates a bold and italic text style: Range: 0 - Plain 1 - Bold 2 - Italic 4 - Underline 16 - Outline (only supported on the Macintosh) 32 - Shadow 256 - Halo 512 - All Caps 1024 - Expanded Default: 0 (plain text)</p>
mif_text_height	<p>The height of the text in ground units. Range: Any real number >= 0 Default: 10</p>
mif_text_justification	<p>The justification of the text. Range: left center right Default: left</p>
mif_text_spacing	<p>The spacing between lines of multiline text. The measure is expressed as a multiple of the text height. Range: 1.0 1.5 2.0 Default: 1.0</p>
mif_text_string	<p>The text to be displayed. Range: Any character string Default: No default</p>
mif_text_width	<p>The total width of the text string in ground units. The</p>

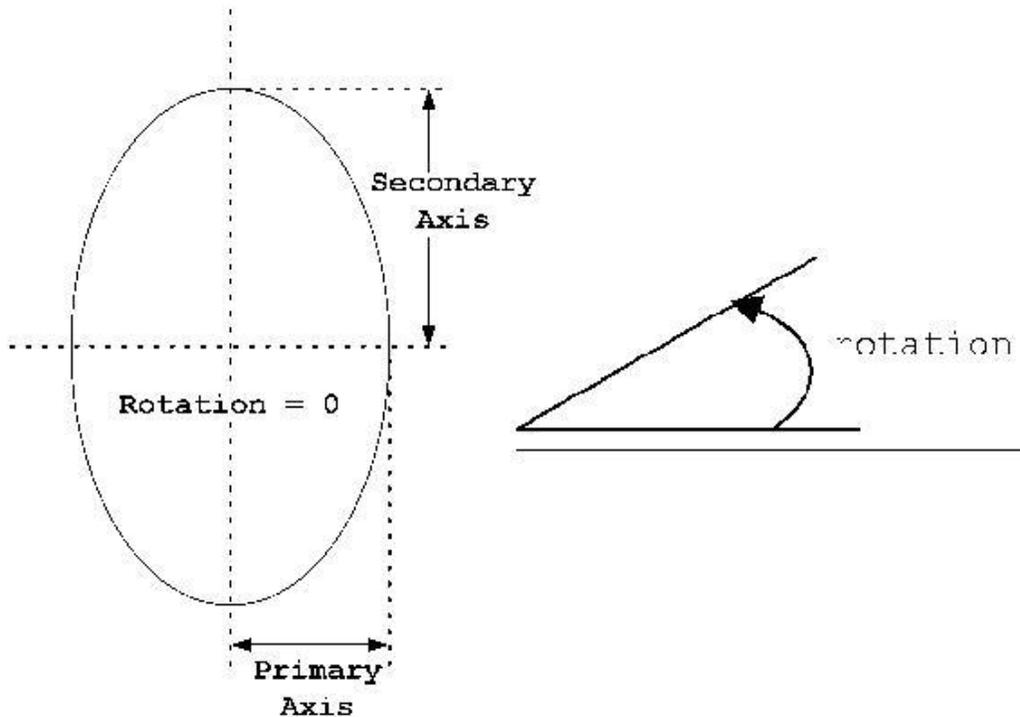
Attribute Name	Contents
	<p>MIF text representation stores a bounding box for the text, and <code>mif_text_width</code> is the width of the bounding box.</p> <p>Range: Any real number ≥ 0</p> <p>Default: 10</p>
<code>mif_text_linetype</code>	<p>The type of line attaching the text to the anchor point.</p> <p>Range: 0 (None: do not display a line with the label.) 1 (Simple: create a callout by using a simple line that connects the label to the anchor point.) 2 (Arrow: create a callout by using an arrow and line that connects the label to anchor point.)</p> <p>Default: 0 (None)</p>
<code>mif_text_line_end_x</code>	<p>The x position of the label line end point. The linetype needs to be 1 or 2 for the label line to be visible.</p> <p>Range: Any real number</p> <p>Default: No default</p>
<code>mif_text_line_end_y</code>	<p>The y position of the label line end point. The linetype needs to be 1 or 2 for the label line to be visible.</p> <p>Range: Any real number</p> <p>Default: No default</p>

Tip: The font color and style settings will not be used unless a font name is specified.

Ellipse

mif_type: `mif_ellipse`

MIF ellipse features are point features, and have only a single coordinate. This point serves as the centre of the ellipse. Additional attributes specify the primary axis and secondary axis of the ellipse. MIF ellipses currently do not support rotation. For compatibility with other systems, the MIF reader always returns a rotation of 0. If a rotation is specified to the writer, the ellipse is turned into a region, vectorized, and rotated by the amount specified.



*Tip: The primary ellipse axis is **not** necessarily the longest axis, but rather the one on the x axis.*

In addition to the attributes below, ellipses also make use of the brush and pen attributes as defined by mif_region.

Attribute Name	Contents
mif_primary_axis	The length of the semi-major axis in ground units. Range: Any real number > 0 Default: No default
mif_secondary_axis	The length of the semi-minor axis in ground units. Range: Any real number > 0 Default: No default
mif_rotation	The rotation of the major axis. The rotation is measured in degrees counterclockwise up from horizontal. Range: -360.0..360.0 Default: 0

Arc

mif_type: mif_arc

MIF arc features are linear features used to specify elliptical arcs. As such, the feature definition for mif_arc is similar to the ellipse definition with two additional angles to control the portion of the ellipse boundary drawn. MIF arcs currently do not support rotation. For compatibility with other systems, the MIF reader always returns a rotation of 0. In addition, if a rotation is specified to the writer, the arc is turned into a polyline, vectorized, and rotated by the amount specified.

Tip: The function @Arc() can be used to convert an arc to a linestring. This is useful for storing Arcs in systems not supporting them directly.

In addition to the attributes below, arcs also make use of the pen attributes as defined on mif_polyline.

Attribute Name	Contents
mif_primary_axis	The length of the semi-major axis in ground units. Range: Any real number > 0 Default: No default
mif_secondary_axis	The length of the semi-minor axis in ground units. Range: Any real number > 0 Default: No default
mif_start_angle	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of start_angle. Range: 0.0..360.0 Default: 0
mif_sweep_angle	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of sweep_angle. Range: 0.0..360.0 Default: No default
mif_rotation	The rotation of the major axis. The rotation is measured in degrees counterclockwise up from horizontal. Range: -360.0..360.0 Default: 0

Rectangle

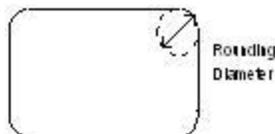
mif_type: mif_rectangle

MIF rectangle objects are represented in the FME as closed polygons. When a MIF rectangle is read, it is turned into a closed polygon feature. When a MIF rectangle is written, the minimum bounding rectangle of the feature is taken and used as the four corners of the rectangle. MIF rectangles take the same additional attributes as MIF regions to specify their brush and pen.

Rounded Rectangle

mif_type: mif_rounded_rectangle

MIF rounded rectangle objects are represented in the FME as closed polygons. When a MIF rounded rectangle is read, it is turned into a closed polygon feature and the corners are vectorized to preserve the intended shape of the rectangle. The rounding radius is also stored as an attribute. When a MIF rounded rectangle is written, the minimum bounding rectangle of the feature is taken and used as the four corners of the rectangle, and the rounding diameter is taken from an attribute of the feature. MIF rounded rectangles take the same additional attributes as MIF regions to specify their brush and pen.



Attribute Name	Contents
mif_rounding	Contains the diameter in ground unit, of the circle used to produce the rounded corners. Range: Any real number > 0 Default: No default

Collection

mif_type: mif_collection

MIF collections are defined as a combination of the other feature types. This is represented as nonhomogeneous aggregates composed of the other feature types.

To create MapInfo collections using FME, set the mif_type attribute to mif_collection on the feature destined for the MIF dataset. It is important that the feature to be saved as a collection is an aggregate feature.

The table below lists the special FME attribute name used to control the MIF collection settings:

Attribute Name	Contents
mif_collection_comp{ Deprecated	This is the list attribute prefix used to store the attributes for each collection part. The suffixes are the attribute names for the control settings of the other feature types. Range: none Default: none

MapInfo TAB Reader/Writer

The MapInfo Native Format Reader and Writer modules provide FME with the ability to read and write directly to MapInfo files. The MapInfo Native Format is a proprietary format used by the MapInfo Professional Desktop mapping product. MapInfo Native format files are often called Tab files.

The MapInfo Native Format reader and writer are closely patterned after the MapInfo MIF/MID reader and writer. This commonality makes it easy to support both MIF and MapInfo Native formats in the same mapping file.

Overview

MapInfo is a two-dimensional system with no provision for transferring elevation data for each vertex in a MapInfo feature. However, point features can define an elevation attribute to store their elevation.

MapInfo files store both feature geometry and attributes. A logical MapInfo file consists of several physical files, having the following file name extensions:

File Name Extension	Contents
.tab	The main file for a MapInfo table, it is associated with the appropriate DAT, MAP, ID, and IND files.
.dat, .dbf, .mdb, .accdb, .xls	Tabular data for a table in MapInfo's native format (.dat), dBASE format (.dbf), MS Access format (.mdb or .accdb) or MS Excel format (.xls). MS Access and MS Excel formats are only supported when using the MITAB reader.
.id	An index to a MapInfo graphical objects (MAP) file.
.map	Contains geographic information describing map objects.
.ind	An index to a MapInfo tabular (DAT) file.

These extensions are added to the basename of the specified MapInfo file. Throughout the remainder of this chapter, references to "file" are references to the logical MapInfo file, not the multiple physical files that make it up.

The MapInfo reader and writer support the storage of point, line, polyline, arc, ellipse, rectangle, rounded rectangle, region (polygon), and text geometric data. The MapInfo format also stores features with no geometry. Features having no geometry are referred to as having a geometry of *none*.

Each geometric entity present in MapInfo has display properties, such as pen and brush width, pattern, and color. In addition, each entity has a row of attributes associated with it. A single MapInfo map file can contain many different types of geometry however, the associated attributes must have the same number and type of fields for each entity in the file.

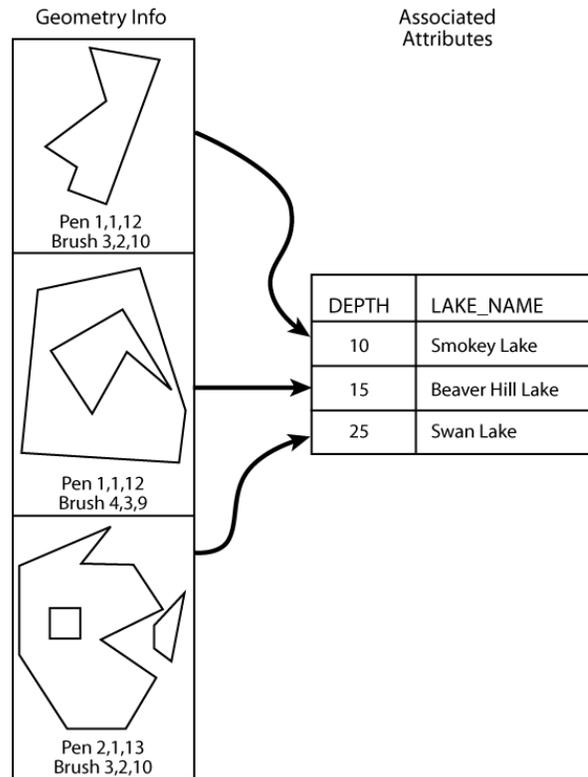
The MapInfo reader can open datasets in which the attributes are stored as .dat, .dbf, .mdb, .accdb, or .xls files. In other words, the MapInfo reader supports datasets of Type NATIVE, Type DBF, Type ACCESS and Type XLS. Types ACCESS and XLS are only supported when using the MITAB reader

In addition, the MapInfo reader also supports datasets of Type FME. These are files that store all of their auxiliary information in a separate FME dataset. The auxiliary dataset can be in any format that the FME supports for reading. MapInfo files of type FME are created in MapInfo 9.0 and later with the "Open Universal Data" option.

The number and type of attributes associated with each entity is specified by the user. There must be at least one attribute field defined before a MapInfo file can be created.

The following illustration shows a MapInfo file containing three *region* entities. Note that the second polygon contains a hole while the third polygon is an aggregate of two disjoint polygons, one of which contains a hole. Each geometric entity in turn corresponds with one record in the attribute table.

FME considers a MapInfo dataset to be a collection of tab files and related files in a single directory. The attribute definitions for each MapInfo file set must be defined in the mapping file before it can be read or written.



When translations are run with enhanced geometry handling turned ON, it enables the MapInfo reader to read complex geometries like heterogeneous aggregates, and enables the FME to store them.

Note, however, that when enhanced geometry handling is turned ON, the reader's `BREAK_COLLECTION` directive will be overwritten and set to NO.

MapInfo Quick Facts

Format Type Identifier	MAPINFO MITAB
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	Directory or File
Feature Type	File base name
Typical File Extensions	.tab (.dat, .id, .map, .ind)
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	Yes
Spatial Index	Always
Schema Required	Yes
Transaction Support	No
Enhanced Geometry	Yes
Geometry Type	mapinfo_type
Encoding Support	Yes

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	yes		polygon	yes
circular arc	yes		raster	no
donut polygon	yes		solid	no
elliptical arc	yes		surface	no
ellipses	yes		text	yes
line	yes		z values	no
none	yes			

Reader Overview

The MapInfo reader first scans the directory it is given for the MapInfo files which have been defined in the mapping file. For each logical MapInfo file that it finds, it checks to see if that file is requested by looking at the list of **IDs** specified in the mapping file. If a match is made, or no **IDs** were specified in the mapping file, the MapInfo file is opened. The MapInfo reader then extracts features from the file one at a time, and passes them on to the rest of the FME for further processing. When the file is exhausted, the MapInfo reader starts on the next file in the directory.

Optionally, a single MapInfo file can be given as the dataset. In this case, only that MapInfo file is read.

Reader Directives

The directives that are processed by the MapInfo reader are listed below. The suffix shown is prefixed by the current <ReaderKeyword> in a mapping file. By default, the <ReaderKeyword> for the MIF reader is MAPINFO.

DATASET

Required/Optional: *Required*

The value for this keyword is the directory containing the MapInfo files to be read, or a single MapInfo file. A typical mapping file fragment specifying an input MapInfo dataset looks like:

```
MAPINFO_DATASET /usr/data/mapinfo/92i080
```

Workbench Parameter: *Source MapInfo TAB File(s)*

DEF

Required/Optional: *Optional*

The definition specifies the base name of the file, and the names and types of all attributes. The syntax of a MapInfo DEF line is:

```
<ReaderKeyword>_DEF <baseName> [<attrName> <attrType>[,indexed]]+
```

The file names of the physical MapInfo files are constructed by using the directory specified by the DATASET keyword, the basename specified on the MapInfo DEF lines, and the file extensions.

MapInfo requires that at least one attribute be defined. The attribute definition given must match the definition of the file being read. If it does not, translation is halted and the true definition of the MapInfo file attributes are logged to the log file. There are no restrictions on the field names of MapInfo attributes. The following table shows the attribute types which are supported.

Field Type	Description
char(<width>)	Character fields store fixed length strings. The width parameter controls the maximum of characters stored by the field. No padding is required for strings shorter than this width.
date	Date fields store dates as character strings with the format dependent on your location. This format is usually YYYYMMDD.
datetime	Datetime fields store dates as character strings with the format YYYYMMDDHHMMSS.FFF
decimal(<width>, <decimals>)	Decimal fields store single and double precision floating point values. The width parameter is the total number of characters allocated to the field, including the decimal point. The decimals parameter controls the precision of the data and is the number of digits to the right of the decimal.
float	Float fields store floating point values. There is no ability to specify the precision and width of the field.

Field Type	Description
integer	Integer fields store 32 bit signed integers.
logical	Logical fields store boolean data. Data read or written from/to such fields must always have a value of either true or false.
smallint	Small integer fields store 16 bit signed integers, and therefore have a range of -32767 to +32767.
time	Time fields store times as character strings with the format HHMMSS.FFF

The attribute type may also have **indexed** when the definition is specified for a writer. When specified, this results in the writer building an attribute index table for the columns that are indexed thereby making queries in MapInfo faster. This directive is only recognized by the writer module.

The following mapping file fragment defines two MapInfo files. Notice that neither definition specifies the geometric type of the entities it will contain because MapInfo files may contain any of the valid geometry types.

```

MAPINFO_DEF landcover \
  area          decimal(12,3) \
  landcoverType char(11) \
  perimeter     float

MAPINFO_DEF roads \
  numberOfLanes smallint \
  roadType      char(5) \
  underConstruction logical \
  divided       logical \
  travelDirection char(6)

```

IDs

Required/Optional: *Optional*

This specification is used to limit the MapInfo files that are read. If no **IDs** are specified, then all defined and available MapInfo files are read. The syntax of the **IDs** keyword is:

```

<ReaderKeyword>_IDS <baseName1> \
  <baseName2> \
  <baseNameN>

```

The basenames must match those used in **DEF** lines.

The example below selects only the **roads** MapInfo file for input during a translation:

```

MAPINFO_IDS roads

```

Workbench Parameter: *Feature Types to Read*

BREAK_COLLECTION (applicable only with classic geometry)

Required/Optional: *Optional*

This directive specifies how the MapInfo collections are processed. If no **BREAK_COLLECTION** is specified, then all MapInfo collections are broken down into their component parts before being returned to FME. If a MapInfo-to-MapInfo translation is being performed, then this may be set to **NO** to preserve the collections as single features.

Note that when enhanced geometry handling is turned ON, this directive will be overwritten and set to NO.

This example shows how collections may be preserved:

```
MAPINFO_BREAK_COLLECTION NO
```

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the `mitab.dll` in the FME home directory to `mapinfo.dll`.

The syntax of the `MAPINFO_SEARCH_ENVELOPE` directive is:

```
MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

```
MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1
```

FME_TABLE_PASSWORD

Required/Optional: *Optional*

This keyword is only applicable when opening datasets of Type FME. If the auxiliary FME dataset is a database reader, then MapInfo will not automatically put the password for the source database in the tab file. This directive allows the user to specify the password dynamically. Additionally, if the password is hardcoded into the tab file, then this directive will supersede that password.

The syntax of the `FME_TABLE_PASSWORD` directive is:

```
<ReaderKeyword>_FME_TABLE_PASSWORD password
```

This directive is optional. However, if it is needed to open the FME dataset, then it will need to be provided on both generation and runtime.

Workbench Parameter: *Password for FME Table*

ENCODING

This directive is applicable only if you are working with foreign (non-English) character sets.

For example, if your source data contains foreign characters, using this directive along with the encoding value ensures that the original data is preserved from the reader to the writer.

Required/Optional

Optional

Values

Values supported by MapInfo 10:

SJIS, CP437, CP850, CP852, CP855, CP857, CP860, CP861, CP863, CP864, CP865, CP869, CP932, CP936, CP950, CP1250, CP1251, CP1253, CP1254, CP1255, CP1256, ISO8859-1, ISO8859-2, ISO8859-3, ISO8859-4, ISO8859-5, ISO8859-6, ISO8859-7, ISO8859-8, ISO8859-9

Mapping File Syntax

```
<ReaderKeyword>_ENCODING <encoding>
```

* Workbench Parameter

Character Encoding (optional)

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the `mitab.dll` in the FME home directory to `mapinfo.dll`.

The syntax of the `MAPINFO_SEARCH_ENVELOPE` directive is:

```
MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

```
MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The `COORDINATE_SYSTEM` directive, which specifies the coordinate system associated with the data to be read, must always be set if the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` to the reader `COORDINATE_SYSTEM` prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the `SEARCH_ENVELOPE` directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The MapInfo writer creates and writes feature data to MapInfo files in the directory specified by the **DATASET** keyword. If the directory does not exist, the writer has to create it. Any old MapInfo files in the directory are overwritten with the new feature data. As features are routed to the MapInfo writer, the MapInfo writer determines the file into which the features are to be written and outputs them accordingly. Many MapInfo files can be written during a single FME session.

The version of TAB files produced depends on the data being written. FME automatically writes the lowest possible version that still supports the data. For example, if time or datetime attributes are being written, or the coordinate system is "Krovak S-JTSK", then the version will be set to at least 900; otherwise it will be lower if the data can be supported in a lower version.

When the MapInfo writer receives a feature with an **fme_color** or **fme_fill_color** attribute, the writer will honor the color values. The only exception is when native MapInfo color settings are also present, in which case the native settings will take precedence.

Writer Directives

The directives that are processed by the MAPINFO writer are listed below. The suffixes shown are prefixed by the current <WriterKeyword> in a mapping file. By default, the <WriterKeyword> for the MapInfo writer is MAPINFO.

The MapInfo writer processes the DATASET and DEF keywords as described in the Reader Directives section. It does, however, make use of some additional directives:

DATASET

Required/Optional: *Required*

Contains the directory name of the output MapInfo files.

Workbench Parameter: *Destination Directory*

DEF

Required/Optional: *Required*

Defines a MapInfo file. The definition contains the file's base name (without any of the extensions), and the definitions of the attributes. There may be many DEF lines, one for each file to be written.

COORDSYS_STATEMENT

Required/Optional: *Optional*

The value for this directive is the coordinate system statement that should be used in the produced MapInfo files. Normally, FME examines the coordinate system information present on the features written to the files, and outputs the coordinate system based on this information. However, in certain circumstances it is necessary to override this and force a particular coordinate system to be output. This is typically done to force the units of a *non-earth* projection to something other than the default, which is metres.

The syntax of this line is the same as that defined for the **CoordSys** line in the MapInfo MIF/MID documentation. For example, to force a non-earth inches coordinate system, this line would be present in the mapping file:

```
MAPINFO_COORDSYS_STATEMENT CoordSys NonEarth Units \"in\"
```

Notice that the quotes must be escaped, as they are required when the coordinate system statement is interpreted by the MapInfo Writer.

Workbench Parameter: *Coordinate System Statement*

BOUNDS

Required/Optional: *Optional*

This directive allows explicit setting of the bounds of the output features. Because MapInfo has limited precision available for the storage of coordinates, defining a tight bound on the range of the data can preserve more accuracy. The syntax of this directive is:

```
MAPINFO_BOUNDS <xmin> <ymin> <xmax> <ymax>
```

Workbench Parameter: *Bounds Min X, Bounds Min Y, Bounds Max X, Bounds Max Y*

BUILD_OPTIMAL_SPATIAL_INDEX

Required/Optional: *Optional*

This directive tells the MapInfo writer to create an optimal spatial index when writing. This will allow for faster spatial queries on the resulting file when using MapInfo Pro or other software that takes advantage of built in spatial indexing. The use of this directive will, however, slow down the writing of the file. The default value for the directive is no. The syntax of this directive is:

```
MAPINFO_BUILD_OPTIMAL_SPATIAL_INDEX <yes|no>
```

Workbench Parameter: *Build Optimal Spatial Index*

Note: This writer directive pertains only to the MITAB writer.

USE_SOURCE_BOUNDING_BOX

The **USE_SOURCE_BOUNDING_BOX** directive tells the MapInfo writer to attempt to use any bounding box information that the Reader for the current FME session can provide it to set its bounds. This will only be used when no coordinate system is set for the MapInfo writer.

Currently, only the Shape reader in FME provides bounding box information, so setting this directive to **YES** will only have an effect if a Shape to MapInfo translation is being performed without any coordinate system being set.

Note: This is a writer directive and applies only to the MapInfo writer. Its use is discouraged.

FILENAME_PREFIX

Required/Optional: *Optional*

The value for this keyword is prepended to every output file that is created by the writer.

For example, to have the word **temp** appear on the front of every file name, this line would be present in the mapping file:

```
MAPINFO_FILENAME_PREFIX temp
```

WRITE_REGION_CENTROIDS

Required/Optional: *Optional*

To direct the Writer to output region centroids, the syntax of this directive is:

```
WRITE_REGION_CENTROIDS yes
```

Workbench Parameter: *Generate and Write Region Centroids*

STROKE_ARCS

Required/Optional: *Optional*

Indicates whether the arcs will be vectorized before writing. If yes, all arcs will be converted to polylines. This option may be useful where sweep angles have precision finer than 0.1 degree.

```
STROKE_ARCS yes
```

Workbench Parameter: *Stroke arcs into polyline*

ENCODING

This directive is applicable only if you are working with foreign (non-English) character sets.

For example, if your data contains foreign characters, using this directive along with the encoding value ensures that the original characters are preserved.

Required/Optional

Optional

Values

Values supported by MapInfo 10:

SJIS, CP437, CP850, CP852, CP855, CP857, CP860, CP861, CP863, CP864, CP865, CP869, CP932, CP936, CP950, CP1250, CP1251, CP1253, CP1254, CP1255, CP1256, ISO8859-1, ISO8859-2, ISO8859-3, ISO8859-4, ISO8859-5, ISO8859-6, ISO8859-7, ISO8859-8, ISO8859-9

Mapping File Syntax

```
<writerKeyword>_ENCODING <encoding>
```

*** Workbench Parameter**

Character Encoding (optional)

Feature Representation

MapInfo features consist of geometry and attributes. The attribute names are defined in the **DEF** line and there is a value for each attribute in each FME MapInfo feature.

In addition, each MapInfo FME feature contains several special attributes to hold the type of the geometric entity and its display parameters. All MapInfo FME features contain the **mapinfo_type** attribute, which identifies the geometric type. All MapInfo features may contain either or both of the **fme_color** and **fme_fill_color** attributes, which store the color and fill color of the feature respectively.

In addition to the generic FME feature attributes that FME Workbench adds to all features (see [About Feature Attributes](#)), this format adds the format-specific attributes described in this section.

Attribute Name	Contents
mapinfo_type	<p>The MapInfo geometric type of this entity.</p> <p>Range:</p> <ul style="list-style-type: none"> mapinfo_point mapinfo_polyline mapinfo_region mapinfo_text mapinfo_ellipse mapinfo_arc mapinfo_rectangle mapinfo_rounded_rectangle mapinfo_collection mapinfo_none <p>Default: No default</p>
fme_color	<p>A normalized RGB triplet representing the color of the feature, with format r,g,b.</p> <p>Range: 0,0,0 to 1,1,1</p> <p>Default: No default</p>
fme_fill_color	<p>A normalized RGB triplet representing the fill color of the feature, with format r,g,b.</p> <p>Range: 0,0,0 to 1,1,1</p> <p>Default: No default</p>

Points

mapinfo_type: mapinfo_point

MapInfo point features specify a single **x** and **y** coordinate in addition to any associated user-defined attributes. An aggregate of point features may also be read or written – this corresponds to the MapInfo **MULTI_POINT** primitive type.

A MapInfo point also specifies a symbol. The symbol is defined by a symbol number, a color, and a size.¹ If no symbol is defined for a point entity, the previous symbol is used. The table below lists the special FME attribute names used to control the MapInfo symbol settings.

Attribute Name	Contents
mapinfo_symbol_color	<p>The color of the symbol. MapInfo colors are defined in relative concentrations of red, green, and blue. Each color ranges from 0 to 255, and the color value is calculated according to the formula:</p> $(\text{red} * 65536) + (\text{green} * 256) + \text{blue}$ <p>Range: $0 \dots 2^{24} - 1$</p>

¹MapInfo symbols cannot be rotated. However, some third-party add-ons to MapInfo will rotate symbols based on a user-defined rotation attribute.

Attribute Name	Contents
	Default: 0 (black)
mapinfo_symbol_shape	The number of the symbol. See the <i>MapInfo Reference Manual</i> for a list of the available symbols. Range: 31...67 Default: 35 (a star)
mapinfo_symbol_size	The point size of the symbol. Note that this size is <i>not</i> scaled depending on the zoom level. Range: Any integer number > 0 Default: 10

Font Points

mapinfo_type: mapinfo_font_point

MapInfo font points are very similar to MapInfo points, but allow a symbol based on a TrueType font to be specified. In addition to the font, a rotation, color, shape number, size, and style may be specified.

The table below lists the special FME attribute names used to control the MapInfo font point settings:

Attribute Name	Contents
mapinfo_symbol_color	The color of the symbol calculated according to the formula: $(\text{red} * 65536) + (\text{green} * 256) + \text{blue}$ Range: 0...2 ²⁴ - 1 Default: 0 (black)
mapinfo_symbol_shape	The number of the shape within the TrueType font to use as the symbol. Range: Integer Default: No default
mapinfo_symbol_size	The point size of the symbol. Range: Integer Default: 12
mapinfo_symbol_font	The name of the TrueType font to be used for the symbol. Range: String Default: No default
mapinfo_symbol_angle	The rotation angle for the symbol, measured in degrees counterclockwise from horizontal. Range: -360.0..360.0 Default: 0
mapinfo_symbol_style	The display style for the symbol. Range: 0 (Plain text)

Attribute Name	Contents
	1 (Bold text) 16 (Black border around symbol) 32 (Drop Shadow) 256 (White border around symbol) Default: 0

Custom Points

mapinfo_type: mapinfo_custom_point

MapInfo custom points are very similar to MapInfo points, but allow a bitmap image to be specified as the symbol to be drawn. In addition to the image, color, size, and style may be specified.

The table below lists the special FME attribute names used to control the MapInfo custom point settings:

Attribute Name	Contents
mapinfo_symbol_color	The color of the symbol calculated according to the formula: $(\text{red} * 65536) + (\text{green} * 256) + \text{blue}$ Whether or not the color is used, depends on the setting of the style attribute. Range: 0...2 ²⁴ - 1 Default: 0 (black)
mapinfo_symbol_file_name	The name of the bitmap file found in the MapInfo CustSymb directory. Range: String Default: No default
mapinfo_symbol_size	The point size of the symbol. Range: Integer Default: 12
mapinfo_symbol_style	The display style for the symbol. Range: 0 (White pixels in the image are transparent, allowing whatever is beneath to show through. Non-white pixels are drawn in the same color as they are in the bitmap.) 1 (White pixels in the image are drawn as white. Non-white pixels are drawn in the same color as they are in the bitmap.) 2 (White pixels in the image are transparent. Non-white pixels are drawn in the color specified by mapinfo_symbol_color.) 3 (White pixels in the image are drawn in white. Non-white pixels are drawn in the color specified by mapinfo_symbol_color)

Attribute Name	Contents
	Default: 0

Multipoints

mapinfo_type: mapinfo_point, mapinfo_font_point, mapinfo_custom_point

MapInfo multipoint is supported as a homogeneous aggregate feature composed of points, font points or custom points.

The MapInfo multipoint uses the same attribute names control settings as the points, font points and custom point.

Polylines

mapinfo_type: mapinfo_polyline

MapInfo polyline features specify linear features defined by a sequence of x and y coordinates. Each polyline has a pen style associated with it that specifies the color, width, and pen pattern of the line. A polyline may also specify that it is a smoothed line¹, in which case MapInfo uses a curve fitting algorithm when rendering the line. If no pen style is defined, the previous style is used.

Tip: MapInfo supports a special type for two point lines. FME transparently converts such lines into polylines, both as it reads and as it writes them.

The table below lists the special FME attribute names used to control the MapInfo polyline settings.

Attribute Name	Contents
mapinfo_pen_color	The color of the polyline. MapInfo colors are defined in relative concentrations of red, green, and blue. Each color ranges from 0 to 255, and the color value is calculated according to the formula: (red*65536) + (green*256) + blue Range: 0...2 ²⁴ - 1 Default: 0 (black)
mapinfo_pen_pattern	The pattern used to draw the line. See the <i>MapInfo Reference Manual</i> for a list of the available patterns. Range: 1...77 Default: 2
mapinfo_pen_width	The width of the line rendered for the polyline feature. This is measured as a thickness in pixels. A width of 1 is always drawn as a hairline. A width of 0 should be considered to be a line with no width, or a line with no style, or invisible, and should not normally be used. If an invisible line is necessary, it should be created by setting the pattern to 1 (None). If a hairline is desired, the pen should be created by setting the width to 1. The width can be specified as a point width, in which case this formula is used: penwidth = (point width * 10) + 10

¹MapInfo renders smoothed polylines substantially slower than unsmoothed polylines.

Attribute Name	Contents
	Range: 0...7 (pixel width) 11...2047 (point width) Default: 1
mapinfo_smooth	Controls whether or not the polyline will be smoothed when rendered. Range: true false Default: false

Regions

mapinfo_type: mapinfo_region

MapInfo region features specify area (polygonal) features. The areas that make up a single feature may or may not be disjoint, and may contain polygons which have holes. Each region has a pen style associated with it to control the color, width, and pen pattern used when its boundary is drawn. In addition, a region may set its brush pattern, foreground, and background color to control how the area it encloses will be filled.

The following table lists the special FME attribute names used to control the MapInfo region settings.

Attribute Name	Contents
mapinfo_brush_pattern	The pattern used to fill the area the region contains. See the <i>MapInfo Reference Manual</i> for a list of the available brush patterns. Range: 1...71 Default: 2 (solid)
mapinfo_brush_foreground	The foreground color used when the region is filled. MapInfo colors are defined in relative concentrations of red, green, and blue. Each color ranges from 0 to 255, and the color value is calculated according to the formula: $(red*65536) + (green*256) + blue$ Range: 0...2 ²⁴ - 1 Default: 0 (black)
mapinfo_brush_background	The background color used when the region is filled. (-1 specifies transparent color) Range: -1...2 ²⁴ - 1 Default: 16777215 (white)
mapinfo_brush_transparent	Controls whether or not the brush's background is transparent. Range: true false Default: true if no brush background was specified or if set to -1; false otherwise
mapinfo_pen_color	The color of the boundary of the region.

Attribute Name	Contents
	Range: 0...2 ²⁴ - 1 Default: 0 (black)
mapinfo_pen_pattern	<p>The pattern used to draw the region's boundary. See the <i>MapInfo Reference Manual</i> for a list of the available patterns.</p> Range: 1...77 Default: 2
mapinfo_pen_width	<p>The width of the line rendered for the region's boundary. This is measured as a thickness in pixels. A width of 1 is always drawn as a hairline. A width of 0 should be considered to be a line with no width, or a line with no style, or invisible, and should not normally be used. If an <i>invisible</i> line is necessary, it should be created by setting the pattern to 1 (None). If a hairline is desired, the pen should be created by setting the width to 1.</p> Range: 0...35 Default: 1
mapinfo_centroid_x	<p>The centroid x coordinate.</p> Range: Any real number Default: 0
mapinfo_centroid_y	<p>The centroid y coordinate.</p> Range: Any real number Default: 0

Text

mapinfo_type: mapinfo_text

MapInfo text features are used to specify annotation information. Each text feature can have its font, color, spacing, justification, and rotation angle set independently. The following table lists the special FME attribute names used to control the MapInfo text settings.

Attribute Name	Contents
mapinfo_rotation	<p>The rotation of the text, as measured in degrees counterclockwise from horizontal.</p> Range: -360.0..360.0 Default: 0
mapinfo_text_fontbgcolor	<p>The background color used when the text is drawn.</p> Range: 0...2 ²⁴ - 1 Default: 16777215 (white)

Attribute Name	Contents
mapinfo_text_fontfgcolor	<p>The foreground color used when the text is drawn. MapInfo colors are defined in relative concentrations of red, green, and blue. Each color ranges from 0 to 255, and the color value is calculated according to the formula: $(\text{red} * 65536) + (\text{green} * 256) + \text{blue}$ Range: $0 \dots 2^{24} - 1$ Default: 0 (black)</p>
mapinfo_text_fontname	<p>The name of the font used to draw the text. The font named must be available on the destination computer system. Range: Any valid system font Default: Helve</p>
mapinfo_text_height	<p>The height of the text in ground units. Range: Any real number ≥ 0 Default: 10</p>
mapinfo_text_justification	<p>The justification of the text. Range: left center right Default: left</p>
mapinfo_text_spacing	<p>The spacing between lines of multiline text. The measure is expressed as a multiple of the text height. Range: 1.0 1.5 2.0 Default: 1.0</p>
mapinfo_text_linetype	<p>The type of line attaching the text to the anchor point. Range: 0 (None: do not display a line with the label.) 1 (Simple: create a callout by using a simple line that connects the label to the anchor point.) 2 (Arrow: create a callout by using an arrow and line that connects the label to anchor point.) Default: 0 (None)</p>
mapinfo_text_line_end_x	<p>The x position of the label line end point. The linetype needs to be 1 or 2 for the label line to be visible. Range: Any real number</p>

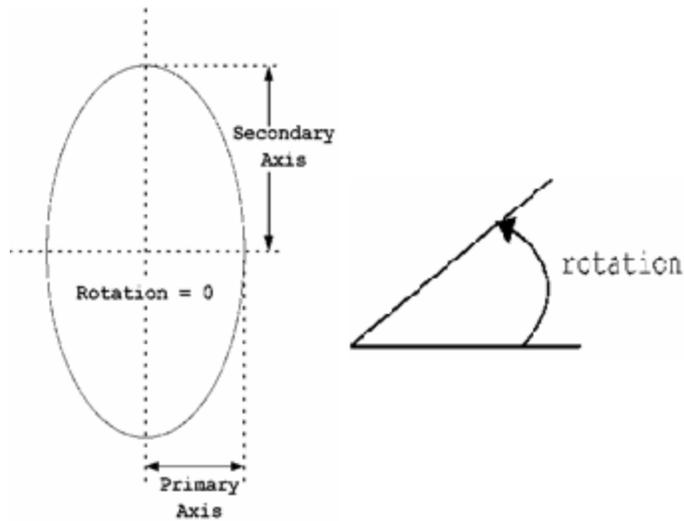
Attribute Name	Contents
	Default: No default
mapinfo_text_line_end_y	The y position of the label line end point. The linetype needs to be 1 or 2 for the label line to be visible. Range: Any real number Default: No default
mapinfo_text_line_pen_color	Stores pen color for text label Range: 0...2 ²⁴ - 1 Default: 0 (black)
mapinfo_text_line_pen_width	Stores pen width for text label Range: 0...7 (pixel width) 11...2047 (point width) Default: 1
mapinfo_text_line_pen_pattern	Stores pen pattern for text label Range: 1...77 Default: 2
mapinfo_text_fontstyle_bold	Indicates if the text is bold or not. Range: true false Default: false
mapinfo_text_fontstyle_italic	Indicates if the text is in Italics Range: true false Default: false
mapinfo_text_fontstyle_underline	Indicates if the text is underlined. Range: true false Default: false
mapinfo_text_fontstyle_strikeout	Indicates if the text has a line through the middle of it. Range: true false Default: false
mapinfo_text_fontstyle_outline	Indicates if the text is outlined Range: true false Default: false
mapinfo_text_fontstyle_shadow	Indicates if the text has a shadow. Range: true false Default: false
mapinfo_text_fontstyle_inverse	Indicates if the text is shown in inverse. Range: true false

Attribute Name	Contents
	Default: false
mapinfo_text_fontstyle_blink	Indicates if the text is blinking. Range: true false Default: false
mapinfo_text_fontstyle_opaque	Indicates if the text is opaque. Range: true false Default: false
mapinfo_text_fontstyle_halo	Indicates if the text has a halo. Range: true false Default: false
mapinfo_text_fontstyle_allcaps	Indicates if the text is uppercase. Range: true false Default: false
mapinfo_text_fontstyle_expanded	Indicates if the text is expanded. Range: true false Default: false
mapinfo_text_string	The text to be displayed. Range: Any character string Default: No default
mapinfo_text_width	The width of the entire text string, in ground units. Range: Any real number ≥ 0 Default: 10

Ellipse

mif_type: mif_ellipse

MapInfo ellipse features are point features, and only have a single coordinate. This point serves as the centre of the ellipse. Additional attributes specify the primary axis and the secondary axis of the ellipse. MapInfo ellipses currently do not support rotation. For compatibility with other systems, the MapInfo reader always returns a rotation of 0. If a rotation is specified to the writer, the ellipse is turned into a region, vectorized, and rotated by the amount specified.



*Tip: The primary ellipse axis is **not** necessarily the longest axis, but rather the one on the x axis.*

In addition to the attributes below, ellipses also make use of the brush and pen attributes as defined by [mapinfo_region](#).

Attribute Name	Contents
mapinfo_primary_axis	The length of the semi-major axis in ground units. Range: Any real number > 0 Default: No default
mapinfo_secondary_axis	The length of the semi-minor axis in ground units. Range: Any real number > 0 Default: No default
mapinfo_rotation	The rotation of the major axis. The rotation is measured in degrees counterclockwise up from horizontal. Range: -360.0..360.0 Default: 0

Arc

mapinfo_type: mapinfo_arc

MapInfo arc features are linear features used to specify elliptical arcs. As such, the feature definition for mapinfo_arc is similar to the ellipse definition with two additional angles to control the portion of the ellipse boundary drawn. MapInfo arcs currently do not support rotation. For compatibility with other systems, the MapInfo reader always returns a rotation of 0. In addition, if a rotation is specified to the writer, the arc is turned into a polyline, vectorized, and rotated by the amount specified.

Tip: The function @Arc() can be used to convert an arc to a linestring. This is useful for storing Arcs in systems not supporting them directly.

In addition the attributes below, arcs also make use of the pen attributes as defined on [mapinfo_polyline](#).

Attribute Name	Contents
mapinfo_primary_axis	The length of the semi-major axis in ground units. Range: Any real number > 0 Default: No default
mapinfo_secondary_axis	The length of the semi-minor axis in ground units. Range: Any real number > 0 Default: No default
mapinfo_start_angle	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of start_angle. Range: 0.0..360.0 Default: 0
mapinfo_sweep_angle	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of sweep_angle. Range: 0.0..360.0 Default: No default
mapinfo_rotation	The rotation of the major axis. The rotation is measured in degrees counter clockwise up from horizontal. Range: -360.0..360.0 Default: 0

Rectangle

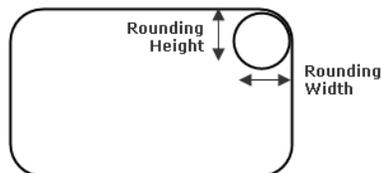
mapinfo_type: mapinfo_rectangle

MapInfo rectangle objects are represented in the FME as closed polygons. When a MapInfo rectangle is read, it is turned into a closed polygon feature. When a MapInfo rectangle is written, the minimum bounding rectangle of the feature is taken and used as the four corners of the rectangle. MapInfo rectangles take the same additional attributes as MapInfo regions to specify their brush and pen.

Rounded Rectangle

mapinfo_type: mapinfo_rounded_rectangle

MapInfo rounded rectangle objects are represented in the FME as closed polygons. When a MapInfo rounded rectangle is read, it is turned into a closed polygon feature and the corners are vectorized to preserve the intended shape of the rectangle. The rounding radius is also stored as an attribute. When a MapInfo rounded rectangle is written, the minimum bounding rectangle of the feature is taken and used as the four corners of the rectangle, and the rounding diameter is taken from an attribute of the feature. MapInfo rounded rectangles take the same additional attributes as MapInfo regions to specify their brush and pen.



Attribute Name	Contents
mapinfo_rounding_width	Contains the width, in ground units, of the ellipse used to produce the rounded corners. Range: Any real number > 0 Default: No default
mapinfo_rounding_height	Contains the height, in ground units, of the ellipse used to produce the rounded corners. Range: Any real number > 0 Default: Value of <code>mapinfo_rounding_width</code>

Collections

mapinfo_type: mapinfo_collection

MapInfo collections are defined as a combination of the other MapInfo geometry types. This is represented as non-homogeneous aggregates composed of the other geometry types.

To create MapInfo collections using FME, set the `mapinfo_type` attribute to `mapinfo_collection` on the feature destined for the MapInfo dataset. It is important that the feature to be saved as a collection is an aggregate feature.

The table below lists the special FME attribute names used to control the MapInfo collection settings:

Attribute Name	Contents
mapinfo_collection_ cmp{}	This is the list attribute prefix used to store the attributes for each collection part. The suffixes are the attribute names for the control settings of the other geometric types. Range: none Default: none
Deprecated	

Microsoft Access Reader/Writer

Format Notes: This format is not supported by FME Base Edition.

Overview

The Microsoft® Access reader and writer provide FME with access to attribute data held in live MS Access database tables. This data may not necessarily have a spatial component to it. FME provides read and write access to live MS Access databases via Microsoft's ActiveX Data Objects (ADO).

Note: Only the standard SQL wildcard characters (% and _) are supported for SQL LIKE queries. Microsoft Access wildcard characters (*, ?, and #) are not supported.

See the @SQL function in the *FME Functions and Factories* manual. This function allows arbitrary Structured Query Language (SQL) statements to be executed against any database.

MS Access Database Quick Facts

Format Type Identifier	MDB_ADO
Reader/Writer	Both
Licensing Level	Professional
Dependencies	<ul style="list-style-type: none">File versions prior to 2007: None, but the format is available only on Windows.File versions 2007 or newer: install a corresponding or newer version of Microsoft Office, or the free download of Microsoft Access Database Engine 2010 Redistributable.
Dataset Type	Database
Feature Type	Table name
Typical File Extensions	N/A
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	Yes
Encoding Support	Yes
Geometry Type	db_none

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	point	no
circles	no	polygon	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
circular arc	no		raster	no
donut polygon	no		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	no		z values	n/a
none	yes			

Reader Overview

The FME considers a database data set to be a collection of relational tables. The tables must be defined in the mapping file before they can be read. Arbitrary where clauses and joins are fully supported.

Reader Directives

The suffixes listed are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the MS Access reader is `MDB_ADO`.

DATASET

Required/Optional: *Required*

This is the file name of the Microsoft Access Database.

Example:

```
MDB_ADO_DATASET c:/data/citySource.mdb
```

Workbench Parameter: *Source Microsoft Access Database File(s)*

PROVIDER_TYPE

Required/Optional: *Optional*

The type of database provider being used. This directive is internal to FME and should always be set to `MDB_ADO`. For example,

```
MDB_ADO_PROVIDER_TYPE MDB_ADO
```

PASSWORD

Required/Optional: *Optional*

The password used to access the database. It can be omitted for Access databases without password protection.

Please note that databases associated with a Microsoft Access workgroup are not supported.

Example:

```
MDB_ADO_PASSWORD moneypenny
```

Workbench Parameter: *Password*

DEF

Required/Optional: *Required*

The syntax of the definition is:

```

MDB_ADO_DEF <tableName> \
    [mdb_where_clause<whereClause>] \
    [<fieldName><fieldType>] +

```

or

```

MDB_ADO_DEF <queryName> \
    [mdb_sql_statement <sqlQuery>] \

```

The **<tableName>** must match the name of an existing MS Access table in the database. This will be used as the feature type of all the features read from the table. The exception to this rule is when using the `mdb_sql_statement` directive. In this case, the **DEF** name may be any valid alphabetic identifier; it does not have to be an existing table name – rather, it is an identifier for the custom SQL query. The feature type of all the features returned from the SQL query are given the query name.

The **<fieldType>** of each field must be given, but it is not verified against the database definition for the field. In effect, it is ignored.

The definition allows specification of separate search parameters for each table. If any of the per table configuration parameters are given, they will override, for that table, whatever global values have been specified by the reader directives such as the **WHERE_CLAUSE**. If any of these parameters is not specified, the global values will be used.

The following table summarizes the definition line configuration parameters:

Parameter	Contents
mdb_where_clause	This specifies the SQL WHERE clause applied to the attributes of the layer's features to limit the set of features returned. If this is not specified, then all the rows are returned. This directive will be ignored if the <code>mdb_sql_statement</code> is present.
mdb_sql_statement	This specifies an SQL SELECT query to be used as the source for the results. If this is specified, the MS Access reader will execute the query, and use the resulting rows as the features instead of reading from the table <code><queryName></code> . All returned features will have a feature type of <code><queryName></code> , and attributes for all columns selected by the query. The <code>mdb_where_clause</code> is ignored if <code>mdb_sql_statement</code> is supplied. This form allows the results of complex joins to be returned to FME.

If no **<whereClause>** is specified, all rows in the table will be read and returned as individual features. If a **<whereClause>** is specified, only those rows that are selected by the clause will be read. Note that the **<whereClause>** does not include the word **WHERE**.

The MS Access reader allows one to use the `mdb_sql_statement` parameter to specify an arbitrary SQL **SELECT** query on the DEF line. If this is specified, FME will execute the query, and use each row of data returned from the query to define at least one feature. Each of these features will be given the feature type named in the DEF line, and will contain attributes for every column returned by the **SELECT**. In this case, all DEF line parameters regarding a **WHERE** clause or spatial querying are ignored, as it is possible to embed this information directly in the text of the **<sqlQuery>**.

In the following example, the all records whose ID is less than 5 will be read from the supplier table:

```

MDB_ADO_DEF supplier \
    mdb_where_clause "id < 5" \

```

```
ID integer \
NAME char(100) \
CITY char(50)
```

In this example, the results of joining the **employee** and **city** tables are returned. All attributes from the two tables will be present on each returned feature. The feature type will be set to **complex**.

```
MDB_ADO_DEF complex \
  mdb_sql_statement \
    "SELECT * FROM EMPLOYEE, CITY WHERE EMPLOYEE.CITY = CITY.NAME"
```

WHERE_CLAUSE

Required/Optional: *Optional*

This optional specification is used to limit the rows read by the reader from each table. If a given table has no `mdb_where_clause` or `mdb_sql_statement` specified in its DEF line, the global `<ReaderKeyword>_WHERE_CLAUSE` value, if present, will be applied as the **WHERE** specifier of the query used to generate the results. If a table's DEF line does contain its own `mdb_where_clause` or `mdb_sql_statement`, it will override the global **WHERE** clause.

The syntax for this clause is:

```
MDB_ADO_WHERE_CLAUSE <whereClause>
```

Note that the `<whereClause>` does not include the word "WHERE."

The example below selects only the features whose lengths are more than 2000:

```
MDB_ADO_WHERE_CLAUSE LENGTH > 2000
```

Workbench Parameter: *Where Clause*

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined database tables that will be read. If no **IDs** are specified, then all tables are read. The syntax of the **IDs** directive is:

```
MDB_ADO_IDS <featureType1> \
  <featureType2> ... \
  <featureTypeN>
```

The feature types must match those used in DEF lines.

The example below selects only the **HISTORY** table for input during a translation:

```
MDB_ADO_IDS HISTORY
```

Workbench Parameter: *Feature Types to Read*

READ_CACHE_SIZE

Required/Optional: *Optional*

This directive controls how the reader retrieves rows from the database. This must be a numeric value which must be greater than 0.

The `READ_CACHE_SIZE` is used to determine the number of rows that are retrieved at one time into local memory from the data source. For example, if the `READ_CACHE_SIZE` is set to 10, after the reader is opened, the reader will read 10 rows into local memory. As features are processed by the FME, the reader returns the data from the local memory buffer. As soon as you move past the last row available in local memory, the reader will retrieve the next 10 rows from the data source.

This directive affects the performance of the reader, and will result in significantly degraded performance if incorrectly set. The optimum value of this directive depends primarily on the characteristics of individual records and the transport between the database and the client machine. It is less affected by the quantity of rows that are to be retrieved.

By default, the READ_CACHE_SIZE is set to 10. This value has been determined to be the optimal value for average datasets.

Workbench Parameter: *Number of Records to Fetch At A Time*

RETRIEVE_ALL_SCHEMAS

Required/Optional: *Optional*

This directive is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

When set to "Yes", indicates to the reader to return all the schemas of the tables in the database.

If this directive is missing, it is assumed to be "No".

Range: YES | NO

Default: NO

RETRIEVE_ALL_TABLE_NAMES

Required/Optional: *Optional*

This specification is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

Similar to **RETRIEVE_ALL_SCHEMAS**; this optional directive is used to tell the reader to only retrieve the table names of all the tables in the source database. If **RETRIEVE_ALL_SCHEMAS** is also set to "Yes", then **RETRIEVE_ALL_SCHEMAS** will take precedence. If this directive is not specified, it is assumed to be "No".

Range: YES | NO

Default: NO

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

*** Workbench Parameter**

Additional Attributes to Expose

Writer Overview

The MS Access writer module stores attribute records into a live relational database. The MS Access writer provides the following capabilities:

- **Transaction Support:** The MS Access writer provides transaction support that eases the data loading process. Occasionally, a data load operation terminates prematurely due to data difficulties. The transaction support provides a mechanism for reloading corrected data without data loss or duplication.
- **Table Creation:** The MS Access writer uses the information within the FME mapping file to automatically create database tables as needed.
- **Writer Mode Specification:** The MS Access writer allows the user to specify what database command should be issued for each feature received. Valid writer modes are INSERT, UPDATE and DELETE. The writer mode can be specified at three unique levels: at the writer level, on the feature type, or on individual features.

Writer Directives

The directives processed by the MS Access Writer are listed below. The suffixes shown are prefixed by the current `<writerKeyword>` in a mapping file. By default, the `<writerKeyword>` for the MS Access writer is `MDB_ADO`.

DATASET, PROVIDER_TYPE

The `DATASET` and `PROVIDER_TYPE` directives operate in the same manner as they do for the MS Access reader. The remaining writer-specific directives are discussed in the following sections.

Workbench Parameter: *Destination Microsoft Access Database File*

PASSWORD

Required/Optional: *Optional*

The password used to access the database. For existing databases, it may be omitted for Access databases without password protection. If the database does not exist, then the newly created Microsoft Access database will be protected by this password.

Please note that databases associated with a Microsoft Access workgroup are not supported.

```
MDB_ADO_PASSWORD moneypenny
```

Workbench Parameter: *Password*

DEF

Required/Optional: *Required*

Each MS Access table must be defined before it can be written. The general form of a MS Access definition statement is:

```
MDB_ADO_DEF <tableName> \
    [mdb_update_key_columns <keyColumns>]\
    [mdb_drop_table (yes|no)]\
    [mdb_truncate_table (yes|no)] \
    [mdb_table_writer_mode (inherit_from_writer|insert|
        update|delete)] \
    [<fieldName><fieldType>[,<indexType>]]+
```

The table definition allows control of the table that will be created. If the fields and types are listed, the types must match those in the database. Fields which can contain NULL values do not need to be listed - these fields will be filled with NULL values.

If the table does not exist, then the field names and types are used to first create the table. In any case, if a `<fieldType>` is given, it may be any field type supported by the target database.

The configuration parameters present on the definition line are described in the following table:

Parameter	Contents
tableName	The name of the table to be written. If a table with the specified name exists, it will be overwritten if the mdb_

Parameter	Contents
	<p>drop_table DEF line parameter is set to YES, or it will be truncated if the mdb_truncate_table DEF line parameter is set to YES. Otherwise the table will be appended. Valid values for table names include any character string devoid of SQL-offensive characters and less than 128 characters in length.</p>
mdb_table_writer_mode	<p>The the default operation mode of the feature type in terms of the types of SQL statements sent to the data-base. Valid values are INSERT, UPDATE, DELETE and INHERIT_FROM_WRITER. Note that INSERT mode allows for only INSERT operations where as UPDATE and DELETE can be overwritten at the feature levels. INHERIT_FROM_WRITER simply indicates to take this value from the writer level and not to override it at the feature type level.</p> <p>Default: INHERIT_FROM_WRITER</p>
mdb_update_key_columns	<p>This is a comma-separated list of the columns which are matched against the corresponding FME attributes' values to specify which rows are to be updated or deleted when the writer mode is either UPDATE or INSERT.</p> <p>For example: mdb_update_key_columns ID would instruct the writer to ensure that the FME attribute is always matched against the column with the same name. Also, the target table is always the feature type specified in the DEF line.</p> <p>Each column listed with the mdb_update_key_columns directive must be defined with a type on the DEF line, in addition to the columns whose values will be updated by the operation.</p>
mdb_drop_table	<p>This specifies that if the table exists by this name, it should be dropped and replaced with a table specified by this definition.</p> <p>Default: NO</p>
mdb_truncate_table	<p>This specifies that if the table exists by this name, it should be cleared prior to writing.</p> <p>Default: NO</p>
fieldName	<p>The name of the field to be written. Valid values for field name include any character string devoid of SQL-offensive characters and less than 128 characters in length.</p>

Parameter	Contents
fieldType	<p>The type of a column in a table. The valid values for the field type are listed below:</p> <ul style="list-style-type: none"> yesno memo hyperlink replicationid oleobject integer byte long autonumber datetime decimal(width,decimal) single double currency text(width)
indexType	<p>The type of index to create for the column.</p> <p>If the table does not previously exist, then upon table creation, a database index of the specified type is created. The database index contains only the one column.</p> <p>The valid values for the column type are listed below:</p> <ul style="list-style-type: none"> indexed: An index without constraints. unique: An index with a unique constraint.

VERSION

Required/Optional: *Required*

This statement tells the MS Access writer what version of database should be created. If the database file already exists, the writer will automatically detect and use the correct version.

Parameter	Contents
<version>	<p>The version of Microsoft Access database file to create. The valid values are listed below:</p> <ul style="list-style-type: none"> 2000/2002/2003 95/97 2.0 <p>Default: 2000/2002/2003</p>

Example:

`MDB_ADO_VERSION 2000/2002/2003`

Workbench Parameter: *Version*

START_TRANSACTION

Required/Optional: *Optional*

This statement tells the MS Access writer module when to start actually writing features into the database. The MS Access writer does not write any features until the feature is reached that belongs to **<last successful transaction> + 1**. Specifying a value of zero causes every feature to be output. Normally, the value specified is zero – a non-zero value is only specified when a data load operation is being resumed after failing partway through.

Parameter	Contents
<last successful transaction>	The transaction number of the last successful transaction. When loading data for the first time, set this value to 0 . Default: 0

Example:

```
MDB_ADO_START_TRANSACTION 0
```

Workbench Parameter: *Start transaction at*

TRANSACTION_INTERVAL

Required/Optional: *Optional*

This statement informs the FME about the number of features to be placed in each transaction before a transaction is committed to the database.

If the **MDB_ADO_TRANSACTION_INTERVAL** statement is not specified, then a value of 500 is used as the transaction interval.

Parameter	Contents
<transaction_interval>	The number of features in a single transaction. Default: 500

If the **MDB_ADO_TRANSACTION_INTERVAL** is set to zero, then feature based transactions are used. As each feature is processed by the writer, they are checked for an attribute called **fme_db_transaction**. The value of this attribute specifies whether the writer should commit or rollback the current transaction. The value of the attribute can be one of **COMMIT_BEFORE**, **COMMIT_AFTER**, **ROLLBACK_AFTER** or **IGNORE**. If the **fme_db_transaction** attribute is not set in any features, then the entire write operation occurs in a single transaction.

Example:

```
MDB_ADO_TRANSACTION_INTERVAL 5000
```

Workbench Parameter: *Transaction interval*

WRITER_MODE

Required/Optional: *Optional*

Note: For more information, see the chapter *Database Writer Mode* on page 19.

This directive informs the MS Access writer which SQL operations will be performed by default by this writer. This operation can be set to **INSERT**, **UPDATE** or **DELETE**. The default writer level value for this operation can be overwritten at the feature type or table level. The corresponding feature type DEF parameter name is called **mdb_table_**

writer_mode. It has the same valid options as the writer level mode and additionally the value **INHERIT_FROM_WRITER** which causes the writer level mode to be inherited by the feature type as the default for features contained in that table.

The operation can be set specifically for individual feature as well. Note that when the writer mode is set to INSERT this prevents the mode from being interpreted from individual features and all features are inserted unless otherwise marked as UPDATE or DELETE features. These are skipped.

If the MDB_ADO_WRITER_MODE statement is not specified, then a value of INSERT is given.

Parameter	Contents
<writer_mode>	The type of SQL operation that should be performed by the writer. The valid list of values are below: INSERT UPDATE DELETE Default: INSERT

Example:

```
MDB_ADO_WRITER_MODE INSERT
```

Workbench Parameter: *Writer Mode*

BEGIN_SQL{n}

Occasionally you must execute some ad-hoc SQL prior to opening a table. For example, it may be necessary to ensure that a view exists prior to attempting to read from it.

Upon opening a connection to read from a database, the reader looks for the directive **<ReaderKeyword>_BEGIN_SQL{n}** (for n=0,1,2,...), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the **FME_SQL_DELIMITER** keyword, embedded at the beginning of the SQL block. The single character following this keyword will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;
DELETE FROM instructors;
DELETE FROM people
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

*** Workbench Parameter**

SQL Statement to Execute Before Translation

END_SQL{n}

Occasionally you must execute some ad-hoc SQL after closing a set of tables. For example, it may be necessary to clean up a temporary view after writing to the database.

Just before closing a connection on a database, the reader looks for the directive `<ReaderKeyword>_END_SQL{n}` (for $n=0,1,2,\dots$), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the `FME_SQL_DELIMITER` directive, embedded at the beginning of the SQL block. The single character following this directive will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;
DELETE FROM instructors;
DELETE FROM people
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

* Workbench Parameter

SQL Statement to Execute After Translation

INIT_TABLES

Required/Optional: *Optional*

This directive informs the MS Access writer when each table should be initialized. Initialization encompasses the actions of dropping or truncating existing tables, and creating new tables as necessary.

When `INIT_TABLES` is set to `IMMEDIATELY`, the MS Access writer will initialize all tables immediately after parsing the `DEF` lines and opening the database file. In this mode, all tables will be initialized, even if the MS Access writer receives no features for a given table.

When `INIT_TABLES` is set to `FIRSTFEATURE`, the MS Access writer will only initialize a table once the first feature destined for that table is received. In this mode, if the MS Access writer does not receive any features for a given table, the table will never be initialized.

Workbench Parameter: *Initialize Tables*

Writer Mode Specification

The MS Access writer allows the user to specify a writer mode, which determines what database command should be issued for each feature received. Valid writer modes are `INSERT`, `UPDATE` and `DELETE`.

Writer Modes

In `INSERT` mode, the attribute values of each received feature are written as a new database record.

In `UPDATE` mode, the attribute values of each received feature are used to update existing records in the database. The records which are updated are determined via the `mdb_update_key_columns DEF` line parameter, or via the `fme_where` attribute on the feature.

In **DELETE** mode, existing database records are deleted according to the information specified in the received feature. Records are selected for deletion using the same technique as records are selected for updating in **UPDATE** mode.

Writer Mode Constraints

In **UPDATE** and **DELETE** mode, the **fme_where** attribute always takes precedence over the **mdb_update_key_columns DEF** line parameter. If both the **fme_where** attribute and the **mdb_update_key_columns DEF** line parameter are not present, then **UPDATE** or **DELETE** mode will generate an error.

When the **fme_where** attribute is present, it is used verbatim as the WHERE clause on the generated **UPDATE** or **DELETE** command. For example, if **fme_where** were set to `'id<5'`, then all database records with field ID less than 5 will be affected by the command.

When the **fme_where** attribute is not present, the writer looks for the **mdb_update_key_columns DEF** line parameter and uses it to determine which records should be affected by the command. Please refer to See "DEF" for more information about the **mdb_update_key_columns DEF** line parameter.

Writer Mode Selection

The writer mode can be specified at three unique levels. It may be specified on the writer level, on the feature type or on individual features.

At the writer level, the writer mode is specified by the **WRITER_MODE** directive. This directive can be superseded by the feature type writer mode specification. For more information on this directive, see the chapter Database Writer Mode.

At the feature type level, the writer mode is specified by the **mdb_writer_mode DEF** line parameter. This parameter supersedes the **WRITER_MODE** directive. Unless this parameter is set to **INSERT**, it may be superseded on individual features by the **fme_db_operation** attribute. Please refer to the DEF line documentation for more information about this parameter.

At the feature level, the writer mode is specified by the **fme_db_operation** attribute. Unless the parameter at the feature type level is set to **INSERT**, the writer mode specified by this attribute always supersedes all other values. Accepted values for the **fme_db_operation** attribute are **INSERT**, **UPDATE** or **DELETE**.

Feature Representation

Features read from a database consist of a series of attribute values. They have no geometry. The attribute names are as defined in the **DEF** line if the first form of the **DEF** line was used. If the second form of the **DEF** line was used, then the attribute names are as they are returned by the query, and as such may have their original table names as qualifiers. The feature type of each MS Access feature is as defined on its **DEF** line.

Features written to the database have the destination table as their feature type, and attributes as defined on the **DEF** line.

Microsoft DirectX Writer

Note: This format is not available in FME Base Edition.

Overview

The DirectX Writer allows FME write Microsoft DirectX® Format (.x) files.

The DirectX format was originally developed as an open interchange format for DirectX 2.0 and was supported until Direct3D 9.0. It is now commonly used as an interchange format between different 3D modelling and rendering applications.

DirectX Quick Facts

Format Type Identifier	DIRECTX
Reader/Writer	Writer
Licensing Level	Professional
Dependencies	None
Dataset Type	Writer: Directory
Feature Type	DIRECTX_MODEL
Typical File Extensions	.x
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	No
Appearance Support (Surfaces Only)	Yes
Spatial Index	Never
Schema Required	Yes
Transaction Support	Never
Enhanced Geometry	Yes
Geometry Type Attribute	directx_type

Geometry Support

Geometry	Supported?	Geometry Supported?	
aggregate	yes	point	no
circles	yes	polygon	yes
circular arc	no	raster	no
donut polygon	yes	Solid	yes
elliptical arc	no	surface	yes
ellipses	yes	Text	no

line	no	z values	yes
none	no		

Writer Overview

The 3D model has a hierarchal structure of Nodes, which are elements of the model. For each node, there is a corresponding mesh, which contains the geometry of the object. Feature types become Nodes. Features become Meshes that may have geometries and attributes.

The DirectX writer produces an '.x' file for each FME feature type sent to the writer. Surface, solid, and polygon geometries are converted into DirectX triangle mesh structures. Texture files are written to the same directory as the models and referenced by DirectX models using the texture name. Any old files in the output directory are overwritten by new files with the same name. If the output files cannot be written, the translation fails.

The DirectX Writer supports feature type fanout and will write a different DirectX model file for each feature type.

- The DirectX format is limited to 32-bit precision for its coordinates and, as a result, translations involving a greater level of precision (i.e., using world coordinates instead of local coordinates) may produce DirectX data where different coordinates are collapsed into a single coordinate. This issue can be resolved by offsetting the x,y,z coordinates such that the model's origin is moved to (0,0,0) or another point close to this, which has the effect of moving the model into a local coordinate system.

Writer Directives

The directives that are processed by the DirectX writer are listed below. The suffixes shown are prefixed by the current <WriterKeyword>_ in a mapping file. By default, the <WriterKeyword> for the DirectX writer is DIRECTX.

DATASET

The value for this directive is the path to the output directory. If the output directory does not exist, the writer will create a new one.

The output file will be created within the specified directory. All associated texture files, if any, will be written to the same directory.

For example, if the output directory is C:\DirectXFiles\house\ and the Feature Type Name is "shed" the output file will be C:\DirectXFiles\house\shed.x.

If an output file exists, the writer will overwrite it. If other applications have an output file opened, the writer will be unable to continue and the translation will fail.

Required/Optional

Required

* Workbench Parameter

Destination DirectX Directory

FILE_TYPE

Values

Text (default) | Binary | Compressed Binary.

- Text: produces a human-readable text file that can be examined using a text editor.
- Binary: produces a binary version of the file in the legacy binary format, which are generally smaller than the text version.
- Compressed Binary: produces a compressed version of the file.

Required/Optional

Optional

*** Workbench Parameter**

Output File Type

LHCS_CONVERSION

Required/Optional: Optional

DirectX uses a left-handed coordinate system (LHCS), while FME natively uses a right-handed coordinate system.

During export, FME features will be transformed to convert to a LHCS according to the option selected on the writer.

Values

Flip Z: FME will convert to LHCS by scaling all z-values by negative one (-1).

Swap Y and Z: FME will convert to LHCS by swapping the Y and Z axes.

Required/Optional

Optional

*** Workbench Parameter**

Conversion to LHCS Method

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see About Feature Attributes), this format adds the format-specific attributes described in this section.

DirectX features consist of geometry alone. All DirectX features contain a `directx_type` attribute, which identifies the geometric type.

Geometries with no Z coordinates (2D geometries) will be assigned zero as their z values.

Mesh

`directx_type`: `directx_model`

Models are meshes are composed of triangular faces. If the input mesh contains faces with more than three distinct vertices, then the face will be converted into multiple triangular faces. The triangular faces of a mesh need not be connected.

Polygons and donuts are treated as meshes. They will be converted into triangular faces that represents the inner area of the polygon or donut.

Microsoft Excel Reader/Writer

Format Notes: This format is not supported by FME Base Edition.

Overview

The Microsoft Excel reader and writer modules provide FME with access to attribute data held in MS Excel workbooks. This data may not necessarily have a spatial component to it. FME provides read and write access to MS Excel workbooks via Microsoft's ActiveX Data Objects (ADO).

Tip: See the @SQL function in the Functions and Factories manual. This function allows arbitrary Structured Query Language (SQL) statements to be executed against any database.

MS Excel Quick Facts

Format Type Identifier	XLS_ADO
Reader/Writer	Both
Licensing Level	Professional
Dependencies	<ul style="list-style-type: none">• File versions prior to 2007: None, but the format is available only on Windows.• File versions 2007 or newer: install a corresponding or newer version of Microsoft Office, or the free download of Microsoft Access Database Engine 2010 Redistributable.
Dataset Type	Database
Feature Type	Table name
Typical File Extensions	N/A
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	Yes
Encoding Support	Yes
Geometry Type	db_none

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	point	no
circles	no	polygon	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
circular arc	no		raster	yes
donut polygon	no		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	no		z values	n/a
none	yes			

Reader Overview

FME considers a database data set to be a collection of relational tables. The tables must be defined in the mapping file before they can be read. Arbitrary where clauses and joins are fully supported. In MS Excel, tables can be either worksheets or named ranges. The FME always denotes named ranges by following their names with an asterisk (*) character.

Reader Directives

The suffixes listed are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the MS Excel reader is **XLS_ADO**.

DATASET

Required/Optional: *Required*

This is the file name of the Microsoft Excel workbook.

Example:

```
XLS_ADO_DATASET C:/data/citySource.xls
```

Workbench Parameter: *Source Microsoft Excel File(s)*

PROVIDER_TYPE

Required/Optional: *Optional*

The type of database provider being used. This keyword is internal to FME and should always be set to **XLS_ADO**.

```
XLS_ADO_PROVIDER_TYPE XLS_ADO
```

DEF

Required/Optional: *Required*

The syntax of the definition is:

```
XLS_ADO_DEF <tableName> \
    [xls_where_clause <whereClause>] \
    [<fieldName> <fieldType>] +
```

OR

```
XLS_ADO_DEF <queryName> \
    [xls_sql_statement <sqlQuery>] \
```

The **<tableName>** must match the name of an existing MS Excel worksheet or named range in the workbook. The name does not support non alpha-numeric characters. If the table is a named range, the must be followed with a *. Worksheets do not need to be followed by any characters. This will be used as the feature type of all the features read

from the worksheet. The exception to this rule is when using the `xls_sql_statement` keyword. In this case, the **DEF** name may be any valid alphabetic identifier; it does not have to be an existing worksheet name – rather, it is an identifier for the custom SQL query. The feature type of all the features returned from the SQL query are given the query name.

The `<fieldType>` of each field must be given, but it is not verified against the database definition for the field. In effect, it is ignored.

The definition allows specification of separate search parameters for each worksheet. If any of the per table configuration parameters are given, they will override, for that table, whatever global values have been specified by the reader keywords such as the **WHERE_CLAUSE**. If any of these parameters is not specified, the global values will be used.

The following table summarizes the definition line configuration parameters:

Parameter	Contents
<code>xls_where_clause</code>	This specifies the SQL WHERE clause applied to the attributes of the layer's features to limit the set of features returned. If this is not specified, then all the rows are returned. This keyword will be ignored if the <code>xls_sql_statement</code> is present.
<code>xls_sql_statement</code>	This specifies an SQL SELECT query to be used as the source for the results. If this is specified, the MS Excel reader will execute the query, and use the resulting rows as the features instead of reading from the table <code><queryName></code> . All returned features will have a feature type of <code><queryName></code> , and attributes for all columns selected by the query. The <code>xls_where_clause</code> is ignored if <code>xls_sql_statement</code> is supplied. This form allows the results of complex joins to be returned to FME.

If no `<whereClause>` is specified, all rows in the table will be read and returned as individual features. If a `<whereClause>` is specified, only those rows that are selected by the clause will be read. Note that the `<whereClause>` does not include the word **WHERE**.

The MS Excel reader allows one to use the `xls_sql_statement` parameter to specify an arbitrary SQL **SELECT** query on the DEF line. If this is specified, FME will execute the query, and use each row of data returned from the query to define at least one feature. Each of these features will be given the feature type named in the **DEF** line, and will contain attributes for every column returned by the **SELECT**. In this case, all **DEF** line parameters regarding a **WHERE** clause or spatial querying are ignored, as it is possible to embed this information directly in the text of the `<sqlQuery>`.

In the following example, the all records whose ID is less than 5 will be read from the Sheet1 worksheet:

```
XLS_ADO_DEF Sheet1$ \
  xls_where_clause "id < 5" \
  ID integer \
  NAME text \
  CITY text
```

In this example, the results of joining the **employee** and **city** tables are returned. All attributes from the two tables will be present on each returned feature. The feature type will be set to **complex**.

```
XLS_ADO_DEF complex \
  xls_sql_statement \
  "SELECT * FROM EMPLOYEE, CITY WHERE EMPLOYEE.CITY = CITY.NAME"
```

WHERE_CLAUSE

Required/Optional: *Optional*

This optional specification is used to limit the rows read by the reader from each table. If a given table has no `xls_where_clause` or `xls_sql_statement` specified in its `DEF` line, the global `<ReaderKeyword>_WHERE_CLAUSE` value, if present, will be applied as the `WHERE` specifier of the query used to generate the results. If a table's `DEF` line does contain its own `xls_where_clause` or `xls_sql_statement`, it will override the global `WHERE` clause.

The syntax for this clause is:

```
XLS_ADO_WHERE_CLAUSE <whereClause>
```

Note that the `<whereClause>` does not include the word "WHERE."

The example below selects only the features whose lengths are more than 2000:

```
XLS_ADO_WHERE_CLAUSE LENGTH > 2000
```

Workbench Parameter: *Where Clause*

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined database tables that will be read. If no `IDs` are specified, then all worksheets are read. Also, if no `IDs` are specified and `SHOW_NAMED_RANGES` is set to `yes` all named ranges are also read. The syntax of the `IDs` keyword is:

```
XLS_ADO_IDS <featureType1> \
  <featureType2> ... \
  <featureTypeN>
```

The feature types must match those used in `DEF` lines.

Note that feature type names for named ranges must be followed by an asterisk (*) character.

The example below selects only the `HISTORY` worksheet for input during a translation:

```
XLS_ADO_IDS HISTORY
```

Workbench Parameter: *Feature Types to Read*

READ_CACHE_SIZE

Required/Optional: *Optional*

This keyword controls how the reader retrieves rows from the database. This must be a numeric value which must be greater than 0.

The `READ_CACHE_SIZE` is used to determine the number of rows that are retrieved at one time into local memory from the data source. For example, if the `READ_CACHE_SIZE` is set to 10, after the reader is opened, the reader will read 10 rows into local memory. As features are processed by the FME, the reader returns the data from the local memory buffer. As soon as you move past the last row available in local memory, the reader will retrieve the next 10 rows from the data source.

This keyword affects the performance of the reader, and will result in significantly degraded performance if incorrectly set. The optimum value of this keyword depends primarily on the characteristics of individual records and the transport between the database and the client machine. It is less affected by the quantity of rows that are to be retrieved.

By default, the READ_CACHE_SIZE is set to 10. This value has been determined to be the optimal value for average datasets.

Workbench Parameter: *Number of Features To Fetch At A Time*

SHOW_NAMED_RANGES

Required/Optional: *Optional*

This keyword controls whether the reader will interpret named ranges as being valid feature types.

If SHOW_NAMED_RANGES is set to yes, then every named range will appear as a unique feature type. The names of these feature types will be the name of the named range, followed by an asterisk (*) character.

If SHOW_NAMED_RANGES is set to no, then named ranges will not appear as feature types.

Each worksheet always appears as a unique feature types.

FIRST_ROW_IS_HEADING

Required/Optional: *Optional*

This directive controls whether the reader will interpret the first row of each column as field names or as data.

If FIRST_ROW_IS_HEADING is set to yes, then the first row of every table will be used as field names.

If FIRST_ROW_IS_HEADING is set to no then the first row of every table will be used as data. Field names will be automatically generated, using the name F1 for the first column, F2 for the second column, and so on.

RETRIEVE_ALL_SCHEMAS

Required/Optional: *Optional*

This specification is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

When set to YES, tells the reader to return all the schemas of the tables in the database.

If this specification is missing, it is assumed to be NO.

Range: YES | NO

Default: NO

RETRIEVE_ALL_TABLE_NAMES

Required/Optional: *Optional*

This specification is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

This directive is similar to RETRIEVE_ALL_SCHEMAS except when set to YES.

When set to YES, this directive tells the reader to only return the names of all of the tables in the database. However, if RETRIEVE_ALL_SCHEMAS is also set to YES, then RETRIEVE_ALL_SCHEMAS takes precedence.

If this specification is missing, it is assumed to be NO.

Range: YES | NO

Default: NO

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the `mitab.dll` in the FME home directory to `mapinfo.dll`.

The syntax of the `MAPINFO_SEARCH_ENVELOPE` directive is:

`MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>`

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

`MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1`

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The `COORDINATE_SYSTEM` directive, which specifies the coordinate system associated with the data to be read, must always be set if the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` to the reader `COORDINATE_SYSTEM` prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

`<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>`

*** Workbench Parameter**

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the `SEARCH_ENVELOPE` directive.

Values

YES | NO (default)

Mapping File Syntax

`<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]`

*** Workbench Parameter**

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The MS Excel writer module stores attribute records into a live relational database. The MS Excel writer provides the following capabilities:

- **Table Creation:** The MS Excel writer uses the information within the FME mapping file to automatically create database tables as needed.
- **Writer Mode Specification:** The MS Excel allows the user to specify what database command should be issued for each feature received. Valid writer modes are INSERT and UPDATE. The writer mode can be specified at three unique levels. It may be specified on the writer level, on the feature type or on individual features.

Writer Directives

The directives processed by the MS Excel Writer are listed below. The suffixes shown are prefixed by the current `<writerKeyword>` in a mapping file. By default, the `<writerKeyword>` for the MS Excel writer is `XLS_ADO`.

DATASET, PROVIDER_TYPE

These directives operate in the same manner as they do for the MS Excel reader. The remaining writer-specific directives are discussed in the following sections.

Workbench Parameter: *Destination Microsoft Excel File*

DEF

Required/Optional: *Required*

Each MS Excel table must be defined before it can be written. The general form of a MS Excel definition statement is:

```
XLS_ADO_DEF <tableName> \  
    [xls_update_key_columns <keyColumns>] \  
    [xls_table_writer_mode (inherit_from_writer|insert|update)] \  
    [xls_is_named_range (yes|no) \  
    [<fieldName> <fieldType>]+
```

The table definition allows control of the table that will be created. If the fields and types are listed, the types must match those in the database. Fields which can contain NULL values do not need to be listed - these fields will be filled with NULL values.

If the table does not exist, then the field names and types are used to first create the table. In any case, if a `<field-
Type>` is given, it may be any field type supported by the target database.

The configuration parameters present on the definition line are described in the following table:

Parameter	Contents
tableName	The name of the table to be written. If a table with the specified name exists it will be truncated if the <code>xls_truncate_table</code> DEF line parameter is set to YES. Otherwise the table will be appended. Table names may consist of only alphanumeric characters, and must be less than 31 characters in length..

Parameter	Contents
xls_table_writer_mode	<p>The the default operation mode of the feature type in terms of the types of SQL statements sent to the data-base. Valid values are INSERT, UPDATE and INHERIT_FROM_WRITER. Note that INSERT mode allows for only INSERT operations where as UPDATE can be over-written at the feature levels. INHERIT_FROM_WRITER simply indicates to take this value from the writer level and not to override it at the feature type level.</p> <p>Default:INHERIT_FROM_WRITER</p>
xls_update_key_columns	<p>This is a comma-separated list of the columns which are matched against the corresponding FME attributes' values to specify which rows are to be updated when the writer mode is UPDATE.</p> <p>For example: xls_update_key_columns ID would instruct the writer to ensure that the FME attribute is always matched against the column with the same name. Also, the target table is always the feature type specified in the DEF line.</p> <p>Each column listed with the xls_update_key_columns keyword must be defined with a type on the DEF line, in addition to the columns whose values will be updated by the operation.</p>
xls_is_named_range	<p>This controls whether the destination table is a worksheet or a named range.</p> <p>Default: YES</p>
fieldName	<p>The name of the field to be written. Valid values for field name include any character string devoid of SQL-offensive characters and less than 255 characters in length.</p>
fieldType	<p>The type of a column in a table. The valid values for the field type are listed below:</p> <ul style="list-style-type: none"> text varchar(width) datetime decimal(width,decimal) currency(width,decimal) double

VERSION

Required/Optional: *Optional*

This statement tells the MS Excel writer what version of workbook should be created. If the workbook already exists, the writer will automatically detect and use the correct version.

Parameter	Contents
<version>	The version of Microsoft Excel workbook to create. The valid values are listed below: 97/2000/2002/2003 5.0/95 4.0 3.0 Default: 97/2000/2002/2003

Example:

`XLS_ADO_VERSION 2000/2002/2003`

Workbench Parameter: *Excel Version*

START_TRANSACTION

Required/Optional: *Optional*

This statement tells the MS Excel writer module when to start actually writing features into the database. The MS Excel writer does not write any features until <last successful feature> + 1 features have been reached. Specifying a value of zero causes every feature to be output. Normally, the value specified is zero – a non-zero value is only specified when a data load operation is being resumed after failing partway through.

Parameter	Contents
<last successful feature>	The number of the last successful feature. When loading data for the first time, set this value to 0. Default: 0

Example:

`XLS_ADO_START_TRANSACTION 0`

Workbench Parameter: *Start Writing at Feature*

TRANSACTION_INTERVAL

Required/Optional: *Optional*

This statement informs the FME about the number of features to be placed in each transaction before a transaction is committed to the database.

Since MS Excel does not support transactions, this value should always be set to 1.

If the `XLS_ADO_TRANSACTION_INTERVAL` statement is not specified, then a value of 1 is used as the transaction interval.

Parameter	Contents
<transaction_interval>	The number of features in a single transaction. This value must always be set to 1. Default: 1

Example:

```
XLS_ADO_TRANSACTION_INTERVAL 1
```

WRITER_MODE

Required/Optional: *Optional*

Note: For more information on this directive, see the chapter Database Writer Mode.

This keyword informs the MS Excel writer which SQL operations will be performed by default by this writer. This operation can be set to **INSERT** or **UPDATE**. The default writer level value for this operation can be overwritten at the feature type or table level. The corresponding feature type DEF parameter name is called `xls_table_writer_mode`. It has the same valid options as the writer level mode and additionally the value **INHERIT_FROM_WRITER** which causes the writer level mode to be inherited by the feature type as the default for features contained in that table.

The operation can be set specifically for individual feature as well. Note that when the writer mode is set to **INSERT** this prevents the mode from being interpreted from individual features and all features are inserted unless otherwise marked as **UPDATE** features. These are skipped.

If the `XLS_ADO_WRITER_MODE` statement is not specified, then a value of **INSERT** is given.

Parameter	Contents
<writer_mode>	The type of SQL operation that should be performed by the writer. The valid list of values are below: INSERT UPDATE Default: INSERT

Example:

```
XLS_ADO_WRITER_MODE INSERT
```

Workbench Parameter: *Writer Mode*

BEGIN_SQL{n}

Occasionally you must execute some ad-hoc SQL prior to opening a table. For example, it may be necessary to ensure that a view exists prior to attempting to read from it.

Upon opening a connection to read from a database, the reader looks for the directive `<ReaderKeyword>_BEGIN_SQL{n}` (for `n=0,1,2,...`), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the **FME_SQL_DELIMITER** keyword, embedded at the beginning of the SQL block. The single character following this keyword will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;
DELETE FROM instructors;
DELETE FROM people
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

* Workbench Parameter

SQL Statement to Execute Before Translation

END_SQL{n}

Occasionally you must execute some ad-hoc SQL after closing a set of tables. For example, it may be necessary to clean up a temporary view after writing to the database.

Just before closing a connection on a database, the reader looks for the directive `<ReaderKeyword>_END_SQL{n}` (for $n=0,1,2,\dots$), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the `FME_SQL_DELIMITER` directive, embedded at the beginning of the SQL block. The single character following this directive will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;  
DELETE FROM instructors;  
DELETE FROM people  
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

* Workbench Parameter

SQL Statement to Execute After Translation

INIT_TABLES

Required/Optional: *Optional*

This keywords informs the MS Excel writer when each table should be initialized. Initialization encompasses the actions of truncating existing tables and creating new tables as necessary.

When `INIT_TABLES` is set to `IMMEDIATELY`, the MS Excel writer will initialize all tables immediately after parsing the `DEF` lines and opening the Excel workbook. In this mode, all tables will be initialized, even if the MS Excel writer receives no features for a given table.

When `INIT_TABLES` is set to `FIRSTFEATURE`, the MS Excel writer will only initialize a table once the first feature destined for that table is received. In this mode, if the MS Excel writer does not receive any features for a given table, the table will never be initialized.

Workbench Parameter: *Initialize Tables*

Writer Mode Specification

The MS Excel writer allows the user to specify a writer mode, which determines what database command should be issued for each feature received. Valid writer modes are `INSERT` and `UPDATE`.

Writer Modes

In **INSERT** mode, the attribute values of each received feature are written as a new row on a worksheet.

In **UPDATE** mode, the attribute values of each received feature are used to update existing rows in the worksheet. The rows which are updated are determined via the **xls_update_key_columns** DEF line parameter, or via the **fme_where** attribute on the feature.

Writer Mode Constraints

In **UPDATE** mode, the **fme_where** attribute always takes precedence over the **xls_update_key_columns** DEF line parameter. If both the **fme_where** attribute and the **xls_update_key_columns** DEF line parameter are not present, then **UPDATE** mode will generate an error.

When the **fme_where** attribute is present, it is used verbatim as the WHERE clause on the generated **UPDATE** command. For example, if **fme_where** were set to `'id<5'`, then all database records with field **id** less than 5 will be affected by the command.

When the **fme_where** attribute is not present, the writer looks for the **xls_update_key_columns** DEF line parameter and uses it to determine which records should be affected by the command. Please refer to the DEF section for more information about the **xls_update_key_columns** DEF line parameter.

Writer Mode Selection

The writer mode can be specified at three unique levels: on the writer level, on the feature type, or on individual features.

At the writer level, the writer mode is specified by the **WRITER_MODE** keyword. This keyword can be superseded by the feature type writer mode specification.

Note: For more information on this directive, see the chapter Database Writer Mode.

At the feature type level, the writer mode is specified by the **xls_writer_mode** DEF line parameter. This parameter supersedes the **WRITER_MODE** keyword. Unless this parameter is set to **INSERT**, it may be superseded on individual features by the **fme_db_operation** attribute. Please refer to the DEF line documentation for more information about this parameter.

At the feature level, the writer mode is specified by the **fme_db_operation** attribute. Unless the parameter at the feature type level is set to **INSERT**, the writer mode specified by this attribute always supersedes all other values. Accepted values for the **fme_db_operation** attribute are **INSERT** or **UPDATE**.

Feature Representation

Features read from a database consist of a series of attribute values. They have no geometry. The attribute names are as defined in the **DEF** line if the first form of the **DEF** line was used. If the second form of the **DEF** line was used, then the attribute names are as they are returned by the query, and as such may have their original table names as qualifiers. The feature type of each MS Excel feature is as defined on its **DEF** line.

Features written to the database have the destination table as their feature type, and attributes as defined on the **DEF** line.

Microsoft SQL Server Reader/Writer

Notes: This format is not supported by FME Base Edition.

Overview

The Microsoft SQL Server reader and writer modules provide FME with access to attribute data held in live MS SQL Server database tables. This data may not necessarily have a spatial component to it. The FME provides read and write access to live MS SQL Server databases via Microsoft's ActiveX Data Objects (ADO).

Tip: See the @SQL function in the FME Functions and Factories manual. This function allows arbitrary Structured Query Language (SQL) statements to be executed against any database.

MS SQL Server Quick Facts

Format Type Identifier	MSSQL_ADO
Reader/Writer	Both
Licensing Level	Professional
Dependencies	None
Dataset Type	Database name
Feature Type	Table name
Typical File Extensions	N/A
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	Yes
Encoding Support	Yes
Geometry Type	db_none

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	point	no
circles	no	polygon	no
circular arc	no	raster	no
donut polygon	no	solid	no
elliptical arc	no	surface	no
ellipses	no	text	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
line	no		z values	n/a
none	yes			

Reader Overview

The FME considers a database data set to be a collection of relational tables. The tables must be defined in the mapping file before they can be read. Arbitrary where clauses and joins are fully supported.

Reader Directives

The suffixes listed are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the MS SQL Server reader is **MSSQL_ADO**.

DATASET

Required/Optional: *Required*

This is the database name.

Example:

MSSQL_ADO_DATASET citySource

Workbench Parameter: *Source Microsoft SQL Server Non-spatial Name*

PROVIDER_TYPE

Required/Optional: *Optional*

The type of database provider being used. This keyword is internal to FME and should always be set to **MSSQL_ADO**. For example,

MSSQL_ADO_PROVIDER_TYPE MSSQL_ADO

SERVER

Required/Optional: *Required*

The host name of the MS SQL Server.

MSSQL_ADO_SERVER mi6

Workbench Parameter: *Server*

USER_NAME

Required/Optional: *Optional*

The name of user who will access the database. If Windows Authentication is being used, this is ignored.

MSSQL_ADO_USER_NAME bond007

Workbench Parameter: *User Name*

PASSWORD

Required/Optional: *Optional*

The password of the user accessing the database. If Windows Authentication is being used, this is ignored.

MSSQL_ADO_PASSWORD moneypenny

Workbench Parameter: *Password*

COMMAND_TIMEOUT

The timeout in seconds for a query to the database. If set to zero, there is no timeout. The default is 30.

Required/Optional

Optional

Values

0 = no timeout

Default: 30

Mapping File Syntax

```
MSSQL_SPATIAL_COMMAND_TIMEOUT 15
```

*** Workbench Parameter**

Command Timeout

USE_SSPI

Required/Optional: *Optional*

This specifies whether Windows Authentication should be used to authenticate to the database server. This keyword should be set to either yes or no.

If USE_SSPI is set to yes, then the USER_NAME and PASSWORD keywords are ignored.

```
MSSQL_ADO_USE_SSPI yes
```

Workbench Parameter: *Use Windows Authentication*

DEF

Required/Optional: *Required*

The syntax of the definition is:

```
MSSQL_ADO_DEF <tableName> \  
    [mssql_where_clause <whereclause>] \  
    [<fieldName> <fieldType>] +
```

or

```
MSSQL_ADO_DEF <queryName> \  
    [mssql_sql_statement <sqlQuery>] \  
    \
```

The **<tableName>** must match the name of an existing MS SQL Server table in the database. This will be used as the feature type of all the features read from the table. The exception to this rule is when using the `mssql_sql_statement` keyword. In this case, the **DEF** name may be any valid alphabetic identifier; it does not have to be an existing table name – rather, it is an identifier for the custom SQL query. The feature type of all the features returned from the SQL query are given the query name.

The **<fieldType>** of each field must be given, but it is not verified against the database definition for the field. In effect, it is ignored.

The definition allows specification of separate search parameters for each table. If any of the per table configuration parameters are given, they will override, for that table, whatever global values have been specified by the reader keywords such as the **WHERE_CLAUSE**. If any of these parameters is not specified, the global values will be used.

The following table summarizes the definition line configuration parameters:

Parameter	Contents
mssql_where_clause	This specifies the SQL WHERE clause applied to the attributes of the layer's features to limit the set of features returned. If this is not specified, then all the rows are returned. This keyword will be ignored if the mssql_sql_statement is present.
mssql_sql_statement	This specifies an SQL SELECT query to be used as the source for the results. If this is specified, the MS SQL Server reader will execute the query, and use the resulting rows as the features instead of reading from the table <queryName>. All returned features will have a feature type of <queryName>, and attributes for all columns selected by the query. The mssql_where_clause is ignored if mssql_sql_statement is supplied. This form allows the results of complex joins to be returned to FME.

If no **<whereClause>** is specified, all rows in the table will be read and returned as individual features. If a **<whereClause>** is specified, only those rows that are selected by the clause will be read. Note that the **<whereClause>** does not include the word **WHERE**.

The MS SQL Server reader allows one to use the mssql_sql_statement parameter to specify an arbitrary SQL **SELECT** query on the DEF line. If this is specified, FME will execute the query, and use each row of data returned from the query to define at least one feature. Each of these features will be given the feature type named in the DEF line, and will contain attributes for every column returned by the **SELECT**. In this case, all DEF line parameters regarding a **WHERE** clause or spatial querying are ignored, as it is possible to embed this information directly in the text of the **<sqlQuery>**.

In the following example, the all records whose ID is less than 5 will be read from the supplier table:

```
MSSQL_ADO_DEF supplier \
  mssql_where_clause "id < 5" \
  ID integer \
  NAME char(100) \
  CITY char(50)
```

In this example, the results of joining the **employee** and **city** tables are returned. All attributes from the two tables will be present on each returned feature. The feature type will be set to **complex**.

```
MSSQL_ADO_DEF complex \
  mssql_sql_statement \
    "SELECT * FROM EMPLOYEE, CITY WHERE EMPLOYEE.CITY = CITY.NAME"
```

WHERE_CLAUSE

Required/Optional: *Optional*

This optional specification is used to limit the rows read by the reader from each table. If a given table has no mssql_where_clause or mssql_sql_statement specified in its DEF line, the global **<ReaderKeyword>_WHERE_CLAUSE** value, if present, will be applied as the **WHERE** specifier of the query used to generate the results. If a table's DEF line does contain its own mssql_where_clause or mssql_sql_statement, it will override the global **WHERE** clause.

The syntax for this clause is:

```
MSSQL_ADO_WHERE_CLAUSE <whereClause>
```

Note that the `<whereClause>` does not include the word "WHERE."

The example below selects only the features whose lengths are more than 2000:

```
MSSQL_ADO_WHERE_CLAUSE LENGTH > 2000
```

Workbench Parameter: *Where Clause*

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined database tables that will be read. If no **IDs** are specified, then no tables are read. The syntax of the **IDs** keyword is:

```
MSSQL_ADO_IDS <featureType1> \  
<featureType2> ... \  
<featureTypeN>
```

The feature types must match those used in **DEF** lines.

The example below selects only the **HISTORY** table for input during a translation:

```
MSSQL_ADO_IDS HISTORY
```

READ_CACHE_SIZE

Required/Optional: *Optional*

This keyword controls how the reader retrieves rows from the database. This must be a numeric value which must be greater than 0.

The **READ_CACHE_SIZE** is used to determine the number of rows that are retrieved at one time into local memory from the data source. For example, if the **READ_CACHE_SIZE** is set to 10, after the reader is opened, the reader will read 10 rows into local memory. As features are processed by the FME, the reader returns the data from the local memory buffer. As soon as you move past the last row available in local memory, the reader will retrieve the next 10 rows from the data source.

This keyword affects the performance of the reader, and will result in significantly degraded performance if incorrectly set. The optimum value of this keyword depends primarily on the characteristics of individual records and the transport between the database and the client machine. It is less affected by the quantity of rows that are to be retrieved.

By default, the **READ_CACHE_SIZE** is set to 10. This value has been determined to be the optimal value for average datasets.

Workbench Parameter: *Number of Records To Fetch At A Time*

RETRIEVE_ALL_SCHEMAS

Required/Optional: *Optional*

This directive is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

This optional directive is used to tell the reader to retrieve the names and the schemas of all the tables in the source database. If this value is not specified, it is assumed to be "No".

Range: YES | NO

Default: NO

RETRIEVE_ALL_TABLE_NAMES

Required/Optional: *Optional*

This specification is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

Similar to **RETRIEVE_ALL_SCHEMAS**; this optional directive is used to tell the reader to only retrieve the table names of all the tables in the source database. If **RETRIEVE_ALL_SCHEMAS** is also set to "Yes", then **RETRIEVE_ALL_SCHEMAS** will take precedence. If this value is not specified, it is assumed to be "No".

Range: YES | NO

Default: NO

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

✳ Workbench Parameter

Additional Attributes to Expose

Writer Overview

The MS SQL Server writer module stores attribute records into a live relational database. The MS SQL Server writer provides the following capabilities:

- **Transaction Support:** The MS SQL Server writer provides transaction support that eases the data loading process. Occasionally, a data load operation terminates prematurely due to data difficulties. The transaction support provides a mechanism for reloading corrected data without data loss or duplication.
- **Table Creation:** The MS SQL Server writer uses the information within the FME mapping file to automatically create database tables as needed.
- **Writer Mode Specification:** The MS SQL Server writer allows the user to specify what database command should be issued for each feature received. Valid writer modes are INSERT, UPDATE and DELETE. The writer mode can be specified at three unique levels: on the writer level, on the feature type, or on individual features.

Writer Directives

The directives processed by the MS SQL Server Writer are listed below. The suffixes shown are prefixed by the current **<writerKeyword>** in a mapping file. By default, the **<writerKeyword>** for the MS SQL Server writer is **MSSQL_ADO**.

DATASET, PROVIDER_TYPE, SERVER, USER_NAME, PASSWORD, USE_SSPI, COMMAND_TIMEOUT

These directives operate in the same manner as they do for the MS SQL Server reader. The remaining writer-specific directives are discussed in the following sections.

START_TRANSACTION

Required/Optional: *Optional*

This statement tells the MS SQL Server writer module when to start actually writing features into the database. The MS SQL Server writer does not write any features until the feature is reached that belongs to **<last successful**

`transaction` > + 1. Specifying a value of zero causes every feature to be output. Normally, the value specified is zero – a non-zero value is only specified when a data load operation is being resumed after failing partway through.

Parameter	Contents
<last successful transaction>	The transaction number of the last successful transaction. When loading data for the first time, set this value to 0. Default: 0

Example:

```
MSSQL_ADO_START_TRANSACTION 0
```

Workbench Parameter: *Start Transaction at*

TRANSACTION_INTERVAL

Required/Optional: *Optional*

This statement informs the FME about the number of features to be placed in each transaction before a transaction is committed to the database.

If the `MSSQL_ADO_TRANSACTION_INTERVAL` statement is not specified, then a value of 500 is used as the transaction interval.

Parameter	Contents
<transaction_interval>	The number of features in a single transaction. Default: 500

If the `MSSQL_ADO_TRANSACTION_INTERVAL` is set to zero, then feature based transactions are used. As each feature is processed by the writer, they are checked for an attribute called `fme_db_transaction`. The value of this attribute specifies whether the writer should commit or rollback the current transaction. The value of the attribute can be one of `COMMIT_BEFORE`, `COMMIT_AFTER`, `ROLLBACK_AFTER` or `IGNORE`. If the `fme_db_transaction` attribute is not set in any features, then the entire write operation occurs in a single transaction.

Example:

```
MSSQL_ADO_TRANSACTION_INTERVAL 5000
```

Workbench Parameter: *Transaction interval*

WRITER_MODE

Required/Optional: *Optional*

Note: For more information on this directive, see the chapter *Database Writer Mode*.

This keyword informs the MS SQL Server writer which SQL operations will be performed by default by this writer. This operation can be set to `INSERT`, `UPDATE` or `DELETE`. The default writer level value for this operation can be overwritten at the feature type or table level. The corresponding feature type DEF parameter name is called `mssql_table_writer_mode`. It has the same valid options as the writer level mode and additionally the value `INHERIT_FROM_WRITER` which causes the writer level mode to be inherited by the feature type as the default for features contained in that table.

The operation can be set specifically for individual feature as well. Note that when the writer mode is set to `INSERT` this prevents the mode from being interpreted from individual features and all features are inserted unless otherwise marked as `UPDATE` or `DELETE` features. These are skipped.

If the MSSQL_ADO_WRITER_MODE statement is not specified, then a value of INSERT is given.

Parameter	Contents
<writer_mode>	The type of SQL operation that should be performed by the writer. The valid list of values are below: INSERT UPDATE DELETE Default: INSERT

Example:

```
MSSQL_ADO_WRITER_MODE INSERT
```

Workbench Parameter: *Writer Mode*

BEGIN_SQL{n}

Occasionally you must execute some ad-hoc SQL prior to opening a table. For example, it may be necessary to ensure that a view exists prior to attempting to read from it.

Upon opening a connection to read from a database, the reader looks for the directive <ReaderKeyword>_BEGIN_SQL{n} (for n=0,1,2,...), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the FME_SQL_DELIMITER keyword, embedded at the beginning of the SQL block. The single character following this keyword will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;  
DELETE FROM instructors;  
DELETE FROM people  
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

✳ Workbench Parameter

SQL Statement to Execute Before Translation

END_SQL{n}

Occasionally you must execute some ad-hoc SQL after closing a set of tables. For example, it may be necessary to clean up a temporary view after writing to the database.

Just before closing a connection on a database, the reader looks for the directive <ReaderKeyword>_END_SQL{n} (for n=0,1,2,...), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the **FME_SQL_DELIMITER** directive, embedded at the beginning of the SQL block. The single character following this directive will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;  
DELETE FROM instructors;  
DELETE FROM people  
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

* Workbench Parameter

SQL Statement to Execute After Translation

INIT_TABLES

Required/Optional: *Optional*

This keyword informs the MS SQL Server writer when each table should be initialized. Initialization encompasses the actions of dropping or truncating existing tables, and creating new tables as necessary.

When INIT_TABLES is set to IMMEDIATELY, the MS SQL Server writer will initialize all tables immediately after parsing the DEF lines and opening the connection to the database. In this mode, all tables will be initialized, even if the MS SQL Server writer receives no features for a given table.

When INIT_TABLES is set to FIRSTFEATURE, the MS SQL Server writer will only initialize a table once the first feature destined for that table is received. In this mode, if the MS SQL Server writer does not receive any features for a given table, the table will never be initialized.

Workbench Parameter: *Initialize Tables*

Writer Mode Specification

The MS SQL Server writer allows the user to specify a writer mode, which determines what database command should be issued for each feature received. Valid writer modes are **INSERT**, **UPDATE** and **DELETE**.

Writer Modes

In **INSERT** mode, the attribute values of each received feature are written as a new database record.

In **UPDATE** mode, the attribute values of each received feature are used to update existing records in the database. The records which are updated are determined via the **mssql_update_key_columns** DEF line parameter, or via the **fme_where** attribute on the feature.

In **DELETE** mode, existing database records are deleted according to the information specified in the received feature. Records are selected for deletion using the same technique as records are selected for updating in **UPDATE** mode.

Writer Mode Constraints

In **UPDATE** and **DELETE** mode, the **fme_where** attribute always takes precedence over the **mssql_update_key_columns** DEF line parameter. If both the **fme_where** attribute and the **mssql_update_key_columns** DEF line parameter are not present, then **UPDATE** or **DELETE** mode will generate an error.

When the `fme_where` attribute is present, it is used verbatim as the WHERE clause on the generated `UPDATE` or `DELETE` command. For example, if `fme_where` were set to `'id<5'`, then all database records with field `id` less than 5 will be affected by the command.

When the `fme_where` attribute is not present, the writer looks for the `mssql_update_key_columns` DEF line parameter and uses it to determine which records should be affected by the command. Please refer to "DEF" on page 915 for more information about the `mssql_update_key_columns` DEF line parameter.

Writer Mode Selection

The writer mode can be specified at three unique levels: on the writer level, on the feature type, or on individual features.

At the writer level, the writer mode is specified by the `WRITER_MODE` keyword. This keyword can be superseded by the feature type writer mode specification. Please refer to the `WRITER_MODE` keyword documentation for more information about this keyword.

At the feature type level, the writer mode is specified by the `mssql_writer_mode` DEF line parameter. This parameter supersedes the `WRITER_MODE` keyword. Unless this parameter is set to `INSERT`, it may be superseded on individual features by the `fme_db_operation` attribute. Please refer to the DEF line documentation for more information about this parameter.

At the feature level, the writer mode is specified by the `fme_db_operation` attribute. Unless the parameter at the feature type level is set to `INSERT`, the writer mode specified by this attribute always supersedes all other values. Accepted values for the `fme_db_operation` attribute are `INSERT`, `UPDATE` or `DELETE`.

Table Representation

Each MS SQL Server table must be defined before it can be written. The general form of a MS SQL Server definition statement is:

```
MSSQL_ADO_DEF <tableName> \
    [mssql_update_key_columns <keyColumns>] \
    [mssql_drop_table (yes|no)] \
    [mssql_truncate_table (yes|no)] \
    [mssql_table_writer_mode (inherit_from_writer|insert|update|delete)] \
    [<fieldName> <fieldType>[,<indexType>]]+
```

The table definition allows control of the table that will be created. If the fields and types are listed, the types must match those in the database. Fields which can contain NULL values do not need to be listed - these fields will be filled with NULL values.

If the table does not exist, then the field names and types are used to first create the table. In any case, if a `<fieldType>` is given, it may be any field type supported by the target database.

The configuration parameters present on the definition line are described in the following table:

Parameter	Contents
tableName	The name of the table to be written. If a table with the specified name exists, it will be overwritten if the <code>mssql_drop_table</code> DEF line parameter is set to YES, or it will be truncated if the <code>mssql_truncate_table</code> DEF line parameter is set to YES. Otherwise the table will be appended. Valid values for table names include any character string devoid of SQL-offensive characters and less than 128 characters in length.

Parameter	Contents
mssql_table_writer_mode	<p>The the default operation mode of the feature type in terms of the types of SQL statements sent to the database. Valid values are INSERT, UPDATE, DELETE and INHERIT_FROM_WRITER. Note that INSERT mode allows for only INSERT operations where as UPDATE and DELETE can be overwritten at the feature levels. INHERIT_FROM_WRITER simply indicates to take this value from the writer level and not to override it at the feature type level.</p> <p>Default: INHERIT_FROM_WRITER</p>
mssql_update_key_columns	<p>This is a comma-separated list of the columns which are matched against the corresponding FME attributes' values to specify which rows are to be updated or deleted when the writer mode is either UPDATE or INSERT.</p> <p>For example: mssql_update_key_columns ID would instruct the writer to ensure that the FME attribute is always matched against the column with the same name. Also, the target table is always the feature type specified in the DEF line.</p> <p>Each column listed with the mssql_update_key_columns keyword must be defined with a type on the DEF line, in addition to the columns whose values will be updated by the operation.</p>
mssql_drop_table	<p>This specifies that if the table exists by this name, it should be dropped and replaced with a table specified by this definition.</p> <p>Default: NO</p>
mssql_truncate_table	<p>This specifies that if the table exists by this name, it should be cleared prior to writing.</p> <p>Default: NO</p>
fieldName	<p>The name of the field to be written. Valid values for field name include any character string devoid of SQL-offensive characters and less than 128 characters in length.</p>
fieldType	<p>See the Attribute Types section below.</p>
indexType	<p>The type of index to create for the column.</p> <p>If the table does not previously exist, then upon table creation, a database index of the specified type is created. The database index contains only the one column. Remember that a given table can contain at most</p>

Parameter	Contents
	<p>one clustered index.</p> <p>The valid values for the column type are listed below:</p> <ul style="list-style-type: none"> • indexed: An index without constraints. • indexed_not_null: An index with a non-nullable constraint. • unique: An index with a unique constraint. • uniqueclustered: A clustered index with a unique constraint. • clustered: A clustered index without constraints. • clustered_not_null: A clustered index with a non-nullable constraint. • not_null: A non-nullable column. • primary_key: A primary key with non-nullable and unique constraints.

Attribute Types

This section of the `<writerKeyword>_DEF` statement defines the attribute types for a table.

bigint

This type is used to represent 64-bit signed integers.

int

This type is used to represent 32-bit signed integers.

smallint

This type is used to represent 16-bit signed integers.

tinyint

This type is used to represent numbers between 0 and 255.

bit

This type is used to represent an integer with a value of 1 or 0

decimal

This type is used to represent fixed precision and scale numeric data from $-10^{38}+1$ to $10^{38}+1$.

numeric

This type is used to represent fixed precision and scale numeric data from $-10^{38}+1$ to $10^{38}+1$.

money

This type is used to represent monetary data values from -2^{63} to $2^{63}-1$. Attribute values are real numbers such as 55.2354.

smallmoney

This type is used to represent monetary data values from -214748.3648 to 214748.3647. Attribute values are real numbers such as 55.2354.

float

This type is used to represent 32-bit floating precision numbers.

real

This type is used to represent 16-bit floating precision numbers.

date

This type is used to represent date data from January 1, 0001 to December 31, 9999. For example, a value of 20061231 represents December 31, 2006. When writing to a pre-2008 SQL Server this type will be converted to date-time.

time

This type is used to represent time data with an accuracy of 100 nanoseconds. For example, a value of 235959.0000000 represents 11:59:59PM. When writing to a pre-2008 SQL Server this type will be converted to date-time.

datetime2

This type is used to represent date and time data from January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds. For example, a value of 20061231235959.0000000 represents 11:59:59PM on December 31, 2006. When writing to a pre-2008 SQL Server this type will be converted to datetime.

datetime

This type is used to represent date and time data from January 1, 1753 to December 31, 9999 with an accuracy of 3.33 milliseconds. For example, a value of 20061231235959 represents 11:59:59PM on December 31, 2006. When writing to the database, the writer expects the date attribute to be in the form YYYYMMDDHHMMSS.

smalldatetime

This type is used to represent date and time data from January 1, 1900 to June 6, 2079 with an accuracy of one minute. For example, a value of 20060101101000 represents 10:10:00AM on January 1, 2006. When writing to the database, the writer expects the date attribute to be in the form YYYYMMDDHHMMSS.

char

This type is used to represent fixed length character data up to a length of 8000 characters.

varchar

This type is used to represent variable length character data up to a length of 8000 characters.

text

This type is used to represent variable length character data up to a length of $2^{31}-1$ characters.

nchar

This type is used to represent fixed length character data up to a length of 4000 characters.

nvarchar

This type is used to represent variable length character data up to a length of 4000 characters.

binary

This type is used to represent fixed length binary data up to a length of 8000 bytes.

varbinary

This type is used to represent variable length binary data up to a length of 8000 bytes.

image

This type is used to represent variable length binary data up to a length of $2^{31}-1$ bytes.

uniqueidentifier

Uniqueidentifier is used to represent GUID's. As such, it must be set up like a valid GUID. Example: {B85E62C3-DC56-40C0-852A-49F759AC68FB}

Note: the {} must be present for the GUID to be written successfully.

identity

This type is used to represent an auto-incrementing integer field.

If you try to write a value to it; the value will be ignored and the database will simply take the largest integer in the column, increment it and put that number into the field.

Feature Representation

Features read from a database consist of a series of attribute values. They have no geometry. The attribute names are as defined in the **DEF** line if the first form of the **DEF** line was used. If the second form of the **DEF** line was used, then the attribute names are as they are returned by the query, and as such may have their original table names as qualifiers. The feature type of each MS SQL Server feature is as defined on its **DEF** line.

Features written to the database have the destination table as their feature type, and attributes as defined on the **DEF** line.

Microsoft SQL Server (Spatial) Reader/Writer

Format Notes: This format is not supported by FME Base Edition.

Overview

The Microsoft SQL Server (Spatial) reader and writer modules provide FME with access to spatial and attribute data held in live MS SQL Server database tables. The FME provides read and write access to live MS SQL Server databases via Microsoft's ActiveX Data Objects (ADO).

MS SQL Server (Spatial) Quick Facts

Format Type Identifier	MSSQL_SPATIAL
Reader/Writer	Both
Licensing Level	Reading: Professional Writing: DB2, Oracle, or SQL Server Edition
Dependencies	SQL Server 2008+
Dataset Type	Database
Feature Type	Table name
Typical File Extensions	N/A
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Never
Schema Required	No
Transaction Support	Yes
Encoding Support	Yes
Geometry Type	mssql_spatial_type

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	yes	point	yes
circles	no	polygon	yes
circular arc	no	raster	no
donut polygon	yes	solid	no
elliptical arc	no	surface	no
ellipses	no	text	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
line	yes		z values	Reader: yes Writer: no
none	yes			

Reader Overview

FME considers a database dataset to be a collection of relational tables. Arbitrary where clauses and joins are fully supported.

Reader Directives

The suffixes listed are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the MS SQL Server (Spatial) reader is `MSSQL_SPATIAL`.

SERVER

Required/Optional: *Required*

The host name of the MS SQL Server.

`MSSQL_SPATIAL_SERVER mi6`

Workbench Parameter: *Server*

USE_SSPI

Required/Optional: *Optional*

This specifies whether Windows Authentication should be used to authenticate to the database server. This keyword should be set to either yes or no.

If USE_SSPI is set to yes, then the USER_NAME and PASSWORD keywords are ignored.

`MSSQL_SPATIAL_USE_SSPI yes`

Workbench Parameter: *Use Windows Authentication*

USER_NAME

Required/Optional: *Optional*

The name of user who will access the database. If Windows Authentication is being used, this is ignored.

`MSSQL_SPATIAL_USER_NAME bond007`

Workbench Parameter: *User Name*

PASSWORD

Required/Optional: *Optional*

The password of the user accessing the database. If Windows Authentication is being used, this is ignored.

`MSSQL_SPATIAL_PASSWORD moneypenny`

Workbench Parameter: *Password*

COMMAND_TIMEOUT

The timeout in seconds for a query to the database. If set to zero, there is no timeout. The default is 30.

Required/Optional

Optional

Values

0 = no timeout

Default: 30

Mapping File Syntax

```
MSSQL_SPATIAL_COMMAND_TIMEOUT 15
```

* Workbench Parameter

Command Timeout

DATASET

Required/Optional: *Required*

This is the database name.

Example:

```
MSSQL_SPATIAL_DATASET citySource
```

Workbench Parameter: *Source Microsoft SQL Server Spatial Name*

WHERE_CLAUSE

Required/Optional: *Optional*

This optional specification is used to limit the rows read by the reader from each table. If a given table has no `mssql_where_clause` or `mssql_sql_statement` specified in its `DEF` line, the global `<ReaderKeyword>_WHERE_CLAUSE` value, if present, will be applied as the `WHERE` specifier of the query used to generate the results. If a table's `DEF` line does contain its own `mssql_where_clause` or `mssql_sql_statement`, it will override the global `WHERE` clause.

The syntax for this clause is:

```
MSSQL_SPATIAL_WHERE_CLAUSE <whereClause>
```

Note that the `<whereClause>` does not include the word "WHERE."

The example below selects only the features whose lengths are more than 2000:

```
MSSQL_SPATIAL_WHERE_CLAUSE LENGTH > 2000
```

Workbench Parameter: *Where Clause*

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined database tables that will be read. If no `IDs` are specified, then no tables are read. The syntax of the `IDs` keyword is:

```
MSSQL_SPATIAL_IDS <featureType1> \  
<featureType2> ... \  
<featureTypeN>
```

The feature types must match those used in `DEF` lines.

The example below selects only the `HISTORY` table for input during a translation:

```
MSSQL_SPATIAL_IDS HISTORY
```

DEF

Required/Optional: *Optional*

The syntax of the definition is:

```
MSSQL_SPATIAL_DEF <tableName> \  
  [(mssql_geom_column <geometry column name>) \  
   |(mssql_geog_column <geography column name>)] \  
  [mssql_where_clause <whereClause>] \  
  [<fieldName> <fieldType>] +
```

or

```
MSSQL_SPATIAL_DEF <queryName> \  
  [(mssql_geom_column <geometry column name>) \  
   |(mssql_geog_column <geography column name>)] \  
  [mssql_sql_statement <sqlQuery>] \  
  [<fieldName> <fieldType>] +
```

The **<tableName>** must match the name of an existing MS SQL Server table in the database. This will be used as the feature type of all the features read from the table. The exception to this rule is when using the `mssql_sql_statement` keyword. In this case, the **DEF** name may be any valid alphabetic identifier; it does not have to be an existing table name – rather, it is an identifier for the custom SQL query. The feature type of all the features returned from the SQL query are given the query name.

The **<fieldType>** of each field must be given, but it is not verified against the database definition for the field. In effect, it is ignored.

The definition allows specification of separate search parameters for each table. If any of the per table configuration parameters are given, they will override, for that table, whatever global values have been specified by the reader keywords such as the **WHERE_CLAUSE**. If any of these parameters is not specified, the global values will be used.

The following table summarizes the definition line configuration parameters:

Parameter	Contents
mssql_geom_column	This specifies the column FME will read geometry from. If the selected table has exactly one spatial column, and neither the <code>mssql_geom_column</code> or <code>mssql_geog_column</code> keywords are used, FME will determine the spatial column automatically. Only one of <code>mssql_geom_column</code> and <code>mssql_geog_column</code> may be specified on each DEF line.
mssql_geog_column	This specifies the column FME will read geography from. If the selected table has exactly one spatial column, and neither the <code>mssql_geom_column</code> or <code>mssql_geog_column</code> keywords are used, FME will determine the spatial column automatically. Only one of <code>mssql_geom_column</code> and <code>mssql_geog_column</code> may be specified on each DEF line.
mssql_where_clause	This specifies the SQL WHERE clause applied to the attributes of the layer's features to limit the set of features returned. If this is not specified, then all the rows are returned. This keyword will be ignored if the <code>mssql_sql_statement</code> is present.

Parameter	Contents
mssql_sql_statement	This specifies an SQL SELECT query to be used as the source for the results. If this is specified, the MS SQL Server (Spatial) reader will execute the query, and use the resulting rows as the features instead of reading from the table <queryName>. All returned features will have a feature type of <queryName>, and attributes for all columns selected by the query. The mssql_where_clause is ignored if mssql_sql_statement is supplied. This form allows the results of complex joins to be returned to FME.

If no <whereClause> is specified, all rows in the table will be read and returned as individual features. If a <whereClause> is specified, only those rows that are selected by the clause will be read. Note that the <whereClause> does not include the word **WHERE**.

The MS SQL Server (Spatial) reader allows one to use the mssql_sql_statement parameter to specify an arbitrary SQL **SELECT** query on the DEF line. If this is specified, FME will execute the query, and use each row of data returned from the query to define one feature. Each of these features will be given the feature type named in the DEF line, and will contain attributes for every column returned by the **SELECT**. In this case, all DEF line parameters regarding a **WHERE** clause or spatial querying are ignored, as it is possible to embed this information directly in the text of the <sqlQuery>.

In the following example, all records whose ID is less than 5 will be read from the supplier table. If the supplier table contains a geometry column, it will be automatically detected and read.

```
MSSQL_SPATIAL_DEF supplier \
  mssql_where_clause "id < 5" \
  ID integer \
  NAME char(100) \
  CITY char(50)
```

In this example, the results of joining the **employee** and **city** tables are returned. All attributes from the two tables will be presented on each returned feature. The feature type will be set to **complex**. Geometry will be read from the GEOM column in the CITY table.

```
MSSQL_SPATIAL_DEF complex \
  mssql_geom_column GEOM \
  mssql_sql_statement \
  "SELECT * FROM EMPLOYEE, CITY WHERE EMPLOYEE.CITY = CITY.NAME"
```

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

READ_CACHE_SIZE

Required/Optional: *Optional*

This keyword controls how the reader retrieves rows from the database. This must be a numeric value which must be greater than 0.

The READ_CACHE_SIZE is used to determine the number of rows that are retrieved at one time into local memory from the data source. For example, if the READ_CACHE_SIZE is set to 10, after the reader is opened, the reader will read 10 rows into local memory. As features are processed by the FME, the reader returns the data from the local memory buffer. As soon as you move past the last row available in local memory, the reader will retrieve the next 10 rows from the data source.

This keyword affects the performance of the reader, and will result in significantly degraded performance if incorrectly set. The optimum value of this keyword depends primarily on the characteristics of individual records and the

transport between the database and the client machine. It is less affected by the quantity of rows that are to be retrieved.

By default, the READ_CACHE_SIZE is set to 10. This value has been determined to be the optimal value for average datasets.

Workbench Parameter: *Number of Records to Fetch at a Time*

ASSUME_ONE_SRID_PER_COL

Required/Optional: *Optional*

SQL Server does not constrain all geometry objects in a column to have the same Spatial Reference ID (SRID). However, it is common practice to use a single SRID within a given column. If this directive is set to "NO", FME will not assume that each geometry column uses a single SRID when querying the database.

Parameter	Contents
<assume_one_srid_per_col>	"YES" or "NO" Default: YES

Example:

```
MSSQL_SPATIAL_ASSUME_ONE_SRID_PER_COL YES
```

Workbench Parameter: *Geometry Columns Have Exactly One SRID*

PERSISTENT_CONNECTION

Required/Optional: *Optional*

If this directive is set to YES, database connections will be left open and reused when possible until FME is shut down.

The syntax of the PERSISTENT_CONNECTION directive is:

```
<ReaderKeyword>_PERSISTENT_CONNECTION [yes | no]
```

Workbench Parameter: *Make Connection Persistent*

PROVIDER_TYPE

Required/Optional: *Optional*

The type of database provider being used. This keyword is internal to FME and, if specified, should always be set to MSSQL_SPATIAL. For example,

```
MSSQL_SPATIAL_PROVIDER_TYPE MSSQL_SPATIAL
```

HANDLE_MULTIPLE_SPATIAL_COLUMNS

If this directive is set to YES, feature geometry will be read into an aggregate. A directive is set on the aggregate to indicate that each part of the aggregate is independent from the others, and its own geometry. Geometry parts of the aggregate are named and contain geometry according to their respective column in the table being read.

When using this feature, neither the geometry/geography column, nor the feature type SELECT statement can be specified.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_HANDLE_MULTIPLE_SPATIAL_COLUMNS YES
```

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The MS SQL Server (Spatial) writer module stores attribute records into a live relational database. The MS SQL Server (Spatial) writer provides the following capabilities:

- **Transaction Support:** The MS SQL Server (Spatial) writer provides transaction support that eases the data loading process. Occasionally, a data load operation terminates prematurely due to data difficulties. The transaction support provides a mechanism for reloading corrected data without data loss or duplication.
- **Table Creation:** The MS SQL Server (Spatial) writer uses the information within the FME mapping file to automatically create database tables as needed.
- **Writer Mode Specification:** The MS SQL Server (Spatial) writer allows the user to specify what database command should be issued for each feature received. Valid writer modes are INSERT, UPDATE and DELETE. The writer mode can be specified at three unique levels: on the writer level, on the feature type, or on individual features.

Writer Directives

The directives processed by the MS SQL Server (Spatial) Writer are listed below. The suffixes shown are prefixed by the current `<WriterKeyword>` in a mapping file. By default, the `<WriterKeyword>` for the MS SQL Server (Spatial) writer is `MSSQL_SPATIAL`.

SERVER

Required/Optional: *Required*

The host name of the MS SQL Server.

```
MSSQL_SPATIAL_SERVER mi6
```

Workbench Parameter: *Server*

USE_SSPI

Required/Optional: *Optional*

This specifies whether Windows Authentication should be used to authenticate to the database server. This keyword should be set to either yes or no.

If USE_SSPI is set to yes, then the USER_NAME and PASSWORD keywords are ignored.

```
MSSQL_SPATIAL_USE_SSPI yes
```

Workbench Parameter: *Use Windows Authentication*

USER_NAME

Required/Optional: *Optional*

The name of user who will access the database. If Windows Authentication is being used, this is ignored.

`MSSQL_SPATIAL_USER_NAME bond007`

Workbench Parameter: *User Name*

PASSWORD

Required/Optional: *Optional*

The password of the user accessing the database. If Windows Authentication is being used, this is ignored.

`MSSQL_SPATIAL_PASSWORD moneypenny`

Workbench Parameter: *Password*

COMMAND_TIMEOUT

The timeout in seconds for a query to the database. If set to zero, there is no timeout. The default is 30.

Required/Optional

Optional

Values

0 = no timeout

Default: 30

Mapping File Syntax

`MSSQL_SPATIAL_COMMAND_TIMEOUT 15`

*** Workbench Parameter**

Command Timeout

DATASET

Required/Optional: *Required*

This is the database name.

Example:

`MSSQL_SPATIAL_DATASET citySource`

Workbench Parameter: *Destination Microsoft SQL Server Spatial Name*

DEF

Required/Optional: *Required*

See Table Representation for details.

WRITER_MODE

Required/Optional: *Optional*

Note: For more information on this directive, see the chapter *Database Writer Mode*.

This directive informs the MS SQL Server (Spatial) writer which SQL operations will be performed by default by this writer. This operation can be set to **INSERT**, **UPDATE** or **DELETE**. The default writer level value for this operation can be overwritten at the feature type or table level. The corresponding feature type DEF parameter name is called `mssql_table_writer_mode`. It has the same valid options as the writer level mode and additionally the value **INHERIT_FROM_WRITER** which causes the writer level mode to be inherited by the feature type as the default for features contained in that table.

The operation can be set specifically for individual features as well. Note that when the writer mode is set to **INSERT** this prevents the mode from being interpreted from individual features and all features are inserted.

If the `MSSQL_SPATIAL_WRITER_MODE` statement is not specified, then a value of **INSERT** is given.

Parameter	Contents
<writer_mode>	The type of SQL operation that should be performed by the writer. The valid list of values are below: INSERT UPDATE DELETE Default: INSERT

Example:

```
MSSQL_SPATIAL_WRITER_MODE INSERT
```

Workbench Parameter: *Writer Mode*

INIT_TABLES

Required/Optional: *Optional*

This keywords informs the MS SQL Server (Spatial) writer when each table should be initialized. Initialization encompasses the actions of dropping or truncating existing tables, and creating new tables as necessary.

When `INIT_TABLES` is set to **IMMEDIATELY**, the MS SQL Server (Spatial) writer will initialize all tables immediately after parsing the DEF lines and opening the connection to the database. In this mode, all tables will be initialized, even if the MS SQL Server (Spatial) writer receives no features for a given table.

When `INIT_TABLES` is set to **FIRSTFEATURE**, the MS SQL Server (Spatial) writer will only initialize a table once the first feature destined for that table is received. In this mode, if the MS SQL Server (Spatial) writer does not receive any features for a given table, the table will never be initialized.

Workbench Parameter: *Initialize Tables*

BEGIN_SQL{n}

Occasionally you must execute some ad-hoc SQL prior to opening a table. For example, it may be necessary to ensure that a view exists prior to attempting to read from it.

Upon opening a connection to read from a database, the reader looks for the directive `<ReaderKeyword>_BEGIN_SQL{n}` (for `n=0,1,2,...`), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the `FME_SQL_DELIMITER` keyword, embedded at the beginning of the SQL block. The single character following this keyword will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;
DELETE FROM instructors;
```

```
DELETE FROM people
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

* Workbench Parameter

SQL Statement to Execute Before Translation

END_SQL{n}

Occasionally you must execute some ad-hoc SQL after closing a set of tables. For example, it may be necessary to clean up a temporary view after writing to the database.

Just before closing a connection on a database, the reader looks for the directive `<ReaderKeyword>_END_SQL{n}` (for $n=0,1,2,\dots$), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the `FME_SQL_DELIMITER` directive, embedded at the beginning of the SQL block. The single character following this directive will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;
DELETE FROM instructors;
DELETE FROM people
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

* Workbench Parameter

SQL Statement to Execute After Translation

START_TRANSACTION

Required/Optional: *Optional*

This statement tells the MS SQL Server (Spatial) writer module when to start actually writing features into the database. The MS SQL Server (Spatial) writer does not write any features until the feature is reached that belongs to `<last successful transaction> + 1`. Specifying a value of zero causes every feature to be output. Normally, the value specified is zero – a non-zero value is only specified when a data load operation is being resumed after failing partway through.

Parameter	Contents
<last successful transaction>	The transaction number of the last suc-

Parameter	Contents
	<p>successful transaction. When loading data for the first time, set this value to 0.</p> <p>Default: 0</p>

Example:

`MSSQL_SPATIAL_START_TRANSACTION 0`

Workbench Parameter: *Start transaction at*

TRANSACTION_INTERVAL

Required/Optional: *Optional*

This statement informs the FME about the number of features to be placed in each transaction before a transaction is committed to the database.

If the `TRANSACTION_INTERVAL` statement is not specified, then a value of 500 is used as the transaction interval.

Parameter	Contents
<transaction_interval>	<p>The number of features in a single transaction.</p> <p>Default: 500</p>

If the `TRANSACTION_INTERVAL` is set to zero, then feature based transactions are used. As each feature is processed by the writer, they are checked for an attribute called `fme_db_transaction`. The value of this attribute specifies whether the writer should commit or rollback the current transaction. The value of the attribute can be one of `COMMIT_BEFORE`, `COMMIT_AFTER`, `ROLLBACK_AFTER` or `IGNORE`. If the `fme_db_transaction` attribute is not set in any features, then the entire write operation occurs in a single transaction.

Example:

`MSSQL_SPATIAL_TRANSACTION_INTERVAL 5000`

Workbench Parameter: *Transaction interval*

SPATIAL_TYPE

Required/Optional: *Optional*

This directive specifies whether to write geometry (planar data) or geography (geodetic data) when writing to tables whose DEF line does not include with the `mssql_geom_column` or `mssql_geog_column` keywords.

This directive only has effect in combination with the `SPATIAL_COLUMN` directive.

Parameter	Contents
<spatial_type>	<p>"geometry" or "geography"</p> <p>Default: geometry</p>

Example:

`MSSQL_SPATIAL_SPATIAL_TYPE geometry`

Workbench Parameter: *Spatial Type*

SPATIAL_COLUMN

Required/Optional: *Optional*

This directive specifies the geometry or geography column to use when writing to tables whose DEF line does not include with the `mssql_geom_column` or `mssql_geog_column` keywords.

This directive only has effect in combination with the `SPATIAL_TYPE` directive.

Parameter	Contents
<spatial_column>	The spatial column name. Default: GEOM

Example:

```
MSSQL_SPATIAL_SPATIAL_COLUMN GEOM
```

Workbench Parameter: *Spatial Column*

ORIENT_POLYGONS

Required/Optional: *Optional*

When writing geography (geodetic) data, polygons must be oriented according to the left-hand rule: outer boundaries must be counter-clockwise and inner boundaries must be clockwise. If this directive is set to "NO", FME will not automatically reorient polygons. You may wish to disable this feature if your input polygons are known to have correct orientation. Note that FME determines polygon orientation by projecting features onto a plane which does not wrap around the earth's poles or 180 degree meridian and does not take into account the curvature of the earth. Therefore, FME may (re)orient polygons incorrectly in some cases.

Parameter	Contents
<reorient_polygons>	"YES" or "NO" Default: YES

Example:

```
MSSQL_SPATIAL_ORIENT_POLYGONS YES
```

Workbench Parameter: *Orient Polygons*

PROVIDER_TYPE

Required/Optional: *Optional*

The type of database provider being used. This keyword is internal to FME and, if specified, should always be set to `MSSQL_SPATIAL`. For example,

```
MSSQL_SPATIAL_PROVIDER_TYPE MSSQL_SPATIAL
```

HANDLE_MULTIPLE_SPATIAL_COLUMNS

If this directive is set to YES, feature geometry will be written from an aggregate.

This aggregate must contain individual geometries, namely that each part is independent from the others and is its own complete geometry. Each part geometry of the aggregate must have a name. If the aggregate contains geometries with names that match the spatial columns of the table being written, the geometries will be written to the appropriate columns.

When using this feature, the geometry/geography columns cannot be specified.

Required/Optional

Optional

Mapping File Syntax

```
<writerKeyword>_HANDLE_MULTIPLE_SPATIAL_COLUMNS YES
```

Writer Mode Specification

The MS SQL Server (Spatial) writer allows the user to specify a writer mode, which determines what database command should be issued for each feature received. Valid writer modes are **INSERT**, **UPDATE** and **DELETE**.

Writer Modes

In **INSERT** mode, the attribute values of each received feature are written as a new database record.

In **UPDATE** mode, the attribute values of each received feature are used to update existing records in the database. The records which are updated are determined via the `mssql_update_key_columns` DEF line parameter, or via the `fme_where` attribute on the feature.

In **DELETE** mode, existing database records are deleted according to the information specified in the received feature. Records are selected for deletion using the same technique as records are selected for updating in **UPDATE** mode.

Writer Mode Constraints

In **UPDATE** and **DELETE** mode, the `fme_where` attribute always takes precedence over the `mssql_update_key_columns` DEF line parameter. If both the `fme_where` attribute and the `mssql_update_key_columns` DEF line parameter are not present, then **UPDATE** or **DELETE** mode will generate an error.

When the `fme_where` attribute is present, it is used verbatim as the WHERE clause on the generated **UPDATE** or **DELETE** command. For example, if `fme_where` were set to 'id<5', then all database records with field id less than 5 will be affected by the command.

When the `fme_where` attribute is not present, the writer looks for the `mssql_update_key_columns` DEF line parameter and uses it to determine which records should be affected by the command. Please refer to [Table Representation](#) for more information about the `mssql_update_key_columns` DEF line parameter.

Writer Mode Selection

The writer mode can be specified at three unique levels: on the writer level, on the feature type, or on individual features.

At the writer level, the writer mode is specified by the `WRITER_MODE` keyword. This keyword can be superseded by the feature type writer mode specification. **Note:** For more information on this directive, see the chapter *Database Writer Mode*.

At the feature type level, the writer mode is specified by the `mssql_writer_mode` DEF line parameter. This parameter supersedes the `WRITER_MODE` keyword. Unless this parameter is set to **INSERT**, it may be superseded on individual features by the `fme_db_operation` attribute. Please refer to the DEF line documentation for more information about this parameter.

At the feature level, the writer mode is specified by the `fme_db_operation` attribute. Unless the parameter at the feature type level is set to **INSERT**, the writer mode specified by this attribute always supersedes all other values. Accepted values for the `fme_db_operation` attribute are **INSERT**, **UPDATE** or **DELETE**.

Table Representation

Each MS SQL Server table must be defined before it can be written. The general form of a MS SQL Server (Spatial) writer definition statement is:

```
MSSQL_SPATIAL_DEF <tableName> \  
    [mssql_update_key_columns <keyColumns>] \  
    [mssql_drop_table (yes|no)] \  
    [mssql_truncate_table (yes|no)] \  
    [mssql_table_writer_mode (inherit_from_writer|insert|
```

```

update|delete)) \
[(mssql_create_index_sql <SQL>) \
[(mssql_geom_column <geometryColumn>) \
|(mssql_geog_column <geographyColumn>)] \
[<fieldName> <fieldType>[,<indexType>]]+

```

The table definition allows control of the table that will be created. If the fields and types are listed, the types must match those in the database. Fields which can contain NULL values do not need to be listed - these fields will be filled with NULL values.

If the table does not exist, then the field names and types are used to first create the table. In any case, if a <field-
Type> is given, it may be any field type supported by the target database.

The configuration parameters present on the definition line are described in the following table:

Parameter	Contents
tableName	The name of the table to be written. If a table with the specified name exists, it will be overwritten if the <code>mssql_drop_table DEF</code> line parameter is set to <code>YES</code> , or it will be truncated if the <code>mssql_truncate_table DEF</code> line parameter is set to <code>YES</code> . Otherwise the table will be appended. Valid values for table names include any character string devoid of SQL-offensive characters and less than 128 characters in length.
mssql_table_writer_mode	The the default operation mode of the feature type in terms of the types of SQL statements sent to the database. Valid values are <code>INSERT</code> , <code>UPDATE</code> , <code>DELETE</code> and <code>INHERIT_FROM_WRITER</code> . Note that <code>INSERT</code> mode allows for only <code>INSERT</code> operations where as <code>UPDATE</code> and <code>DELETE</code> can be overwritten at the feature levels. <code>INHERIT_FROM_WRITER</code> simply indicates to take this value from the writer level and not to override it at the feature type level. Default: <code>INHERIT_FROM_WRITER</code>
mssql_update_key_columns	This is a comma-separated list of the columns which are matched against the corresponding FME attributes' values to specify which rows are to be updated or deleted when the writer mode is either <code>UPDATE</code> or <code>INSERT</code> . For example: <code>mssql_update_key_columns ID</code> would instruct the writer to ensure that the FME attribute is always matched against the column with the same name. Also, the target table is always the feature type specified in the DEF line. Each column listed with the <code>mssql_update_key_columns</code> keyword must be defined with a type on the DEF line, in addition to the columns whose values will be updated by the operation.
mssql_drop_table	This specifies that if the table exists by this name, it should be dropped and replaced with a table specified by this def-

Parameter	Contents
	<p>initiation.</p> <p>Default: NO</p>
mssql_truncate_table	<p>This specifies that if the table exists by this name, it should be cleared prior to writing.</p> <p>Default: NO</p>
geometryColumn	<p>The name of the field containing geometry. The MS SQL Server (Spatial) Writer supports multiple columns per table when used with the <code>HANDLE_MULTIPLE_SPATIAL_COLUMNS</code> directive. The geometry column should not be listed with the other fields in the <code><fieldName></code> and <code><fieldType></code> parameters.</p>
geographyColumn	<p>The name of the field containing geography. The MS SQL Server (Spatial) Writer supports multiple columns per table when used with the <code>HANDLE_MULTIPLE_SPATIAL_COLUMNS</code> directive. The geography column should not be listed with the other fields in the <code><fieldName></code> and <code><fieldType></code> parameters.</p>
fieldName	<p>The name of the field to be written. Valid values for field name include any character string devoid of SQL-offensive characters and less than 128 characters in length.</p>
indexType	<p>The type of index to create for the column.</p> <p>If the table does not previously exist, then upon table creation, a database index of the specified type is created. The database index contains only the one column. Remember that a given table can contain at most one clustered index.</p> <p>The valid values for the column type are listed below:</p> <ul style="list-style-type: none"> • indexed: An index without constraints. • unique: An index with a unique constraint. • uniqueclustered: A clustered index with a unique constraint. • clustered: A clustered index without constraints. • indexed: An index without constraints. • indexed_not_null: An index with a non-nullable constraint. • unique: An index with a unique constraint. • uniqueclustered: A clustered index with a unique constraint. • clustered: A clustered index without constraints. • clustered_not_null: A clustered index with a non-nullable constraint. • not_null: A non-nullable column. • primary_key: A primary key with non-nullable and

Parameter	Contents
	unique constraints.
mssql_create_index_sql	<p>Due to the complexity of Spatial Index Creation, FME allows one to specify a SQL statement to create an index on a specific spatial column in a MSSQL Spatial table. This SQL is honored on table creation, and is executed immediately after the table is created.</p> <p>In this example, a default spatial index is created on geometry column "a" of table "my_table". A Bounding Box must be specified for every spatial index, making this a minimal spatial index creation statement.</p> <pre>MSSQL_SPATIAL_PROVIDER_INDEX_SQL "CREATE SPATIAL INDEX SIDX_a ON my_table(a) WITH (BOUNDING_BOX = (-200, -200, 200, 200))"</pre> <p>In this example, a spatial index is created on geography spatial column "b" of "my_table". If an existing index named SIDX_b exists, it will be dropped and replaced by this one.</p> <pre>"CREATE SPATIAL INDEX [SIDX_b] ON my_table(b) USING GEOGRAPHY_GRID WITH (GRIDS = (LOW, MEDIUM, HIGH, HIGH), CELLS_PER_OBJECT = 32, DROP_EXISTING = ON)"</pre> <p>See the MSDN article titled "CREATE SPATIAL INDEX (Transact-SQL)" for more information and examples.</p>

Attribute Types

This section of the `<writerKeyword>_DEF` statement defines the attribute types for a table.

bigint

This type is used to represent 64-bit signed integers.

int

This type is used to represent 32-bit signed integers.

smallint

This type is used to represent 16-bit signed integers.

tinyint

This type is used to represent numbers between 0 and 255.

bit

This type is used to represent an integer with a value of 1 or 0

decimal

This type is used to represent fixed precision and scale numeric data from $-10^{38}+1$ to $10^{38}+1$.

numeric

This type is used to represent fixed precision and scale numeric data from $-10^{38}+1$ to $10^{38}+1$.

money

This type is used to represent monetary data values from -2^{63} to $2^{63}-1$. Attribute values are real numbers such as 55.2354.

smallmoney

This type is used to represent monetary data values from -214748.3648 to 214748.3647. Attribute values are real numbers such as 55.2354.

float

This type is used to represent 32-bit floating precision numbers.

real

This type is used to represent 16-bit floating precision numbers.

date

This type is used to represent date data from January 1, 0001 to December 31, 9999. For example, a value of 20061231 represents December 31, 2006.

time

This type is used to represent time data with an accuracy of 100 nanoseconds. For example, a value of 235959.0000000 represents 11:59:59PM.

datetime2

This type is used to represent date and time data from January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds. For example, a value of 20061231235959.0000000 represents 11:59:59PM on December 31, 2006.

datetime

This type is used to represent date and time data from January 1, 1753 to December 31, 9999 with an accuracy of 3.33 milliseconds. For example, a value of 20061231235959 represents 11:59:59PM on December 31, 2006. When writing to the database, the writer expects the date attribute to be in the form YYYYMMDDHHMMSS.

smalldatetime

This type is used to represent date and time data from January 1, 1900 to June 6, 2079 with an accuracy of one minute. For example, a value of 20060101101000 represents 10:10:00AM on January 1, 2006. When writing to the database, the writer expects the date attribute to be in the form YYYYMMDDHHMMSS.

char

This type is used to represent fixed length character data up to a length of 8000 characters.

varchar

This type is used to represent variable length character data up to a length of 8000 characters.

text

This type is used to represent variable length character data up to a length of $2^{31}-1$ characters.

nchar

This type is used to represent fixed length character data up to a length of 4000 characters.

nvarchar

This type is used to represent variable length character data up to a length of 4000 characters.

binary

This type is used to represent fixed length binary data up to a length of 8000 bytes.

varbinary

This type is used to represent variable length binary data up to a length of 8000 bytes.

image

This type is used to represent variable length binary data up to a length of $2^{31}-1$ bytes.

uniqueidentifier

Uniqueidentifier is used to represent GUIDs. As such, it must be set up like a valid GUID. Example: {B85E62C3-DC56-40C0-852A-49F759AC68FB}

Note: the {} must be present for the GUID to be written successfully.

identity

This type is used to represent an auto-incrementing integer field.

If you try to write a value to it; the value will be ignored and the database will simply take the largest integer in the column, increment it and put that number into the field.

Feature Representation

Features are written to the database table with the same name as their feature type, and include geometry as well as attributes as defined on the **DEF** line.

Northgate StruMap Reader/Writer

The StruMap® reader and writer modules allow the Feature Manipulation Engine (FME) to read and write Northgate Information Solutions StruMap SGF/SGX format files. StruMap SGF/SGX files use a published ASCII format.

Overview

The StruMap reader and writer support the storage of point, line, polygon, and text geometric data in a single file. The names of the user-defined attributes are not stored with each feature, but rather in the file header where the structure of the features are defined.

FME considers a StruMap dataset to be a single file.

StruMap Quick Facts

Format Type Identifier	STRUMAP
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	File for Both
Feature Type	Feature Code
Typical File Extensions	.sgf
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	Yes
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	strumap_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	yes
donut polygon	no		solid	no
elliptical arc	no		surface	no
ellipses	no		text	yes
line	yes		z values	no
none	no			

Reader Overview

The StruMap reader extracts features from the input file individually and passes them on to the rest of the FME for further processing. An important thing to note in the StruMap input file is that text components or child can be attached to any geometric types, whereas line symbols can only be attached to lines. The StruMap reader will not output these as a single aggregate feature, but rather as a series of features sharing the same `strumap_id`. Additionally, a `strumap_child_id` is added to each child to identify its order.

Reader Directives

The directives listed below are processed by the StruMap reader. The suffixes listed are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the StruMap reader is `STRUMAP`.

DATASET

Required/Optional: *Required*

The value for this keyword is the directory containing the StruMap files to be read. A typical mapping file fragment specifying an input StruMap dataset looks like:

Example:

```
STRUMAP_DATASET /usr/data/strumap/input.sgf
```

Workbench Parameter: *Source Northgate StruMap File(s)*

DEF

Required/Optional: *Required*

Each StruMap feature must be defined before it can be read. The definition specifies the feature code of the feature, and the names and the types of all attributes. The syntax of a StruMap `DEF` line is:

```
<ReaderKeyword>_DEF <baseName> \  
  [<attrName> <attrType>]+
```

The following table shows the attribute types supported.

Field Type	Description
char(<width>)	Character fields store fixed length strings. The width parameter controls the maximum number of characters that can be stored by the field. No padding is required for strings shorter than this width.
date	Date fields store date as character strings with the format <code>YYYYMMDD</code> .
double	Float fields store 64-bit floating point values. There is no ability to specify the precision and width of the field.
integer	Integer fields store 32-bit signed integers.
logical	Logical fields store <code>TRUE/FALSE</code> data. Data read or written from and to such fields must always have a value of either true or false.

IDs

Required/Optional: *Optional*

This optional specification limits StruMap features read. If no **IDs** are specified, then all StruMap features in the input file are read.

The syntax of the **IDs** keyword is:

```
<ReaderKeyword>_IDS <baseName> \  
    <baseName1> \  
    <baseNameN>
```

The basenames must match those used in **DEF** lines.

The example below selects only the **roads** StruMap feature for input during a translation:

```
STRUMAP_IDS roads
```

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the **mitab.dll** in the FME home directory to **mapinfo.dll**.

The syntax of the **MAPINFO_SEARCH_ENVELOPE** directive is:

```
MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

```
MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The **COORDINATE_SYSTEM** directive, which specifies the coordinate system associated with the data to be read, must always be set if the **SEARCH_ENVELOPE_COORDINATE_SYSTEM** directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the **SEARCH_ENVELOPE_COORDINATE_SYSTEM** to the reader **COORDINATE_SYSTEM** prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

Writer Overview

The StruMap writer creates and writes all features it is given to the file specified by the DATASET keyword. Any existing StruMap files in the directory that have the same name as the output dataset will be overwritten with the new data.

Multiple geometric types can be stored in each StruMap file. Any feature whose feature type is not specified in the mapping file will not be written to file. If no **strumap_id** is present, a feature is considered a single feature in the output file (as opposed to being part of an aggregate). Features that share the same ID, which pass through the writer one after another (grouped by ID prior to entering the writer), are considered a single feature when output to file. It is an additional requirement that components of a parent feature pass through the writer before their children. Otherwise, the parent and child relationship may not be realized.

The JOIN directive is not supported by the writer; in other words, features must carry all the attribution before they are passed into the writer.

Writer Directives

The directives processed by the StruMap writer are listed below. The suffixes shown are prefixed by the current <WriterKeyword> in a mapping file. By default, the <WriterKeyword> for the StruMap writer is STRUMAP.

DATASET

Required/Optional: *Required*

This specifies the output file to which the features are to be written. A typical mapping file fragment specifying an output StruMap dataset looks like:

```
STRUMAP_DATASET /usr/data/Strumap/output.sgf
```

Workbench Parameter: *Destination Northgate StruMap File*

DEF

Required/Optional: *Required*

Each StruMap feature type must be defined before it can be written. The definition specifies a unique feature code of the feature type, and the names and the types of all attributes. The syntax of a StruMap DEF line is:

```
<WriterKeyword>_DEF <baseName> \  
    [<attrName> <attrType>]+
```

The attribute types supported are the same as those listed in the Reader section.

STRING_DELIMITER

Required/Optional: *Optional*

This is the character added around a string to indicate the beginning and the ending of a string. For example, to indicate that a pair character “\” should be used to quote a string, the following is used. By default, double quotation marks are used.

```
<writerKeyword>_STRING_DELIMITER “\”
```

Feature Representation

StruMap features consist of geometry and attributes. In particular, all StruMap features contain a `strumap_type` attribute, which identifies the geometric type of the feature.

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

There are five types defined for the `strumap_type` attribute:

Attribute Name	Contents
<code>strumap_type</code>	<p>The StruMap geometric type of this feature.</p> <p>Range: <code>strumap_point</code> <code>strumap_line</code> <code>strumap_line_symbol</code> <code>strumap_polygon</code> <code>strumap_displayed_text</code></p> <p>Default: No default</p> <p>Note: <code>strumap_displayed_text</code> is used in place of both <code>strumap_text</code> and <code>strumap_ftext</code> because the latter types are not supported by the Writer.</p>
<code>strumap_id</code>	<p>The StruMap ID number of the feature. This helps the writer identify the components of each feature. Features which are meant to be part of an aggregate feature should share the same ID. For the reader, this is automatically set. For the writer, if this is not specified, then each feature is considered a single output feature when written to file.</p> <p>Range: Positive Integers</p> <p>Default: No default</p>
<code>strumap_child_id</code>	<p>The StruMap geometric type of this feature.</p> <p>Range: <code>strumap_point</code> <code>strumap_line</code> <code>strumap_line_symbol</code> <code>strumap_polygon</code> <code>strumap_displayed_text</code></p> <p>Default: No default</p>

Points

strumap_type: strumap_point

This indicates that the feature is a StruMap point. Additional attributes include:

Attribute Name	Contents
strumap_angle	The angle of the rotated symbol. Range: Real Default: No default

strumap_type: strumap_line_symbol

This indicates that the feature is a StruMap line symbol. This is actually a component of a StruMap line and cannot be written to file on its own in terms for the StruMap definition; however, it can be represented by other formats.

In addition to the geometry are the following attributes:

Attribute Name	Contents
strumap_linesym_num	The line symbol number of the line symbol. Range: integers greater or equal to 0 Default: No default
strumap_linesym_angle	The rotation of the line symbol. Range: Any real number Default: No default
strumap_linesym_scale	The scale factor of the line symbol. Range: Any real number Default: No default

Lines

strumap_type: strumap_line

This indicates that the feature is a StruMap line. In addition to the geometry are the following attributes:

Attribute Name	Contents
strumap_line_style	The line style of the line. Range: integers greater than or equal to 0 Default: No default
strumap_red	The red intensity of the line. This must be used together with strumap_green and strumap_blue in order to be used by the StruMap writer. Range: Integer from 1 to 255 Default: No default
strumap_green	The green intensity of the line. Range: Integer from 1 to 255 Default: No default
strumap_blue	The blue intensity of the line.

Attribute Name	Contents
	<p>Range: Integer from 1 to 255 Default: No default</p>
strumap_flowdir_position	<p>The position of the flow direction. This must be used with the <code>strumap_flowdir_easting</code> and <code>strumap_flowdir_northing</code> attributes.</p> <p>Range: 0 (not set) 1 (flow from point 1 to point 2) 2 (flow from point 2 to point 1) 3 (bidirectional flow) 4 (blocked)</p> <p>Default: No default</p>
strumap_flowdir_x1	<p>The <code>x</code> coordinate for the first point in terms for the <code>strumap_flowdir_position</code>.</p> <p>Range: Real Default: No default</p>
strumap_flowdir_y1	<p>The <code>y</code> coordinate for the first point in terms for the <code>strumap_flowdir_position</code>.</p> <p>Range: Real Default: No default</p>
strumap_flowdir_x2	<p>The <code>x</code> coordinate for the second point in terms for the <code>strumap_flowdir_position</code>.</p> <p>Range: Real Default: No default</p>
strumap_flowdir_y2	<p>The <code>y</code> coordinate for the second point in terms for the <code>strumap_flowdir_position</code>.</p> <p>Range: Real Default: No default</p>
strumap_mask_x{<number>}	<p>The <code>x</code> coordinate of the mask. <code><number></code> is a positive integer used to indicate the order of the mask. Hence, in order for this to be valid, it has to be used in conjunction with <code>strumap_mask_y{<number>}</code> and <code>strumap_mask_length{<number>}</code> where <code><number></code> share the exact same value. As a result of this mechanism, more than one mask can be added to each line feature.</p> <p>Range: Real Default: No default</p>
strumap_mask_y{<number>}	<p>The <code>y</code> coordinate of the mask. See <code>strumap_mask_x</code>.</p> <p>Range: Real</p>

Attribute Name	Contents
	Default: No default
strumap_mask_length {<number>}	The length of the gap. See strumap_mask_x . Range: Real Default: No default

Polygons

strumap_type: strumap_polygon

This indicates that the feature is a StruMap polygon. The first and last coordinates of the polygon must be the same.

Additional attributes include:

Attribute Name	Contents
strumap_seed_x	The x coordinate of the seed for the polygon. Range: Real Default: Defaults to the x coordinate of the first point in the polygon.
strumap_seed_y	The y coordinate of the seed for the polygon. Range: Real Default: Defaults to the y coordinate of the first point in the polygon.

Text

strumap_type: strumap_displayed_text

StruMap displayed text features are used to specify annotation information. Each text feature has a location defined by a single point geometry, and can have its text string, style, justification, and rotation angle set independently.

The following table lists the special FME attribute names used to control the [Strumap_display_text](#) settings.

Attribute Name	Contents
strumap_attr_code	A text string representing the attribute short code as defined in the rule file.
strumap_attr_value	A text string containing the value of the attribute.
strumap_height	The height of the strumap_attr_value in ground units.
strumap_width	The width of the strumap_attr_value in ground units.
strumap_red	The red color component of the strumap_attr_value.
strumap_green	The green color component of the strumap_attr_value.
strumap_blue	The blue color component of the strumap_attr_value.
strumap_angle	The angle at which the strumap_attr_value is displayed.
strumap_position	The justification of the strumap_attr_value. Range:

Attribute Name	Contents
	0 (not set) 1 (bottom left) 2 (center left) 3 (top left) 4 (bottom center) 5 (center center/ original position) 6 (top center) 7 (bottom right) 8 (center right) 9 (top right) Default: No default
strumap_min_length	The minimum length of span on which to display bubble. Range: Positive real numbers Default: 0
strumap_circle	Indicates that an ellipse should be drawn instead of a box. Range: true Default: false
strumap_box	Indicates that a box should be drawn. Range: true Default: false
strumap_solid	Indicates that the ellipse or the box should be filled. Range: true Default: false
strumap_line	Indicates that a line should be drawn from the text to the item. Range: true Default: false
strumap_arrow	Indicates that an arrow should be drawn on a line. Range: true Default: false

NULL (Nothing) Reader/Writer

Overview

The NULL format is useful only for testing purposes.

As a reader, it returns no (0) features.

As a writer, any feature written to it is deleted, and it produces no output.

ODBC 3.x Reader

Format Notes: This format is not supported by FME Base Edition.

Overview

The ODBC 3.x reader enables FME to access live databases accessible via Open DataBase Connectivity (ODBC) (ODBC driver version 3 or better).

This reader is mainly intended for FoxPro databases as the commonly available FoxPro ODBC driver only supports ODBC specifications version 3.x.

Tip: See the @SQL function in the FME Functions and Factories manual. This function allows arbitrary Structured Query Language (SQL) statements to be executed against any database.

ODBC 3.x Quick Facts

Format Type Identifier	ODBC2
Reader/Writer	Reader
Licensing Level	Professional
Dependencies	None
Dataset Type	ODBC Datasource name
Feature Type	Table name
Typical File Extensions	N/A
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	Yes
Geometry Type	db_type

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	point	no
circles	no	polygon	no
circular arc	no	raster	no
donut polygon	no	solid	no
elliptical arc	no	surface	no
ellipses	no	text	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
line	no		z values	n/a
none	yes			

Reader Overview

FME considers a database dataset to be a collection of relational tables. The tables must be defined in the mapping file before they can be read. Arbitrary WHERE clauses and joins are fully supported.

Reader Directives

The directives that are processed by the ODBC 3.x reader are listed below. The suffixes shown are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the ODBC 3.x reader is `ODBC2`.

DATASET

Required/Optional: *Required*

Example:

```
ODBC2_DATASET citySource
```

Workbench Parameter: *Source ODBC 3.x Dataset*

USER_NAME

Required/Optional: *Optional*

The name of user who will access the database. For some database types, this is ignored.

```
ODBC2_USER_NAME bond007
```

Workbench Parameter: *User ID*

PASSWORD

Required/Optional: *Optional*

The password of the user accessing the database. For some database types, this is ignored.

```
ODBC2_PASSWORD moneypenny
```

Workbench Parameter: *Password*

DEF

Required/Optional: *Required*

Each database table must be defined before it can be read. The definition may take two forms:

The syntax of the first form is:

```
ODBC2_DEF <tableName> \
    [odbc2_where_clause <whereClause>] \
    [odbc2_sql <sql statement>] \
    [<fieldName> <fieldType>] +
```

In this form, the fields and their types are listed. The `<fieldType>` of each field must be given, but it is not verified against the database definition for the field. In effect, it is ignored.

The `<tableName>` must match a table in the database. This will be used as the feature type of all the features read from the table.

If no `<whereClause>` is specified, all rows in the table will be read and returned as individual features, unless limited by a global directive:

```
<ReaderKeyword>_WHERE_CLAUSE
```

If a `<whereClause>` is specified, only those rows that are selected by the clause will be read. Note that the `<whereClause>` does not include the word "WHERE."

In this example, the all records whose ID is less than 5 will be read from the supplier table:

```
ODBC2_DEF supplier \  
  odbc2_where_clause "id < 5" \  
  ID integer \  
  NAME char(100) \  
  CITY char(50)
```

The syntax of the second form is:

```
ODBC2_DEF <tableName> \  
  odbc2_sql <sqlStatement>
```

In this form, an arbitrary complete `<sqlStatement>` will be executed. The statement is passed untouched to the database (and therefore may include non-portable database constructions). The results of the statement will be returned, one row at a time, as features to FME. This form allows the results of complex joins to be returned to FME.

All features will be given the feature type `<tableName>`, even though they may not necessarily have come from that particular table. Indeed, with this form, the `<tableName>` need not exist as a separate table in the database.

In this example, the results of joining the `employee` and `city` tables are returned. All attributes from the two tables will be present on each returned feature. The feature type will be set to `complex`.

```
ODBC2_DEF complex \  
  odbc2_sql \  
    "SELECT * FROM EMPLOYEE, CITY WHERE EMPLOYEE.CITY = CITY.NAME"
```

WHERE_CLAUSE

Required/Optional: *Optional*

This optional specification is used to limit the rows read by the reader from each table. If a given table has no `odbc2_where_clause` or `odbc2_sql` specified in its `DEF` line, the global `<ReaderKeyword>_WHERECLAUSE` value, if present, will be applied as the `WHERE` specifier of the query used to generate the results. If a table's `DEF` line does contain its own `odbc2_where_clause` or `odbc2_sql`, it will override the global `WHERE` clause.

The syntax for this clause is:

```
ODBC2_WHERECLAUSE <whereClause>
```

Note that the `<whereClause>` does not include the word "WHERE."

The example below selects only the features whose lengths are more than 2000:

```
ODBC2_WHERECLAUSE LENGTH > 2000
```

Workbench Parameter: *Where Clause*

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined database tables files that will be read. If no `IDs` are specified, then all defined and available tables are read. The syntax of the `IDs` keyword is:

```
ODBC2_IDS <featureType1> \  
<featureType2> ... \  
<featureTypeN>
```

The feature types must match those used in **DEF** lines.

The example below selects only the **HISTORY** table for input during a translation:

```
ODBC2_IDS HISTORY
```

PERSISTENT_CONNECTION

A user may want to keep a connection to a database for reuse during a particular FME session. For example, when running a batch of 100 mapping files on the same database connection, it may be desirable to keep a connection open and save the processing time required to make and break a database connection.

A database connection will be determined to be the same when the database name, the username, the password, and the transaction interval are the same.

Values: *YES* | *NO*

Default value: *NO*

Example:

```
ODBC2_PERSISTENT_CONNECTION YES
```

Workbench Parameter: *Persistent Connection*

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

*** Workbench Parameter**

Additional Attributes to Expose

Feature Representation

Features read from a database consist of a series of attribute values. They have no geometry. The attribute names are as defined in the **DEF** line if the first form of the **DEF** line was used. If the second form of the **DEF** line was used, then the attribute names are as they are returned by the query, and as such may have their original table names as qualifiers. The feature type of each Database feature is as defined on its **DEF** line.

Mapping File Example

This example illustrates how the two forms of the **DEF** lines can be used to read from an ODBC database source, which is named **rogers**.

```

READER_TYPE ODBC2
ODBC2_DATASET      rogers

# As we are reading from an MS-ACCESS database, we don't need to
# specify these:
# DATABASE_USER_NAME <userName>
# DATABASE_PASSWORD <password>

# Form 1 of the DEF line is used like this -- it reads just
# the two fields we list and applies the where clause

ODBC2_DEF supplier \
    odbc2_where_clause "id < 5" \
    ID integer \
    CITY char(50)

# Form 2 of the DEF line is used like this -- we let SQL
# figure out what fields we want and do a complex join
# involving 3 tables. The FME features will have whatever
# fields are relevant. The "feature type" as far as
# FME is concerned is whatever was put on the DEF line.
# In this case "complex" is the feature type, even though no
# table named "complex" is present in the database.

ODBC2_DEF complex \
    odbc2_sql "SELECT CUSTOMER.NAME, CUSTOMER.ID,
VIDEOS.ID, VIDEOS.TITLE FROM RENTALS, CUSTOMER,
VIDEOS WHERE RENTALS.customerID = CUSTOMER.ID AND
VIDEOS.ID = RENTALS.videoID AND CUSTOMER.ID = 1"

# Finally, define the NULL writer as our output -- we will
# just log everything we read to the log file for inspection.

WRITER_TYPE NULL
NULL_DATASET null

FACTORY_DEF * SamplingFactory \
    INPUT FEATURE_TYPE * @Log()

```

OpenStreetMap (OSM) XML Reader/Writer

Format Notes: This format is not supported by FME Base Edition.

The OpenStreetMap (OSM) is a collaborative mapping project for creating a free and editable map of the whole world.

This section assumes familiarity with the OSM format. Further information on OSM can be found at <http://www.openstreetmap.org>.

Overview

OpenStreetMap data can be downloaded in a topologically structured XML format. The data primitives in an OSM data file are nodes, ways, and relations.

- A node is a lat/lon pair.
- A way is a list of at least two node references describing a linear feature. Ways can be closed, in which case the first and the last node are identical. Areas are not explicitly represented in OSM but are identified via community-approved tags.
- Relations are a group of zero or more primitives with an associated role. All data in OSM are in the WGS-84 datum.

OSM has no explicit schema (feature type) definitions. Each node, way, and relation can have an arbitrary number of attributes, called tags in OSM. A tag is composed of a key and a value. The OpenStreetMap wiki does define a set of recommend tags that can be used to classify the nodes and ways into higher-level groupings, i.e., feature types.

The FME OSM reader provides some user settings to help influence the classification of the OSM data being read. This is needed for most GIS formats that have explicit schema definitions. The community-defined feature types can be found at http://wiki.openstreetmap.org/index.php/Map_Feature.

OSM Quick Facts

Format Type Identifier	OSM
Reader/Writer	Both
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	
Typical File Extensions	.osm
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Never
Schema Required	No
Transaction Support	No
Geometry Type	xml_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	yes		z values	no
none	yes			

Reader Overview

The OSM reader can interpret OSM XML files in several different ways. The reader may output un-categorized nodes, ways and relations, or it may categorize these data primitives according to the recommended interpretations found in http://wiki.openstreetmap.org/index.php/Map_Features.

The OSM reader provides two options when directed to interpret the OSM data with the OSM community approved feature types. The data can be interpreted with either broad or specific categories.

For example, when set to *specific*, the reader will output **highway_motorway**, **highway_primary**, **highway_unclassified**, etc., as feature types, whereas it will only output a **highway** feature type when its mode is set to *broad*. Note that in either case, a schema scan of the data file being read is performed to determine the possible attribute name/value pairs of each feature type.

With three exceptions the OSM relations are not interpreted. The interpreted relations are the ones that are tagged with a key of **type** and values of either **multipolygon**, **route**, or **restriction**. Both **route** and **restriction** are output as non-geometrical features, and their feature types are set to **relation_route** and **relation_restriction**, respectively. The **multipolygon** relation is output as an area or multi-area geometry; this includes donuts, or an aggregate of polygons and donuts. The **multipolygon** relation may be output as a line or a non-geometrical feature. In the case of bad geometrical information, the **multipolygon** relation feature type will be set to **relation_multipolygon**.

Geometry

The OSM reader supports points, lines, and area geometries. Points are constructed from OSM nodes, linear features from OSM ways, and area features are constructed from either appropriately tagged closed OSM ways, or from OSM multipolygon relations.

Coordinate Systems

The features output by the OSM reader are always in LL84.

FME Feature Attributes

All OSM tags are loaded from the OSM data primitives are loaded as FME feature attributes. Relation feature types will have their members mapped as an FME list attribute in the FME feature. The list attribute will have 3 components:

```
member{}.type
member{}.ref
member{}.role
```

Relation feature types will also have an attribute named as **osm_relation_type** to describe the type of relation.

You can use this member list attribute to further process the relations within FME Workbench.

In OSM way features are constructed by reference to node IDs. These node IDs are mapped into the `nd{}.ref` list attribute in FME way features.

Reader Directives

The suffixes shown are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the OSM reader is `OSM`.

DATASET

The location of the OSM file to be read.

Required/Optional

Required

Mapping File Syntax

```
OSM_DATASET c:\sample.osm
```

* Workbench Parameter

Source OpenStreetMap File(s)

CLOSE_AREAS

Specifies whether the OSM reader should create area geometries for OSM ways, whose first and last point equal, even if their "area" tag is not appropriately set. The valid values for this directive are 'Yes' and 'No', with the default being 'Yes'.

This directive is used when generating workspaces and mapping files. As a result, it is not editable within Workbench after the workspace has been generated.

Required/Optional

Optional

Mapping File Syntax

```
OSM_CLOSE_AREAS No
```

* Workbench Parameter

not applicable

CUSTOM_AREA

This directive relies on a text file whose content lists the way feature types that should be converted into area geometries when their first and last coordinates equal.

The format of the text file is simple: each feature type that should be considered a possible area is listed on a separate line. Two example files can be found under `<FME InstallDir>\xml\osm\AreaFeatureLists\osm_specific_areas.txt` and `<FME InstallDir>\xml\osm\AreaFeatureLists\osm_broad_areas.txt`.

This directive is used when generating workspaces and mapping files. As a result, it is not editable within Workbench after the workspace has been generated.

Required/Optional

Optional

Mapping File Syntax

OSM_CUSTOM_AREA c:\my_osm_areas.txt

* Workbench Parameter

not applicable

SCHEMA_CHOICE

This optional directive specifies whether the OSM primitives should be interpreted via predefined FME factory pipelines that categorize the nodes, ways, and relations according to the recommended interpretations found in http://wiki.openstreetmap.org/index.php/Map_Features, or if a user defined factory pipeline should be used.

This directive is used when generating workspaces and mapping files. As a result, it is not editable within Workbench after the workspace has been generated.

Values

COMMUNITY (default) | CUSTOM

The `COMMUNITY_FEATURE_TYPES` directive should be used in conjunction with this directive when this directive is set to 'COMMUNITY'.

The `CUSTOM_OSM_PIPELINE` directive should be used in conjunction with this directive when this directive is set to 'CUSTOM'.

Required/Optional

Optional

Mapping File Syntax

OSM_SCHEMA_CHOICE CUSTOM

* Workbench Parameter

not applicable

COMMUNITY_FEATURE_TYPES

This optional directive is applicable when the `SCHEMA_CHOICE` is set to `COMMUNITY`, it specifies the predefined FME factory pipeline that categorizes the nodes, ways, and relations into the recommended interpretations found in http://wiki.openstreetmap.org/index.php/Map_Features. The valid values for this directive are 'BROAD', 'SPECIFIC' and 'RAW', with 'BROAD' being the default.

When 'BROAD' is chosen, then the OSM reader will use the "{FME Directory}\xml\osm\schemaMap\osm_broad_schema.fmi" to categorize the OSM data primitives into broader categories. The `osm_broad_schema.fmi` contains a sequence of FME factories that would assign a node, way, or relation a specific feature according to the existence of certain tag keys. For example, any way with a 'highway' tag key regardless of its tag value will be categorized into a highway feature type.

When 'SPECIFIC' is chosen, then the OSM reader will use the "{FME Directory}\xml\osm\schemaMap\osm_specific_schema.fmi" to categorize the OSM data primitives into more detailed categories. The `osm_specific_schema.fmi` pipeline contains a `SchemaMappingFactory` that loads a CSV file, {FME Directory}\xml\osm\schemaMap\osm_specific_schema.fmi, to help it categorize the OSM data primitives according to the tag key and tag value. For example, any OSM way primitive with a 'highway' tag key and tag value of 'primary' will be categorized into a `highway_primary` feature type.

When 'RAW' is chosen, then the OSM reader will leave the OSM nodes, ways, and relations unprocessed.

This directive is used when generating workspaces and mapping files. As a result, it is not editable within Workbench after the workspace has been generated.

Required/Optional

Optional

Mapping File Syntax

```
OSM_COMMUNITY_FEATURE_TYPES SPECIFIC
```

* Workbench Parameter

not applicable

CUSTOM_OSM_PIPELINE

This optional directive is applicable when the SCHEMA_CHOICE is set to CUSTOM. It specifies a user-defined factory pipeline that can be used to transform the OSM nodes, ways and relations into user-defined feature types.

This directive is used when generating workspaces and mapping files. As a result, it is not editable within the Workbench after the workspace has been generated.

Required/Optional

Optional

Mapping File Syntax

```
OSM_CUSTOM_OSM_PIPELINE c:\my_osm_feature_categorization.fmi
```

* Workbench Parameter

not applicable

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the `mitab.dll` in the FME home directory to `mapinfo.dll`.

The syntax of the `MAPINFO_SEARCH_ENVELOPE` directive is:

```
MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

```
MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The `COORDINATE_SYSTEM` directive, which specifies the coordinate system associated with the data to be read, must always be set if the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` to the reader `COORDINATE_SYSTEM` prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The OSM writer will write out features as nodes, ways, and relations into an OSM file. Lines and areas that have more than 2000 points will be split into multiple ways, each having at most 2000 nodes. Duplicated nodes are also handled to make sure no duplicate OSM nodes are written out. Conversion of coordinates into WGS84/LL84 is done automatically into the writer.

OSM Tags Handling

Feature types in OSM writer do not have any predefined schema. User attributes that are specified on the feature type will be used as tags in the OSM elements of the output file, if such attributes exist on the features or geometry traits of the geometry.

Example: A feature type named "amenity" has 3 tags that OSM writer will create if the features have such attributes:

```
OSM_2_DEF amenity \  
amenity xml_char(17) \  
building xml_char(4) \  
source xml_char(19)
```

OSM Elements Writing

There are common attributes on each OSM element that the writer will write out if feature attributes or geometry traits exist: `id`, `uid`, `timestamp`, `visible`, `changeset`, and `version`.

If an OSM node, point or relation does not have an `id` attribute found on geometry or feature, then a negative value will be assigned to each element.

OSM node

All geometries that contain points (including line, polygon, donut or any multi geometry) will have their coordinates written out as OSM nodes.

Tags on the OSM node will only be written out if there are user attributes defined on feature types and such attributes exist on the feature as attributes or point-level geometry as traits.

In the case where there are multiple sources of the same node (for example, from a point and a line), only the first node with positive ID is written out. If there are no nodes with positive ID, then the first node the writer encounters will be written out.

OSM way

All linear geometries will have corresponding OSM way elements written out. The writer does not check for duplicated lines.

OSM relation

A feature is written out as an OSM relation if `osm_relation_type` attribute exist, and the value of such attribute determines the type of relation.

Tags that are associated with the relation must be specified as attributes on the features. Tags for the members (OSM ways and nodes) must be specified as traits on each geometry part of feature.

Multipolygon relation

OSM writer also writes out multipolygon relation from donut or multipolygon geometry, even if `osm_relation_type` attribute does not exist on the feature.

The multipolygon relation that is written out follows the rules and examples described in <http://wiki.openstreetmap.org/wiki/Relation:multipolygon>

Other relation

A feature with null or no geometry can be used to describe an OSM relation by specifying the following attributes:

<code>osm_relation_type</code>	The type of OSM relation to be written out
<code>member{}.ref</code> , <code>member{}.role</code> and <code>member{}.type list</code>	Each attribute describes the reference id, role and type of members in relation. The length of list attributes must be equal; otherwise, no relation will be written out.

A feature with collection geometry can be written out as any relation provided that `osm_relation_type` is specified. All non-null geometry parts will be written out as ways and/or nodes and included as members in the relation.

The attributes of `<member>` element (`role` and `type`) must be specified as traits to the geometry parts. The `ref` attribute of `<member>` is populated from the ID of the ways or nodes that have been written out.

Writer Directives

The suffixes shown are prefixed by the current `<writerKeyword>` in a mapping file. By default, the `<writerKeyword>` for the OSM writer is `OSM`.

NUM_DEC_POINTS

This directive controls the number of decimal points that will be written in the OSM file for the lat/long coordinates of OSM points. If it is not specified, the default is 7 decimal points.

Required/Optional

Optional

Mapping File Syntax

```
OSM_NUM_DEC_POINTS 7
```

* Workbench Parameter

Number of Decimal Points

WRITER_CHARSET

The character set encoding in which the OSM file will be written.

Values

UTF-8 | UTF-16 | UTF-16BE | UTF16-LE | UTF-32 | UTF-32BE | UTF-32LE

If no character set is specified, the OSM file will be written in the UTF-8 character set

Required/Optional

Optional

Mapping File Syntax

```
OSM_WRITER_CHARSET UTF-16
```

* Workbench Parameter

Output Character Set

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Please see the Writer Overview section for details on how the features are converted into OSM elements in the OSM Writer.

Geometry

The geometry features produced by the OSM reader can be identified by the `xml_type` attribute. The valid values for this attribute are:

xml_type	Description
xml_no_geom	FME Feature with no geometry.
xml_point	Point feature.
xml_line	Linear feature.
xml_area	Simple polygon, donut, or an aggregate of polygon and donut features.

No Geometry

xml_type: xml_no_geom

Features with their xml_type attribute set to xml_no_geom do not contain any geometry data.

Points

xml_type: xml_point

Features with their xml_type set to xml_point are single coordinate features or an aggregate of single points.

Lines

xml_type: xml_line

Features with their xml_type set to xml_line are polyline features or an aggregate of polylines.

Areas

xml_type: xml_polygon

Features with their xml_type set to xml_polygon are polygon features which may or may not have interior boundaries, or an aggregate of such polygons.

Oracle Reader/Writer

Format Notes: This format is not supported by FME Base Edition.

Oracle® Version: Any references to Oracle *8i* throughout this chapter are also applicable to Oracle *9i* and Oracle *10g*.

Oracle Instant Client: Instant Client can be used to run your OCI, OCCI, JDBC, and ODBC applications without installing a full Oracle Client. Instant Client supports SQL*Plus.

For more information on how it works with FME, see http://www.f-mepedia.com/index.php/OverviewOracle_Instant_Client.

Overview

The Oracle Reader/Writer enables FME to read and write attribute data stored using Oracle. This module communicates directly with Oracle for maximum throughput.

Oracle Spatial is also supported by FME:

- The object-relational model is documented in the *Oracle Spatial Object Reader/Writer*.
- The relational model is documented in *Oracle Spatial Relational Reader/Writer*.

If only attributes are to be read or written, then this Oracle Database reader and writer module of FME should be used. In addition, an **OracleQueryFactory** is available to extract data from an Oracle database within the FME factory pipeline.

Tip: See the QueryFactory in the FME Functions and Factories manual. This factory also exploits the powerful query capabilities of Oracle Spatial.

See the @SQL function, also in the FME Functions and Factories manual. This function allows arbitrary Structured Query Language (SQL) statements to be executed against any Oracle database.

Oracle Quick Facts

Format Type Identifier	ORACLE8I_DB
Reader/Writer	Both
Licensing Level	Professional
Dependencies	None
Dataset Type	Database
Feature Type	Table name
Typical File Extensions	N/A
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	Yes
Geometry Type	oracle_type

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	point	no
circles	no	polygon	no
circular arc	no	raster	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
donut polygon	no		solid	yes
elliptical arc	no		surface	yes
ellipses	no		text	no
line	no		z values	n/a
none	yes			

Reader Overview

FME considers an Oracle dataset to be a database containing a collection of relational tables. The tables must be defined in the mapping file before they can be read. Arbitrary **WHERE** clauses and joins are fully supported. When using the object-relational model, an entire arbitrary SQL **SELECT** statement may also be used as a source of results.

Reader Directives

The directives listed below are processed by the Oracle Database reader. The suffixes listed are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the Oracle Database reader is **ORACLE8I_DB**.

DATASET

Required/Optional: *Required*

This specifies the SQL/Net service name for the Oracle database, which can be blank to use the default service. If it is specified, then the service must have been set up in the local SQL/Net configuration.

Example:

```
ORACLE8I_DB_DATASET citySource
```

Workbench Parameter: *Source Oracle Non-spatial Service*

USER_NAME

The name of user who will access the database.

Required/Optional

Optional

Mapping File Syntax

```
ORACLE8I_DB_USER_NAME bond007
```

If the database is configured to use an external authentication adapter (such as Windows NT or Kerberos authentication), the username may be left blank, or may be completely omitted.

If a connection cannot be established using the provided username, a second attempt will be made using the upper-case version of the username.

* Workbench Parameter

Username

PASSWORD

Required/Optional: *Required*

The password of the user accessing the database.

```
ORACLE8I_DB_PASSWORD moneypenny
```

If the database is configured to use an external authentication adapter (such as Windows NT or Kerberos authentication), the password may be left blank, or may be completely omitted.

Workbench Parameter: *Password*

WORKSPACE

Required/Optional: *Optional*

The name of the Oracle Workspace Manager workspace which will be used by the reader. All tables read by the reader will be read using the same workspace. If this parameter is omitted, or left blank, the default LIVE workspace will be used.

```
ORACLE8I_DB_WORKSPACE B_focus_1
```

Workbench Parameter: *Oracle Workspace*

DEF

Required/Optional: *Optional*

The syntax of the definition is:

```
ORACLE8I_DB_DEF <tableName> \  
    [oracle_where_clause_encoded <whereClause>] \  
    [oracle_sql_encoded <sqlQuery>] \  
    [oracle_table_writer_mode (inherit_from_writer|insert|update|delete)] \  
    [<fieldName> <fieldType>] +
```

The <fieldType> of each field must be given, but it is not verified against the database definition for the field. In effect, it is ignored.

The <tableName> must match a table in the Oracle database. This will be used as the feature type of all the features read from the table. If the <tableName> does not match a table in the database, a second attempt will be made using the uppercase version of the <tableName>.

The definition allows specification of separate search parameters for each table. If any of the configuration parameters are given, they will override, for that table, whatever global values have been specified by the reader directives listed above. If any of these parameters is not specified, the global values will be used.

The following table summarizes the definition line configuration parameters:

Parameter	Contents
oracle_where_clause_encoded	This specifies the SQL WHERE clause applied to the attributes of the layer's features to limit the set of features returned. If this is not specified, the value of the <ReaderKeyword>_WHERE_CLAUSE directive is used. This parameter is encoded as described in the section <i>Substituting Strings in Mapping Files</i> in FME Fundamentals help > Mapping File Syntax.
oracle_sql_encoded	This specifies an SQL SELECT query to be used as the source for the results. If this is specified, the Oracle Database reader will execute the query, and use the resulting rows as the features instead of reading from the table <layerName>. All returned features

Parameter	Contents
	<p>will have a feature type of <layerName>, and attributes for all columns selected by the query.</p> <p>The oracle_where_clause_encoded are ignored if oracle_sql_encoded is supplied.</p> <p>This parameter is encoded as described in the section <i>Substituting Strings in Mapping Files</i> in FME Fundamentals help > Mapping File Syntax.</p>
oracle_table_writer_mode	<p>The the default operation mode of the feature type in terms of the types of SQL statements sent to the database. Valid values are INSERT, UPDATE, DELETE and INHERIT_FROM_WRITER. Note that INSERT mode allows for only INSERT operations where as UPDATE and DELETE can be overwritten at the feature levels. INHERIT_FROM_WRITER simply indicates to take this value from the writer level and not to override it at the feature type level.</p> <p>Default:INHERIT_FROM_WRITER</p>

If no <whereClause> is specified, all rows in the table will be read and returned as individual features. If a <whereClause> is specified, only those rows which are selected by the clause will be read. Note that the <whereClause> does not include the word **WHERE**.

When using the object model, the FME allows one to use the oracle_sql_encoded parameter to specify an arbitrary SQL SELECT query. If this is specified, the FME will execute the query, and use each row of data returned from the query to define a feature. Each of these features will be given the feature type named in the **DEF** line, and will contain attributes for every column returned by the SELECT. In this case, all **DEF** line parameters regarding a **WHERE** clause are ignored, as it is possible to embed this information directly in the text of the <sqlQuery>.

The following example joins the tables ROADS and ROADNAMES, placing the resulting data into FME features with a feature type of MYROADS. Imagine that ROADS defines some attributes for the roads, and has a numeric field named ID, and that ROADNAMES joins the numeric field ID with character arrays with the roads' names.

```
ORACLE8I_DB_DEF MYROADS \
  oracle_sql "SELECT * FROM ROADS, \
  ROADNAMES WHERE ROADS.ID = ROADNAMES.ID"
```

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined database tables files that will be read. If no **IDs** are specified, then all defined and available tables are read. The syntax of the **IDs** directive is:

```
ORACLE8I_DB_IDS <featureType1> \
  <featureType2> ... \
  <featureTypeN>
```

The feature types must match those used in **DEF** lines.

The example below selects only the **ROADS** table for input during a translation:

```
ORACLE_IDS ROADS
```

WHERE_CLAUSE

Required/Optional: *Optional*

This specifies an SQL **WHERE** clause, which is applied to the table's columns to limit the resulting features. This feature is currently limited to apply only to the attributes of the target table, and does not allow for joining multiple tables together. The effect of table joins can be achieved by specifying the entire queries in the **DEF** line with an **oracle_sql_encoded** parameter.

By default, there is no **WHERE** clause applied to the results, so all features in the table are returned.

CHUNK_SIZE

Required/Optional: *Optional*

The features are read from the Oracle database using a bulk reading technique to maximize performance. Normally 1000 rows of data are read from the database at a time.

This directive allows one to tune the performance of the reader. It specifies how many rows are read from the database at a time.

Workbench Parameter: *Rows to Read at a Time*

BEGIN_SQL{n}

Occasionally you must execute some ad-hoc SQL prior to opening a table. For example, it may be necessary to ensure that a view exists prior to attempting to read from it.

Upon opening a connection to read from a database, the reader looks for the directive **<ReaderKeyword>_BEGIN_SQL{n}** (for **n=0,1,2,...**), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the **FME_SQL_DELIMITER** keyword, embedded at the beginning of the SQL block. The single character following this keyword will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;  
DELETE FROM instructors;  
DELETE FROM people  
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

* Workbench Parameter

SQL Statement to Execute Before Translation

END_SQL{n}

Occasionally you must execute some ad-hoc SQL after closing a set of tables. For example, it may be necessary to clean up a temporary view after writing to the database.

Just before closing a connection on a database, the reader looks for the directive `<ReaderKeyword>_END_SQL{n}` (for $n=0,1,2,\dots$), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the `FME_SQL_DELIMITER` directive, embedded at the beginning of the SQL block. The single character following this directive will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;
DELETE FROM instructors;
DELETE FROM people
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

✳ Workbench Parameter

SQL Statement to Execute After Translation

REMOVE_SCHEMA_QUALIFIER

Required/Optional: *Optional*

Specifies whether to keep or remove the schema qualifier. The full name of a table in an Oracle database is of the format `<schema_name>.<table_name>`. Setting this keyword to **YES** indicates that the reader should return the table name without any prefixes. This is useful when:

- creating a workspace that will be passed on to another organization using the same table names,

When this keyword is set to **YES** during the generation of a mapping file or workspace, the source feature types will be the table names without any prefix; otherwise, they will contain the owner name as a prefix. It is recommended that this keyword not be changed in value after generating the mapping file/workspace as it is possible for no features to be successfully passed onto the writer (since the writer is expecting feature types with different names).

Note that even when `REMOVE_SCHEMA_QUALIFIER` is set to **YES**, if the table is owned by a user other than the current user, the `<owner_name>` prefix will **not** be dropped so that the reader will find the correct table; however, the `<database_name>` prefix will still be dropped.

Value: YES | NO

Default Value: NO

Example:

```
ORACLE8I_DB_REMOVE_SCHEMA_QUALIFIER YES
```

Workbench Parameter: *Remove Schema Qualifier*

USE_UNIFIED_DATE_ATTRS

Required/Optional: *Optional*

Specifies whether we want to use unified date attributes, where the date and time are read into one attribute, or whether we want to use split date attributes, where two attributes are produced, one with only the date and another with both the date and time.

The value of this keyword should not be changed. It is automatically set to YES in new mapping files and workspaces. To maintain backwards compatibility, if this keyword is not present, the reader will behave as though the keyword is set to NO.

Value: YES | NO

Default Value: YES (in new mapping files and workspaces), NO otherwise

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

✳ Workbench Parameter

Additional Attributes to Expose

Writer Overview

The Oracle Database writer module stores attribute data in an Oracle database. Only uppercase table names are supported.

The Oracle Database writer provides the following capabilities:

- **Transaction Support:** The Oracle Database writer provides transaction support that eases the data loading process. Occasionally, a data load operation terminates prematurely due to data difficulties. The transaction support provides a mechanism for reloading corrected data without data loss or duplication.
- **Table Creation:** The Oracle Database writer uses the information within the FME mapping file to automatically create database tables as needed.
- **Table Dropping:** The Oracle Database writer has an option that allows each table to be written to be dropped if recreating, or truncated if appending.
- **Index Creation:** The Oracle Database writer will set up and populate all needed indexes and index tables as part of the loading process.
- **Bulk Loading:** The Oracle Database writer uses a bulk loading technique to ensure speedy data load.

Writer Directives

The directives processed by the Oracle Database writer are listed below. The suffixes shown are prefixed by the current `<writerKeyword>` in a mapping file. By default, the `<writerKeyword>` for the Oracle Database writer is `ORACLE8I_DB`.

DATASET, USER_NAME, WORKSPACE, BEGIN_SQL{ }, and END_SQL{ }

These directives operate in the same manner as they do for the Oracle Database reader.

DEF

Required/Optional: *Required*

Each Oracle Database table must be defined before it can be written. The general form of an Oracle Database definition statement is:

```
ORACLE8I_DB_DEF <tableName> \
  [oracle_sql_encoded <sqlQuery>] \
  [oracle_update_key_columns <column>[,<column>]... \
  [oracle_delete_key_columns <column>[,<column>]... \
  [oracle_drop_table (yes|no)] \
  [oracle_truncate_table (yes|no)] \
  [oracle_params <creationParams>] \
  [oracle_sequenced_cols column[:seqname][;column[:seqname]]...] \
  [<fieldName> <fieldType>]*
```

If the user wishes to create a table, the table definition allows control of the table that will be created. Otherwise, the table definition serves to provide options for inserting, updating or deleting data in an existing table. In each case, some parameters may be unused. The recommended approach is to take advantage of the updated Workbench GUI to limit mistakes when setting the layer parameters.

If the table already exists in the database, then it is not necessary to list the fields and their types – FME will use the schema information in the database to determine this. If the fields and types are listed, they must match those in the database, however, not all fields must be listed.

If the table does not exist, then the field names and types are used to first create the table. In any case, if a **<field-Type>** is given, it must be a field type supported by the target database.

The configuration parameters present on the definition line are described in the following table:

Parameter	Contents
oracle_sql_encoded	<p>This specifies an SQL INSERT or UPDATE query to be used to define the results. If this is specified, the Oracle Database writer will execute the query, defining one row for each feature from the FME.</p> <p>The values in the query are specified by embedding :attrName in the query itself, where attrName is the name of the FME feature's attribute; for example: INSERT INTO EXAMPLE VALUES :a, :b</p> <p>In this example, the attributes named a and b will be taken from each feature written to <tableName>.</p> <p>The attributes named in the query must be listed on the DEF line so that the FME knows what type to use. There is no necessary or implied correlation between the FME attribute name and the Oracle column name. Take, for example, this UPDATE query: UPDATE RR SET TEXTSTRING=:mytext WHERE ID=:myid</p> <p>In this example, the Oracle column named ID is compared to the value of each feature's attribute named myid, and the value of the table's column named TEXTSTRING is set from the feature attribute named mytext.</p> <p>This parameter is encoded as described in the section <i>Substituting Strings in Mapping Files</i> in FME Fundamentals help > Mapping File Syntax.</p>
oracle_params	<p>This specifies additional parameters to be appended to the Oracle CREATE query used to create the output table. It is</p>

Parameter	Contents
	<p>used to specify table allocation characteristics and the like.</p> <p>If this is specified, it will override the global <code>CREATE_TABLE_PARAMS</code> directive.</p>
<code>oracle_update_key_columns</code>	<p>This instructs the Oracle Database writer to perform an <code>UPDATE</code> operation on the table, rather than performing an <code>INSERT</code>. The argument is a comma-separated list of the columns which are matched against the corresponding FME attributes' values to specify which rows are to be updated with the other attribute values.</p> <p>For example: <code>oracle_update_key_columns ID</code></p> <p>would have a similar effect to the "UPDATE" example in the above discussion of the <code>oracle_sql_encoded</code> directive. In this case, however, the FME attribute is always matched against the Oracle column with the same name. Also, the target table is always the feature type specified in the DEF line.</p> <p>Each column listed with the <code>oracle_update_key_columns</code> directive must be defined with a type on the DEF line, in addition to the columns whose values will be updated by the operation.</p>
<code>oracle_delete_key_columns</code>	<p>This instructs the Oracle Database writer to perform an <code>DELETE</code> operation on the table, rather than performing an <code>INSERT</code>. The argument is a comma-separated list of the columns which are matched against the corresponding FME attributes' values to specify which rows are to be updated with the other attribute values.</p> <p>For example: <code>oracle_delete_key_columns ID</code></p> <p>In this case, the FME attribute is always matched against the Oracle column with the same name. Also, the target table is always the feature type specified in the DEF line.</p> <p>Each column listed with the <code>oracle_delete_key_columns</code> directive must be defined with a type on the DEF line.</p>
<code>oracle_drop_table</code>	<p>This specifies whether a table should be dropped, if it exists, before being recreated. If the table does not exist, then the operation is ignored and the user is warned. Note that the drop table option is only available when specifying table creation parameters.</p>
<code>oracle_truncate_table</code>	<p>This specifies whether a table should be truncated, if it exists, before data is inserted. If the table does not exist,</p>

Parameter	Contents
	then the operation is ignored and the user is warned. Note that the truncate table option is only available when not specifying table creation parameters.
oracle_sequenced_cols	<p>Indicates which columns' values come from sequences. The format for this parameter is of the form</p> <pre>oracle_sequenced_cols col- umn1:seqname1;column2:seqname2;...</pre> <p>where "columnN" is the name of the column whose value is provided by the sequence, and "seqnameN" is the name of the sequence providing the value.</p> <p>If ":seqnameN" is not given, the column's value will be provided by a sequence with the same name as the column. Sequence names are case-sensitive. The sequences will be created if they do not already exist, in which case a message will be written to the log file.</p>
oracle_contains_measures	<p>This directs the writer to write measures to the destination table. When this directive is set to yes and the incoming feature does not have any measures, then null values are written. This parameter applies when writing to existing tables.</p> <p>Default is NO.</p>

START_TRANSACTION

Required/Optional: *Optional*

This statement tells the Oracle Database writer module when to start actually writing features into the database. The Oracle Database writer does not write any features until the feature is reached that belongs to **<last successful transaction> + 1**. Specifying a value of zero causes every feature to be output. Normally, the value specified is zero – a non-zero value is only specified when a data load operation is being resumed after failing partway through.

Parameter	Contents
<last successful transaction>	The transaction number of the last successful transaction. When loading data for the first time, set this value to 0 .

Example:

```
ORACLE8I_DB_START_TRANSACTION 0
```

Workbench Parameter: *Transaction to Start Writing At*

TRANSACTION_INTERVAL

Required/Optional: *Optional*

This statement informs the FME about the number of features to be placed in each transaction before a transaction is committed to the database.

If the TRANSACTION_INTERVAL statement is not specified, then a value of 2000 is used as the transaction interval.

Parameter	Contents
<transaction_interval>	The number of features in a single transaction.

Example:

```
ORACLE8I_DB_TRANSACTION_INTERVAL 5000
```

Workbench Parameter: *Features To Write Per Transaction*

CHUNK_SIZE

See the CHUNK_SIZE directive in the Reader Directives section.

BEGIN_SQL{n}

This directive is described in the *Reader Directives* section. In the case of the writer, the statements will be executed only when the first feature actually written to the dataset.

END_SQL{n}

This directive is described in the *Reader Directives* section. In the case of the writer, the statements will be executed only if at least one feature has been written to the dataset.

STRICT_ATTR_CONVERSION

This directive instructs the Oracle writer on how to proceed when a problem arises while converting a value from one of a feature's attributes to an oracle column value. Examples of such problems would be the truncation of a string value to fit into the target character column, an error in converting a non-numeric attribute to write to a numeric column.

In normal operation, the Oracle writer will silently truncate strings which are too long, or null out values which cannot be successfully converted. It can optionally log features which have conversion problems, or drop problem features and write a warning to the log.

Possible values for this directive are summarized in the following table:

Parameter	Contents
NO	Silently ignore conversion errors. (This is the default behaviour.)
YES	Features are dropped from the translation and a warning is written to the log.
WARN	Log any features causing conversion errors, and then continue the translation as usual

Example:

```
ORACLE8I_DB_STRICT_ATTR_CONVERSION WARN
```

Workbench Parameter: *Enforce strict attribute conversion*

WRITER_MODE

Required/Optional: *Optional*

Note: For more information on this directive, see the chapter *Database Writer Mode*.

This directive informs the Oracle Database writer which SQL operations will be performed by default by this writer. This operation can be set to **INSERT**, **UPDATE** or **DELETE**. The default writer level value for this operation can be overwritten at the feature type or table level. The corresponding feature type DEF parameter name is called `oracle_table_writer_mode`. It has the same valid options as the writer level mode and additionally the value **INHERIT_FROM_WRITER** which causes the writer level mode to be inherited by the feature type as the default for features contained in that table.

The operation can be set specifically for individual feature as well. Note that when the writer mode is set to **INSERT** this prevents the mode from being interpreted from individual features and all features are inserted unless otherwise marked as **UPDATE** or **DELETE** features. These are skipped.

If the `WRITER_MODE` statement is not specified, then a value of **INSERT** is given.

Parameter	Contents
<writer_mode>	The type of SQL operation that should be performed by the writer. The valid list of values are below: INSERT UPDATE DELETE Default: INSERT

Example:

```
ORACLE8I_DB_WRITER_MODE INSERT
```

Workbench Parameter: *Writer Mode*

Writer Mode Specification

The Oracle Database writer allows the user to specify a writer mode, which determines what database command should be issued for each feature received. Valid writer modes are **INSERT**, **UPDATE** and **DELETE**.

Writer Modes

In **INSERT** mode, the attribute values of each received feature are written as a new database record.

In **UPDATE** mode, the attribute values of each received feature are used to update existing records in the database. The records which are updated are determined via the `oracle_update_key_columns` DEF line parameter, or via the `fme_where` attribute on the feature.

In **DELETE** mode, existing database records are deleted according to the information specified in the received feature. Records are selected for deletion using the same technique as records are selected for updating in **UPDATE** mode.

Writer Mode Constraints

In **UPDATE** and **DELETE** mode, the `fme_where` attribute always takes precedence over the `oracle_update_key_columns` DEF line parameter. If both the `fme_where` attribute and the `oracle_update_key_columns` DEF line parameter are not present, then **UPDATE** or **DELETE** mode will generate an error.

When the `fme_where` attribute is present, it is used verbatim as the WHERE clause on the generated **UPDATE** or **DELETE** command. For example, if `fme_where` were set to `'id<5'`, then all database records with field `id` less than 5 will be affected by the command.

When the `fme_where` attribute is not present, the writer looks for the `oracle_update_key_columns` DEF line parameter and uses it to determine which records should be affected by the command. Please refer to DEF on page ??? for more information about the `oracle_update_key_columns` DEF line parameter.

Writer Mode Selection

The writer mode can be specified at three unique levels: on the writer level, on the feature type, or on individual features.

At the writer level, the writer mode is specified by the `WRITER_MODE` keyword. This keyword can be superseded by the feature type writer mode specification. **Note:** For more information on this directive, see the chapter *Database Writer Mode*.

At the feature type level, the writer mode is specified by the `oracle_writer_mode` DEF line parameter. This parameter supersedes the `WRITER_MODE` keyword. Unless this parameter is set to `INSERT`, it may be superseded on individual features by the `fme_db_operation` attribute. Please refer to the DEF line documentation for more information about this parameter.

At the feature level, the writer mode is specified by the `fme_db_operation` attribute. Unless the parameter at the feature type level is set to `INSERT`, the writer mode specified by this attribute always supersedes all other values. Accepted values for the `fme_db_operation` attribute are `INSERT`, `UPDATE` or `DELETE`.

Feature Representation

Features read from Oracle Databases consist of a series of attribute values. They have no geometry. The feature type of each Database feature is as defined on its `DEF` line.

Features written to the database have the destination table as their feature type, and attributes as defined by on the `DEF` line.

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), the Oracle Database module makes use of the following special attribute names:

Attribute Name	Contents
<code>oracle_type</code>	The type of geometric entity stored within the feature. This is always set to: <code>oracle_ni1</code>

Features read from, or written to, Oracle Databases have an attribute for each column in the database table. The feature attribute name will be the same as the source or destination column name. The attribute and column names are case-sensitive.

Troubleshooting

Problems sometimes arise when attempting to connect to an Oracle database. This is almost always due to a misconfiguration in the user's environment. The following suggestions can often help detect and overcome such problems.

- Ensure you can connect to the database with the service name, user name, and password using `SQL*Plus`.
- Ensure that you have the correct version of the Oracle client software installed. Oracle 8.1.5 or newer is recommended. Note that many clients have had problems if they have both 8.0.4 and 8.1.x installed on the same computer.
- Ensure that your `ORACLE_HOME` environment variable is correctly set: see the Oracle documentation for details. This is required for some specific versions of Oracle 8i, and may be required even if `SQL*Plus` appears to operate correctly without it.
- If you have had older versions of the Oracle client software installed, make sure that your `PATH` variable has the current version's Oracle directory **first**, before any other Oracle software, including the WebDB package.

- It is sometimes helpful to define an environment variable named **ORACLE**, with the same value as the **ORACLE_HOME** variable. With some installations, it often helps to ensure that the variable named **ORACLE** is *not* defined.
- When running on UNIX, the following environment variables should be defined:

Variable	Contents	Sample Value
ORACLE_BASE	Top level of directory into which Oracle client software is installed.	/opt2/oracle8i/app/oracle
ORACLE_HOME	The Oracle product directory.	/opt2/oracle8i/app/oracle/product.8.1.5
ORACLE_SID	The system ID for the host's database instance.	FME
LD_LIBRARY_PATH	A list of directories which will be searched for shared objects. This list must include the FME_HOME path, as well as the lib sub-directory of ORACLE_HOME .	\${LD_LIBRARY_PATH}:\${FME_HOME}:- \${ORACLE_HOME}/lib

- In most cases, the **ORACLE_SERVER_NAME** and **ORACLE_DATABASE** directives should be left with blank values, with the **ORACLE_DATASET** directive containing the Oracle service name of the database.

Oracle Spatial Object Reader/Writer

Format Notes: This format is not supported by FME Base Edition.

Object Writing

Object writing is available only with these FME Editions: Oracle, DB2, Smallworld, and Server.

Raster Support

- Raster writing is a beta format.
- Raster writing is available only with FME Oracle Edition.
- Raster reading is a beta format.
- Raster data is supported only for Oracle 10g *and above*.

3D Geometry Support

Reading and writing of 3D surfaces and solids is available only for Oracle 11g.

Oracle® Version

Any references to Oracle 8i throughout this chapter are also applicable to Oracle 9i and Oracle 10g.

Oracle Instant Client

Instant Client can be used to run your OCI, OCCI, JDBC, and ODBC applications without installing a full Oracle Client. Instant Client supports SQL*Plus.

For more information on how it works with FME, see http://www.fmepedia.com/index.php/Oracle_Instant_Client.

Overview

The Oracle Spatial Reader/Writer module enables FME to read and write geometric, raster and attribute data stored using Oracle Spatial. This module communicates directly with Oracle Spatial for maximum throughput. Both the relational and object-relational models of Oracle Spatial are supported by FME: the object-relational model is discussed here and the relational model is discussed in a separate chapter – *Oracle Spatial Relational Reader/Writer*. The object-relational model is often referred to as simply the *object model*, to prevent confusion with the pure relational model.

Raster data, both reading and writing, is supported for Oracle Spatial Databases version 10g and above. Raster reading and writing is identified by a unique reader and writer option but is covered in this documentation chapter.

If only attributes are to be read or written, then the Database reader and writer module of FME should be used. In addition, an OracleQueryFactory is available to extract data from an Oracle Spatial database within the FME factory pipeline.

This section assumes familiarity with Oracle Spatial, the geometry types it supports, its indexing mechanisms and raster components.

Tip:

See the QueryFactory in the FME Functions and Factories manual. This factory also exploits the powerful query capabilities of Oracle Spatial.

See the @SQL function, also in the FME Functions and Factories manual. This function allows arbitrary Structured Query Language (SQL) statements to be executed against any Oracle database.

Oracle Spatial Object Quick Facts

Format Type Identifier	ORACLE8I ORACLERASTER
Reader/Writer	Both
Licensing Level	See Format Notes
Dependencies	See Format Notes
Dataset Type	Database
Feature Type	Table name
Typical File Extensions	N/A
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Always
Schema Required	Yes
Transaction Support	Yes
Enhanced Geometry	Yes
Geometry Type	oracle_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	yes		polygon	yes
circular arc	yes		raster	yes
donut polygon	yes		solid	yes
elliptical arc	no		surface	yes
ellipses	no		text	no
line	yes		z values	yes
none	yes			

Raster-Specific Quick Facts

Band Interpretations	Red8, Red16, Green8, Green16, Blue8, Blue16, Alpha8, Alpha16, Gray8, Gray16, Int8, UInt8, Int16, UInt16, Int32, UInt32, Real32, Real64
Palette Key Interpretations	UInt32
Palette Value Interpretations	Gray8, RGB24, RGBA32
Nodata Value	Any, but all bands on a raster must have the same value.
Cell Origin (x, y)	(0.5, 0.5) or (0.0, 1.0)
Pyramid Generation	yes
Native Compression	yes
Rotation Support	yes
GCP Support	yes
World File Support	no
TAB File Support	no

Reader Overview

The FME considers an Oracle Spatial dataset to be a database containing a collection of relational tables together with their geometry or GeoRaster columns. The tables must be defined in the mapping file before they can be read. Arbitrary **WHERE** clauses and joins are fully supported. When using the object-relational model, an entire arbitrary SQL **SELECT** statement may also be used as a source of results.

The reader is also capable of reading the GeoRaster column along with all the related metadata tables. Note that if the raster image is compressed then the reader ignores compression and always reads it as uncompressed data.

When reading 3D spatial data, it is important to set the `READ_3D_POLYGON_AS_FACE` directive to "YES". This directive is described in greater detail in the following section.

Reader Directives

The directives listed below are processed by the Oracle Spatial reader. The suffixes listed are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the Oracle Spatial reader is `ORACLE8I`.

DATASET

Required/Optional: *Required*

This specifies the SQL/Net service name for the Oracle Spatial database, which can be blank to use the default service. If it is specified, then the service must have been set up in the local SQL/Net configuration.

Example:

```
ORACLE_DATASET citySource
```

Workbench Parameter: *Source Oracle Spatial Object Service*

USER_NAME

The name of user who will access the database.

Required/Optional

Optional

Mapping File Syntax

```
ORACLE_USER_NAME bond007
```

If the database is configured to use an external authentication adapter (such as Windows NT or Kerberos authentication), the username may be left blank, or may be completely omitted.

If a connection cannot be established using the provided username, a second attempt will be made using the uppercase version of the username.

* Workbench Parameter

User ID

PASSWORD

Required/Optional: *Required*

The password of the user accessing the database.

```
ORACLE_PASSWORD moneypenny
```

If the database is configured to use an external authentication adapter (such as Windows NT or Kerberos authentication), the password may be left blank, or may be completely omitted.

Workbench Parameter: *Password*

WORKSPACE

Required/Optional: *Optional*

The name of the Oracle Workspace Manager workspace which will be used by the reader. All tables read by the reader will be read using the same workspace. If this parameter is omitted, or left blank, the default LIVE workspace will be used.

```
ORACLE8I_DB_WORKSPACE B_focus_1
```

Workbench Parameter: *Oracle Workspace*

DEF

Required/Optional: *Optional*

The syntax of the definition is:

```
ORACLE_DEF <tableName> \  
    [oracle_envelope_min_x <xmin>] \  
    [oracle_envelope_min_y <ymin>] \  
    [oracle_envelope_max_x <xmax>] \  
    [oracle_envelope_max_y <ymax>] \  
    [oracle_interaction <interactionType>] \  
    [oracle_interaction_result <interactionQualifier>] \  
    [oracle_where_clause_encoded <whereClause>] \  
    [oracle_sql_encoded <sqlQuery>] \  
    [oracle_dim <dim>] \  
    [oracle_mapinfo_symbology_style_column <columnName>] \  
    [<fieldName> <fieldType>] +
```

The <fieldType> of each field must be given, but it is not verified against the database definition for the field. In effect, it is ignored.

The exception to this is the geometry field type, which is used to specify a geometry column. When reading from a table with multiple geometry columns, the FME normally selects one of the columns arbitrarily to define the geometry of the resulting features; when the DEF line specifies a single field with a type of geometry, the specified column will be used to define the geometry of the features.

The <tableName> must match a table in the Oracle database. This will be used as the feature type of all the features read from the table. If the <tableName> does not match a table in the database, a second attempt will be made using the uppercase version of the <tableName>.

The definition allows specification of separate search parameters for each table. If any of the configuration parameters are given, they will override, for that table, whatever global values have been specified by the reader directives listed above. If any of these parameters is not specified, the global values will be used.

The following table summarizes the definition line configuration parameters:

Parameter	Contents
oracle_envelope _minx oracle_envelope _miny oracle_envelope _maxx oracle_envelope _maxy	These specify the spatial extent of the features to be read from the layer. If these are not all specified, the values from the <ReaderKeyword>_SEARCH_ENVELOPE directive are used. (Note that when this directive is set to yes for the raster reader, then it always clips to the envelope specified).
oracle_interaction	This specifies the spatial interaction type to be tested for this layer. If this is not specified, the value of the <ReaderKeyword>_INTERACTION directive is used.
oracle_interaction _result	This specifies the required result of the spatial interaction comparison performed for this layer. If this is not specified, the value of the <ReaderKeyword>_INTERACTION_RESULT directive is used.
oracle_where_clause_encoded	This specifies the SQL WHERE clause applied to the attributes of the layer's features to limit the set of features returned. If this is not specified, the value of the <ReaderKeyword>_WHERE_CLAUSE directive is

Parameter	Contents
	<p>used.</p> <p>This parameter is encoded as described in the section <i>Substituting Strings in Mapping Files</i> in FME Fundamentals help > Mapping File Syntax.</p>
oracle_sql_encoded	<p>This specifies an SQL SELECT query to be used as the source for the results. If this is specified, the Oracle Spatial reader will execute the query, and use the resulting rows as the features instead of reading from the table <layerName>. All returned features will have a feature type of <layerName>, and attributes for all columns selected by the query.</p> <p>The oracle_where_clause_encoded and all parameters which specify a spatial constraint – oracle_envelope_minx, oracle_interaction, and so on – are ignored if oracle_sql_encoded is supplied.</p> <p>If the object model is being used, and the SELECT statement returns a column containing geometry, the oracle_dim parameter must be used to specify the dimension of the geometry. This is necessary because Oracle 8.1.5 does not provide a way for the FME to determine the dimension of an arbitrary column without knowing the exact name of the originating column and its source table.</p> <p>This parameter is encoded as described in the section <i>Substituting Strings in Mapping Files</i> in FME Fundamentals help > Mapping File Syntax</p>
oracle_dim	<p>This specifies the number of dimensions (2 or 3) for the table's geometry. This is required only if the oracle_sql_encoded parameter is used to specify a SELECT query.</p>
oracle_mapinfo_symbology_style_column	<p>This specifies the name of the column which contains style information for MapInfo symbology. When set, the read feature will contain a set of MIF format attributes and FME generic attributes which correspond to the symbology information parsed from this field. If this parameter is not set, no symbology is read.</p>

If no whereClause is specified, all rows in the table will be read and returned as individual features. If a whereClause is specified, only those rows which are selected by the clause will be read. Note that the whereClause does not include the word *WHERE*.

When using the object model, FME allows you to use the oracle_sql_encoded parameter to specify an arbitrary SQL SELECT query. If this is specified, FME will execute the query, and use each row of data returned from the query to define a feature. Each of these features will be given the feature type named in the DEF line, and will contain

attributes for every column returned by the SELECT. In this case, all DEF line parameters regarding a WHERE clause or spatial querying is ignored, as it is possible to embed this information directly in the text of the <sqlQuery>.

The following example joins the tables ROADS and ROADNAMES, placing the resulting data into FME features with a feature type of MYROADS. Imagine that ROADS defines the geometry for the roads, and has a numeric field named ID, and that ROADNAMES joins the numeric field ID with character arrays with the roads' names.

```
ORACLE8I_DEF MYROADS \
  oracle_dim 2 \
  oracle_sql "SELECT * FROM ROADS, \
    ROADNAMES WHERE ROADS.ID = ROADNAMES.ID"
```

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined database tables files that will be read. If no **IDs** are specified, then all defined and available tables are read. The syntax of the **IDs** directive is:

```
ORACLE_IDS <featureType1> \
  <featureType2> ... \
  <featureTypeN>
```

The feature types must match those used in **DEF** lines.

The example below selects only the **ROADS** table for input during a translation:

```
ORACLE_IDS ROADS
```

Workbench Parameter: *Feature Types to Read*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

✳ Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

Mapping File Example

The example below selects a small area for extraction:

```
ORACLE_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]

* Workbench Parameter

Clip To Envelope

INTERACTION

Required/Optional: *Optional*

This specifies the type of relationship which must exist between the search envelope and the geometry in the target layer. Any supported relationship, or combination of relationships, may be specified. (Not supported for the Raster reader).

This table lists the valid geometry interaction relationships.

Search Method	Description
ANYINTERACT	The objects are non-disjoint.
CONTAINS	The interior and boundary of one object is completely contained in the interior of the other object.
COVEREDBY	The opposite of COVERS. A COVEREDBY B implies B COVERS A.
COVERS	The interior of one object is completely contained in the interior or the boundary of the other object and their boundaries intersect.
DISJOINT	The boundaries and interiors do not intersect.

Search Method	Description
EQUAL	The two objects have the same boundary and interior.
INSIDE	The opposite of CONTAINS. A INSIDE B implies B CONTAINS A.
ON	The interior and boundary of one object is on the boundary of the other object (and the second object covers the first object). This relationship occurs, for example, when a line is on the boundary of a polygon.
OVERLAPBDYDISJOINT	The interior of one object intersects the boundary and interior of the other object, but the two boundaries do not intersect. This relationship occurs, for example, when a line originates outside a polygon and ends inside that polygon.
OVERLAPBDYINTERSECT	The boundaries and interiors of the two objects intersect.
TOUCH	The boundaries intersect but the interiors do not intersect.

In addition to specifying a single relationship, one may specify a combination of relationships to be tested by concatenating them with a plus sign (+). For example, the `<ReaderKeyword>_INTERACTION` may be specified as `INSIDE + TOUCH`.

Workbench Parameter: *Relationship to Query Feature*

INTERACTION_RESULT

Required/Optional: *Optional*

This specifies the test that is applied to the results of the above geometry relationship comparison. When using the object model, Spatial queries return results of **TRUE** rather than the name of the interaction – as they do with the older relational model – so the default test for the object model is “= 'TRUE'”, regardless of the type of interaction involved. (Not supported for raster reader).

This directive is of little use when using the object model, and is provided only for backward compatibility.

Workbench Parameter: *Relationship Result Test*

WHERE_CLAUSE

Required/Optional: *Optional*

This specifies an SQL WHERE clause, which is applied to the table’s columns to limit the resulting features. This feature is currently limited to apply only to the attributes of the target Spatial layer, and does not allow for joining multiple tables together. The effect of table joins can be achieved using the object model, by specifying the entire queries in the DEF line with an `oracle_sql_encoded` parameter.

By default, there is no WHERE clause applied to the results, so all features in the layer are returned.

Workbench Parameter: *Where Clause*

CHUNK_SIZE

Required/Optional: *Optional*

The geometry is read from the Oracle database using a bulk reading technique to maximize performance. Normally 1000 rows of data are read from the database at a time.

This directive allows one to tune the performance of the reader. It specifies how many rows are read from the database at a time.

Workbench Parameter: *Rows to Read at a Time*

STRUCTURED_GEOMETRY (only applicable with classic geometry)

Required/Optional: *Optional*

Using enhanced geometry (by setting the global directive **FME_GEOMETRY_HANDLING** to **YES**) removes the need for using this directive and makes it simple to translate complex Oracle geometries into other formats, preserving the original geometry as it was stored in Oracle. Note that this keyword is not supported for raster reader.

Oracle 8i Spatial's object model stores geometry in a much more structured fashion than is normally used by FME features. While this structure can be mirrored in FME by using aggregates and list attributes, it is far more difficult to deal with than are simple points, lines, polygons, and donuts. Therefore, the Oracle Spatial reader normally flattens out the features – such as by combining polygons' boundaries and holes to form donuts, and stroking out sequential arcs to make simple line strings – before they are returned from the reader.

Occasionally, however, one might want a faithful representation of exactly how the geometry appears in the Oracle geometry object. The **STRUCTURED_GEOMETRY** directive, when given a value of **YES**, tells the Oracle 8i Spatial reader to provide the geometry in a form which mirrors the structure in the database. The representation of this structure is defined below, in the section titled *Structured Geometry Representation*.

Note: Structured geometry requires special processing: it should not be enabled unless a framework to deal with the structured geometry is provided in the mapping file or FME Objects application. It is only available when reading Oracle 8i Spatial tables using the object model – it is not available when using the relational model, or when writing to an Oracle 8i database.

BEGIN_SQL{n}

Occasionally you must execute some ad-hoc SQL prior to opening a table. For example, it may be necessary to ensure that a view exists prior to attempting to read from it.

Upon opening a connection to read from a database, the reader looks for the directive **<ReaderKeyword>_BEGIN_SQL{n}** (for $n=0,1,2,\dots$), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the **FME_SQL_DELIMITER** keyword, embedded at the beginning of the SQL block. The single character following this keyword will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;  
DELETE FROM instructors;  
DELETE FROM people  
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

*** Workbench Parameter**

SQL Statement to Execute Before Translation

END_SQL{n}

Occasionally you must execute some ad-hoc SQL after closing a set of tables. For example, it may be necessary to clean up a temporary view after writing to the database.

Just before closing a connection on a database, the reader looks for the directive `<ReaderKeyword>_END_SQL{n}` (for $n=0,1,2,\dots$), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the `FME_SQL_DELIMITER` directive, embedded at the beginning of the SQL block. The single character following this directive will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;
DELETE FROM instructors;
DELETE FROM people
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

* Workbench Parameter

SQL Statement to Execute After Translation

TOPOLOGY

Required/Optional: *Optional*

The reader reads `sdo_topo_geometry` column if the table contains it. If the table has both an `sdo_geometry` column and `sdo_topo_geometry` column, then only the column that appears first is read. If a table has both types, to ensure reading `sdo_topo_geometry` column, its name should be specified on the DEF line.

REMOVE_SCHEMA_QUALIFIER

Required/Optional: *Optional*

Specifies whether to keep or remove the schema qualifier. The full name of a table in an Oracle database is of the format `<schema_name>.<table_name>`. Setting this keyword to **YES** indicates that the reader should return the table name without any prefixes. This is useful when:

- creating a workspace that will be passed on to another organization using the same table names,

When this keyword is set to **YES** during the generation of a mapping file or workspace, the source feature types will be the table names without any prefix; otherwise, they will contain the owner name as a prefix. It is recommended that this keyword not be changed in value after generating the mapping file/workspace as it is possible for no features to be successfully passed onto the writer (since the writer is expecting feature types with different names).

Note that even when `REMOVE_SCHEMA_QUALIFIER` is set to **YES**, if the table is owned by a user other than the current user, the `<owner_name>` prefix will **not** be dropped so that the reader will find the correct table; however, the `<database_name>` prefix will still be dropped.

Value: YES | NO

Default Value: NO

Example:

ORACLE_REMOVE_SCHEMA_QUALIFIER YES

USE_UNIFIED_DATE_ATTRS

Required/Optional: *Optional*

Specifies whether we want to use unified date attributes, where the date and time are read into one attribute, or whether we want to use split date attributes, where two attributes are produced, one with only the date and another with both the date and time.

The value of this keyword should not be changed. It is automatically set to YES in new mapping files and workspaces. To maintain backwards compability, if this keyword is not present, the reader will behave as though the keyword is set to NO.

Value: YES | NO

Default Value: YES (in new mapping files and workspaces), NO otherwise

MAPINFO_SYMBOLOGY_STYLE_COLUMN

Required/Optional: *Optional*

This optional generation parameter specifies the name of the MapInfo symbology style column. If a column by this name is found, it will be omitted from the schema of the generated source feature type, and the oracle_mapinfo_symbology_style_column DEF line parameter will be set to the name of this column. (Not supported for raster reader).

MAPINFO_SYMBOLOGY_INDEX_COLUMN

Required/Optional: *Optional*

This optional generation parameter specifies the name of the MapInfo symbology index column. If a column by this name is found, it will be omitted from the schema of the generated source feature type. (Not supported for raster reader).

READ_3D_POLYGON_AS_FACE

Required/Optional: *Optional*

This optional keyword controls whether 3D polygons are read by Oracle Spatial as 3D face geometries or as regular polygons. (Not supported for raster reader).

Value: YES | NO

Default Value: NO

Workbench Parameter: Read 3D Polygons as Faces

HANDLE_MULTIPLE_SPATIAL_COLUMNS

If this directive is set to YES, feature geometry will be read into an aggregate. A directive is set on the aggregate to indicate that each part of the aggregate is independent from the others, and its own geometry. Geometry parts of the aggregate are named and contain geometry according to their respective column in the table being read. If a geometry is read as NULL, it will be appended to the aggregate as a null geometry.

When using this feature, neither the geometry column, nor the feature type SELECT statement can be specified. Also, tables with topology columns will not be read in this mode.

Required/Optional

Optional

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The Oracle Spatial writer module stores geometric, raster and attribute data in an Oracle Spatial database. Only upper-case table names are supported.

The Oracle Spatial writer provides the following capabilities:

- **Transaction Support:** The Oracle Spatial writer provides transaction support that eases the data loading process. Occasionally, a data load operation terminates prematurely due to data difficulties. The transaction support provides a mechanism for reloading corrected data without data loss or duplication.
- **Table Creation:** The Oracle Spatial writer uses the information within the FME mapping file to automatically create database tables as needed. If the relational model of storing geometry is used, the writer will create and populate all needed supporting tables. In such a case, the `<layername>`, `<layername>_SDOGEOM`, `<layername>_SDOLAYER`, `<layername>_SDOINDEX`, and `<layername>_SDODIM` tables will all be created. Likewise, creation of tables containing GeoRaster columns is also supported, including the corresponding Raster Data Table (RDT) creation. Metadata tables and triggers are also created accordingly.
- **Table Dropping:** The Oracle Spatial writer has an option that allows each table to be written to be dropped if re-creating, or truncated if appending. Metadata information is updated using triggers for rasters and it is replaced on write for geometry. Likewise, dropping of RDTs associated with the GeoRaster columns is performed if the RDT is empty. Metadata tables are updated via DML triggers, however the triggers themselves are not currently dropped.
- **Index Creation:** The Oracle Spatial writer will set up and populate all needed indexes and index tables as part of the loading process. For the relational model, indexes on `SDO_GID` columns in the `<layername>` and `<layername>_SDOGEOM` tables are created, and a compound index on the `SDO_GID` and `SDO_CODE` columns in the `<layername>_SDOINDEX` is created. The `<layername>_SDOINDEX` table will also be populated.
- **Bulk Loading:** The Oracle Spatial writer uses a bulk loading technique to ensure speedy data load.
- **Raster Interleaving:** Raster data can be written using Band SeQuential (BSQ), Band Interleaved by Line (BIL) or Band Interleaved by Pixel (BIP) interleaving. The default interleaving is BSQ.
- **Raster Compression:** Native DEFLATE, JPEG-B and JPEG-F compression can optionally be performed as a post-process once writing is complete.
- **Raster Pyramid Generation:** Native pyramid generation can optionally be performed as a post-process once writing is complete. Number of levels and the resampling type can be optionally specified.
- **Raster Validation:** Native validation can optionally be performed as a post-process once writing is complete.

Writer Directives

The directives processed by the Oracle Spatial writer are listed below. The suffixes shown are prefixed by the current `<writerKeyword>` in a mapping file. By default, the `<writerKeyword>` for the Oracle Spatial writer is `ORACLE8I` or `ORACLERASTER` when using the object model.

DATASET, USER_NAME, and WORKSPACE

These directives operate in the same manner as they do for the Oracle Spatial reader.

DEF

Required/Optional: *Required*

Each Oracle Spatial layer (table) must be defined before it can be written. The general form of an Oracle Spatial definition statement is:

```
ORACLE_DEF <tableName> \  
  [oracle_model object          \  
  [oracle_dim <dim>]           \  
  [oracle_srid <spatialReference>] \  
  [oracle_x_name <ord1>] \  
  [oracle_y_name <ord2>] \  
  [oracle_z_name <ord3>] \  
  [oracle_x_tol <xtol>] \  
  [oracle_y_tol <ytol>] \  
  [oracle_z_tol <ztol>] \  
  [oracle_min_x <xmin>] \  
  [oracle_min_y <ymin>] \  
  [oracle_min_z <zmin>] \  
  [oracle_max_x <xmax>] \  
  [oracle_max_y <ymax>] \  
  [oracle_max_z <zmax>] \  
  [oracle_create_indices (yes|no)] \  
  [oracle_index_params <paramString>] \  
  [oracle_levels <levels>] \  
  [oracle_numtiles <numtiles>] \  
  [oracle_gid_name <gidName>] \  
  [oracle_sql_encoded <sqlQuery>] \  
  [oracle_table_writer_mode (inherit_from_writer|insert|update|delete)] \  
  [oracle_update_key_columns <column>[,<column>]...] \  
  [oracle_default_geom_column <columnName>] \  
  [oracle_geom_column <columnName>] \  
  [oracle_raster_column <columnName>] \  
  [oracle_force_geom_updates (yes|no)] \  
  [oracle_force_raster_updates (yes|no)] \  
  [oracle_force_index_creation (yes|no)] \  
  [oracle_drop_table (yes|no)] \  
  [oracle_truncate_table (yes|no)] \  
  [oracle_contains_measures (yes|no)] \  
  [oracle_default_contains_measures (yes|no)] \  
  [oracle_gid_name <gidName>] \  
  [oracle_params <creationParams>] \  
  [oracle_sequenced_cols column[:seqname][;column[:seqname]]...] \  
  [<fieldName> <fieldType>]*
```

If the user wishes to create a table, the table definition allows control of the layer that will be created. Otherwise, the table definition serves to provide options for inserting, updating or deleting data in an existing table. In each case, some parameters may be unused. The recommended approach is to take advantage of the updated Workbench GUI to limit mistakes when setting the layer parameters.

If the table already exists in the database, then it is not necessary to list the fields and their types – FME will use the schema information in the database to determine this. If the fields and types are listed, they must match those in the database, however, not all fields must be listed.

If the table does not exist, then the field names and types are used to first create the table. In any case, if a **<field-Type>** is given, it must be a field type supported by the target database.

If the object model is being used, a field's type can be specified as geometry, which indicates that the named field is to hold the geometry contained on the FME features. If no geometry column is defined, and the table being written already exists with one or more columns of geometric data in the database, the FME will arbitrarily choose one of the table's geometric columns to contain the FME features' spatial components.

The same is true of GeoRaster columns which can be set as type GeoRaster.

The configuration parameters present on the definition line are described in the following table:

Parameter	Contents
oracle_model	<p>This indicates what model for storing geometry should be used.</p> <p>This is an optional parameter—the default is object, the only value supported for the Oracle Spatial (Object) writer.</p>
oracle_dim	<p>This specifies the dimension of the layer, which can currently be 2 or 3. The default is 2.</p>
oracle_srid	<p>This specifies the spatial referencing information for the geometry in the table. It is specified as an integer, and corresponds to the spatial reference identifier (SRID) column in the global table MDSYS.CS_SRS.</p> <p>All geometry within a given table must have the same spatial referencing. If the target table exists in the database and the value specified for oracle_srid does not match the value contained in USER_SDO_GEOM_METADATA, the metadata's SRID will be used in place of the specified oracle_srid.</p> <p>If oracle_srid is not specified, tables will be created with a NULL value for the SRID field.</p>
oracle_index_params	<p>This specifies a set of parameters which is inserted into the SQL query used to create an index for the table. Consult the Oracle Spatial documentation for information on specifying index creation parameters.</p>
oracle_levels	<p>This specifies the tessellation level used to create the spatial index for the layer. The larger the number, the longer spatial index creation will take by the finer the granularity of the index. The range is any integer between 1 and 64. The default is 0 is given, so that no levels will be specified when the index is created; depending on the version of the Oracle Spatial database being written to, this may force it to use RTree indexing instead of fixed or hybrid indexing. If using fixed or hybrid indexing, a positive integral value must be specified.</p>
oracle_numtiles	<p>This specifies the number of variable-sized tiles used (per geometry) when creating a hybrid spatial index within the object model. The range is any positive integer. If this is not specified at the time when a spatial index is created, and oracle_levels is specified, fixed spatial indexing will be used in the created index.</p>
oracle_x_name	<p>This specifies the name to use for the first ordinate. This name is used when the geometry column's information in</p>

Parameter	Contents
	the SDO_GEOM_METADATA table is created. The default is X.
oracle_y_name	This specifies the name to use for the second ordinate. This name is used when the geometry column's information in the SDO_GEOM_METADATA table is created. The default is Y.
oracle_z_name	This specifies the name to use for the third ordinate. This name is used when the geometry column's information in the SDO_GEOM_METADATA table is created. The default is Z.
oracle_x_tol	This specifies the comparison tolerance for the x coordinates. Coordinates in x that are closer than this value are considered equal. The default is 0.00000005.
oracle_y_tol	This specifies the comparison tolerance for the y coordinates. Coordinates in y that are closer than this value are considered equal. The default is 0.00000005.
oracle_z_tol	This specifies the comparison tolerance for the z coordinates. Coordinates in z that are closer than this value are considered equal. The default is 0.00000005.
oracle_m_tol	This specifies the comparison tolerance for the measures (LRS). Values in measures that are closer than this value are considered equal. The default is 0.00000005.
oracle_min_x	The minimum x value expected in the dataset. If any x values are present which are less than this value, the spatial index will give undefined results. For best spatial search performance, this value should be as close to the true minimum x as possible. This parameter must be specified.
oracle_min_y	The minimum y value expected in the dataset. If any y values are present which are less than this value, the spatial index will give undefined results. For best spatial search performance, this value should be as close to the true minimum y as possible. This parameter must be specified.
oracle_min_z	The minimum z value expected in the dataset. In the current release of Oracle Spatial, no indexing is done on the z axis, so the value can be arbitrarily assigned. This parameter must be specified if the dimension of the layer is 3.

Parameter	Contents
oracle_min_m	The minimum "measure" value expected in the dataset. Default is 0.
oracle_max_x	The maximum x value expected in the dataset. If any x values are present which are greater than this value, the spatial index will give undefined results. For best spatial search performance, this value should be as close to the true maximum x as possible. This parameter must be specified.
oracle_max_y	The maximum y value expected in the dataset. If any y values are present which are greater than this value, the spatial index will give undefined results. For best spatial search performance, this value should be as close to the true maximum y as possible. This parameter must be specified.
oracle_max_z	The maximum z value expected in the dataset. In the current release of Oracle Spatial, no indexing is done on the z axis, so the value can be arbitrarily assigned. This parameter must be specified if the dimension of the layer is 3.
oracle_max_m	The minimum "measure" value expected in the dataset. Default is 0.
oracle_create_indices	This indicates whether or not indices are to be created as part of the data load. The valid choices for the object model are yes and no. If no is specified, no index creation is done. The default is yes.
oracle_sql_encoded	This specifies an SQL INSERT or UPDATE query to be used to define the results. If this is specified, the Oracle Spatial writer will execute the query, defining one row for each feature from the FME. The values in the query are specified by embedding :attrName in the query itself, where attrName is the name of the FME feature's attribute; for example: <code>INSERT INTO EXAMPLE VALUES :a, :b</code> In this example, the attributes named a and b will be taken from each feature written to <tableName>. The attributes named in the query must be listed on the DEF line so that the FME knows what type to use. There is no necessary or implied correlation between the FME attribute name and the Oracle column name. Take, for example, this UPDATE query:

Parameter	Contents
	<p><code>UPDATE RR SET TEXTSTRING=:mytext WHERE ID=:myid</code></p> <p>In this example, the Oracle column named <code>ID</code> is compared to the value of each feature's attribute named <code>myid</code>, and the value of the table's column named <code>TEXTSTRING</code> is set from the feature attribute named <code>mytext</code>.</p> <p>If one of the named attributes is a geometry attribute, a given row's feature will define the geometry for that column. In this case, the <code>oracle_dim</code> parameter must be used to specify the dimension of the geometry. This is necessary because Oracle 8.1.5 does not provide a way for the FME to determine the dimension of an arbitrary column without knowing the exact name of the originating column and its source table.</p> <p>This parameter is available only when using the object model.</p> <p>This parameter is encoded as described in the section <i>Substituting Strings in Mapping Files</i> in FME Fundamentals help > Mapping File Syntax.</p>
oracle_params	<p>This specifies additional parameters to be appended to the Oracle CREATE query used to create the output table. It is used to specify table allocation characteristics and the like.</p> <p>If this is specified, it will override the global <code>CREATE_TABLE_PARAMS</code> directive.</p>
oracle_sequenced_cols	<p>Indicates which columns' values come from sequences. The format for this parameter is of the form</p> <pre>oracle_sequenced_cols col- umn1:seqname1;column2:seqname2;...</pre> <p>where "columnN" is the name of the column whose value is provided by the sequence, and "seqnameN" is the name of the sequence providing the value.</p> <p>If ":seqnameN" is not given, the column's value will be provided by a sequence with the same name as the column. Sequence names are case-sensitive. The sequences will be created if they do not already exist, in which case a message will be written to the log file.</p>
oracle_table_writer_mode	<p>The default operation mode of the feature type in terms of the types of SQL statements sent to the database. Valid values are <code>INSERT</code>, <code>UPDATE</code>, <code>DELETE</code> and <code>INHERIT_FROM_WRITER</code>. Note that <code>INSERT</code> mode allows for only <code>INSERT</code> operations where as <code>UPDATE</code> and <code>DELETE</code> can be over-</p>

Parameter	Contents
	<p>written at the feature levels. INHERIT_FROM_WRITER simply indicates to take this value from the writer level and not to override it at the feature type level.</p> <p>Default: INHERIT_FROM_WRITER</p>
<p>oracle_update_key_columns</p>	<p>This instructs the Oracle Spatial writer to perform an UPDATE operation on the table, rather than performing an INSERT. The argument is a comma-separated list of the columns which are matched against the corresponding FME attributes' values to specify which rows are to be updated with the other attribute values.</p> <p>For example: oracle_update_key_columns ID</p> <p>would have a similar effect to the "UPDATE" example in the above discussion of the oracle_sql_encoded directive. In this case, however, the FME attribute is always matched against the Oracle column with the same name. Also, the target table is always the feature type specified in the DEF line.</p> <p>Each column listed with the oracle_update_key_columns directive must be defined with a type on the DEF line, in addition to the columns whose values will be updated by the operation.</p>
<p>oracle_force_geom_updates</p>	<p>If this option is given a value of YES, and a table UPDATE operation is being performed, then the resulting UPDATE will affect the table's geometry column, even if no geometry column is specified on the DEF line. Otherwise, only attributes specified on the DEF line will be updated.</p>
<p>oracle_force_raster_updates</p>	<p>If this option is given a value of YES, and a table UPDATE operation is being performed, then the resulting UPDATE will affect the table's GeoRaster column, even if no GeoRaster column is specified on the DEF line. Otherwise, only attributes specified on the DEF line will be updated.</p>
<p>oracle_gid_name</p>	<p>This specifies that a particular column in the table is to hold a unique number. The FME will determine the highest value in this column when it starts writing to the table. This value will be incremented and written to the specified column for each row of the table. For example, if your table had the column MY_ID, which you wish to have a unique value, you'd include the oracle_gid_name in your DEF line</p> <pre> ORACLE8I_DEF MYTABLE \ oracle_gid_name MY_ID \ MY_ID INT \ </pre>

Parameter	Contents
	<pre>GEOM GEOMETRY \ NAME VARCHAR2(20)</pre>
<p>oracle_default_geom_column</p>	<p>When writing to the object model, the Oracle Spatial writer will create a column to hold the geometry, even if no attribute of type GEOMETRY is specified on the DEF line. This column is typically called GEOM, but may be changed to any other name by using this directive. For example:</p> <pre>oracle_default_geom_column GEOMETRIE</pre> <p>would cause the Oracle Spatial writer to write the FME features' geometry to a column named "GEOMETRIE" in the resulting table, if no other geometry column is specified in the DEF line.</p> <p>Note that this directive merely specifies a default value for the geometry column. If a geometry column or columns are explicitly named on the DEF line, the explicit name will be chosen instead of the default. If the table already exists in the Oracle database, then the geometry column will be chosen from those defined on the existing table.</p> <p>Also, if the writer directive "OVERRIDE_DEFAULT_GEOM" has been specified in the mapping file or FMEObjects session with a value of YES, then the default geometry column mechanism will be disabled. If this is the case, geometry columns will only be added to new tables if explicitly listed in the DEF line.</p>
<p>oracle_geom_column</p>	<p>This table definition parameter is the same as the one above except that it is operative only when the table exists and is being updated. If this is the case, then this parameter specifies which of the possibly multiple geometry columns to update with the geometry on the update feature.</p>
<p>oracle_raster_column</p>	<p>When writing to the object model, the Oracle Spatial writer will create a column to hold the GeoRaster object if this parameter has a non-blank value. The default column is typically called RAST, but may be changed to any other name by using this directive. For example:</p> <pre>oracle_raster_column RASTERIZER</pre> <p>would cause the Oracle Spatial Raster writer to write the FME feature's geometry to a column named "RASTERIZER" in the resulting table, if no other GeoRaster column is specified in the DEF line.</p> <p>Note that this directive merely specifies a default value for the geometry column. If a GeoRaster column or col-</p>

Parameter	Contents
	<p>umns are explicitly named on the DEF line, the explicit name will be chosen instead of the default. If the table already exists in the Oracle database, then the GeoRaster column will be chosen from those defined on the existing table.</p>
oracle_index_name	<p>This specifies the name for the spatial index that will be created on the table. If this is not specified, a spatial index name will be created based on a database sequence and the name of the table being written.</p>
oracle_force_index_creation	<p>This specifies whether a new spatial index will always be created when writing to a table. If set to yes, a new spatial index will always be created for the table being written, and any existing spatial index with the same name will be dropped before it is created. Otherwise index creation will not occur when the table being written already contains an index.</p>
oracle_drop_table	<p>This specifies whether a table should be dropped, if it exists, before being recreated. If the table does not exist, then the operation is ignored and the user is warned. Note that the drop table option is only available when specifying table creation parameters.</p>
oracle_truncate_table	<p>This specifies whether a table should be truncated, if it exists, before data is inserted. If the table does not exist, then the operation is ignored and the user is warned. Note that the truncate table option is only available when not specifying table creation parameters.</p>
oracle_contains_measures	<p>This directs the writer to write measures to the destination table. When this directive is set to yes and the incoming feature does not have any measures, then null values are written. This parameter applies when writing to existing tables. Default is NO.</p>
oracle_default_contains_measures	<p>This directs the writer to write measures to the destination table. When this directive is set to yes and the incoming feature does not have any measures, then null values are written. This parameter applies when writing to new tables. Default is NO.</p>

Linear Referencing (Measures): The writer will consider table metadata definition first before writing measures to an existing table. If the DEF line parameters oracle_contains_measures or oracle_default_contains_measures do not agree with the table definition, the writer will issue a warning indicating whether or not either measure has

been written, even if those directives are set to No or Yes respectively. Those two parameters will be followed only when oracle_drop_table is set to Yes or the table does not exist before writing.

When creating new tables, the writer will create the table with metadata definitions specified by the DEF line parameters. Currently, 3D surfaces and solids do not handle measure reading and writing – therefore, null value placeholders will be used for those geometries when writing measures to oracle. All other geometries will have measure values written if values are assigned; otherwise, null values will be written.

START_TRANSACTION

Required/Optional: *Optional*

This statement tells the Oracle Spatial writer module when to start actually writing features into the database. The Oracle Spatial writer does not write any features until the feature is reached that belongs to **<last successful transaction> + 1**. Specifying a value of zero causes every feature to be output. Normally, the value specified is zero – a non-zero value is only specified when a data load operation is being resumed after failing partway through.

Parameter	Contents
<last successful transaction>	The transaction number of the last successful transaction. When loading data for the first time, set this value to 0.

Example:

```
ORACLE_START_TRANSACTION 0
```

Workbench Parameter: *Transaction to Start Writing At*

TRANSACTION_INTERVAL

Required/Optional: *Optional*

This statement informs the FME about the number of features to be placed in each transaction before a transaction is committed to the database.

If the **ORACLE_TRANSACTION_INTERVAL** statement is not specified, then a value of 2000 is used as the transaction interval.

Parameter	Contents
<transaction_interval>	The number of features in a single transaction.

Example:

```
ORACLE_TRANSACTION_INTERVAL 5000
```

Workbench Parameter: *Features to Write Per Transaction*

CREATE_TABLE_PARAMS

Required/Optional: *Optional*

This statement allows the mapping file to specify parameters that are defined on the tables created by the Oracle Spatial writer. Its value is appended to the SQL **CREATE TABLE** query used to create the tables.

If the **CREATE_TABLE_PARAMS** statement is not specified, then each table will be created with the default table parameters, unless the table's DEF line contains the oracle_params directive. An individual table's **oracle_params** will always take precedence over the global **CREATE_TABLE_PARAMS** directive.

Parameter	Contents
<parameters>	The string to be appended to the CREATE TABLE query.

Example:

```
ORACLE8I_CREATE_TABLE_PARAMS "TABLESPACE JOEDATA1"
```

CHUNK_SIZE

See the `CHUNK_SIZE` directive in the *Reader Directives* section.

Workbench Parameter: *Features Per Bulk Write*

OVERRIDE_DEFAULT_GEOM

Required/Optional: *Optional*

FME normally includes a geometry column named **"GEOM"** in any Oracle Spatial (object model) table it creates. This is true for automatically generated semantic mapping files, as well as for tables created within an FME Objects application.

The geometry column name can be changed in a mapping file by using the `"oracle_default_geom_column"` parameter to the DEF line, or can be completely eliminated by removing any mention of a geometry column or default geometry column. This, however, does not apply to FME Objects applications, which cannot be as explicit about the contents of the DEF line.

The `OVERRIDE_DEFAULT_GEOM` directive allows an FME Objects application to create tables in Oracle 8i Spatial which have no geometric data. If this directive is given with a **YES** value, then the default geometry column mechanism will be completely disabled, and tables will be created with geometry columns only if such columns are explicitly defined as attributes of type `GEOMETRY`.

The following FME Objects code (written in Java) shows an example of disabling the default geometry mechanism for an Oracle Spatial (object model) writer:

```
...
IFMESTringArray writerDirectives = session.createStringArray();
...
writerDirectives.append("ORACLE8I_OVERRIDE_DEFAULT_GEOM");
writerDirectives.append("Yes");
writer.open(destInfo.dataset,writerDirectives);
...
```

BEGIN_SQL{n}

This directive is described in the *Reader Directives* section. In the case of the writer, the statements will be executed only when the first feature actually written to the dataset.

Workbench Parameter: *SQL Statement to Execute Before Translation*

END_SQL{n}

This directive is described in the *Reader Directives* section. In the case of the writer, the statements will be executed only if at least one feature has been written to the dataset.

Workbench Parameter: *SQL Statement to Execute After Translation*

STRICT_ATTR_CONVERSION

This directive instructs the Oracle writer on how to proceed when a problem arises while converting a value from one of a feature's attributes to an oracle column value. Examples of such problems would be the truncation of a string value to fit into the target character column, an error in converting a non-numeric attribute to write to a numeric column, or an error converting an FME geometry to fit into an `SDO_GEOMETRY` value.

In normal operation, the Oracle writer will silently truncate strings which are too long, or null out values which cannot be successfully converted. It can optionally log features which have conversion problems, or drop problem features and write a warning to the log.

Possible values for this directive are summarized in the following table:

Parameter	Contents
NO	Silently ignore conversion errors. (This is the default behaviour.)
YES	Features are dropped from the translation and a warning is written to the log.
WARN	Log any features causing conversion errors, and then continue the translation as usual

Example:

```
ORACLE8I_STRICT_ATTR_CONVERSION WARN
```

Workbench Parameter: *Enforce Strict Attribute Conversion*

WRITER_MODE

Required/Optional: *Optional*

Note: For more information on this directive, see the chapter **Database Writer Mode**.

This directive informs the Oracle writer which SQL operations will be performed by default by this writer. This operation can be set to **INSERT**, **UPDATE** or **DELETE**. The default writer level value for this operation can be overwritten at the feature type or table level. The corresponding feature type DEF parameter name is called `oracle_table_writer_mode`. It has the same valid options as the writer level mode and additionally the value **INHERIT_FROM_WRITER** which causes the writer level mode to be inherited by the feature type as the default for features contained in that table.

The operation can be set specifically for individual feature as well. Note that when the writer mode is set to **INSERT** this prevents the mode from being interpreted from individual features and all features are inserted unless otherwise marked as **UPDATE** or **DELETE** features. These are skipped.

If the **WRITER_MODE** statement is not specified, then a value of **INSERT** is given.

Parameter	Contents
<writer_mode>	<p>The type of SQL operation that should be performed by the writer. The valid list of values are below:</p> <p style="text-align: center;">INSERT UPDATE DELETE</p> <p>Default: INSERT</p>

Example:

```
ORACLE8I_WRITER_MODE INSERT
```

Workbench Parameter: *Writer Mode*

HANDLE_MULTIPLE_SPATIAL_COLUMNS

If this directive is set to YES, feature geometry will be written from an aggregate. This aggregate must contain individual geometries, namely that each part is independent from the others and is its own complete geometry. Each part geometry of the aggregate must have a name. If the aggregate contains geometries with names that match the spatial columns of the table being written, the geometries will be written to the appropriate columns. If the aggregate does not contain a name that matches a spatial column on the table being written NULL will be written in INSERT mode.

When using this feature, the geometry column cannot be specified and the table must already exist.

Required/Optional

Optional

Writer Mode Specification

The Oracle Spatial writer allows the user to specify a writer mode, which determines what database command should be issued for each feature received. Valid writer modes are **INSERT**, **UPDATE** and **DELETE**.

Writer Modes

In **INSERT** mode, the attribute values of each received feature are written as a new database record.

In **UPDATE** mode, the attribute values of each received feature are used to update existing records in the database. The records which are updated are determined via the **oracle_update_key_columns** DEF line parameter, or via the **fme_where** attribute on the feature.

In **DELETE** mode, existing database records are deleted according to the information specified in the received feature. Records are selected for deletion using the same technique as records are selected for updating in **UPDATE** mode.

Writer Mode Constraints

In **UPDATE** and **DELETE** mode, the **fme_where** attribute always takes precedence over the **oracle_update_key_columns** DEF line parameter. If both the **fme_where** attribute and the **oracle_update_key_columns** DEF line parameter are not present, then **UPDATE** or **DELETE** mode will generate an error.

When the **fme_where** attribute is present, it is used verbatim as the WHERE clause on the generated **UPDATE** or **DELETE** command. For example, if **fme_where** were set to 'id<5', then all database records with field id less than 5 will be affected by the command.

When the **fme_where** attribute is not present, the writer looks for the **oracle_update_key_columns** DEF line parameter and uses it to determine which records should be affected by the command. Please refer to "DEF" on page 989 for more information about the **oracle_update_key_columns** DEF line parameter.

Writer Mode Selection

The writer mode can be specified at three unique levels: on the writer level, on the feature type, or on individual features.

At the writer level, the writer mode is specified by the **WRITER_MODE** keyword. This keyword can be superseded by the feature type writer mode specification.

Note: For more information on this directive, see the chapter *Database Writer Mode*.

At the feature type level, the writer mode is specified by the **oracle_writer_mode** DEF line parameter. This parameter supersedes the **WRITER_MODE** keyword. Unless this parameter is set to **INSERT**, it may be superseded on individual features by the **fme_db_operation** attribute. Please refer to the DEF line documentation for more information about this parameter.

At the feature level, the writer mode is specified by the **fme_db_operation** attribute. Unless the parameter at the feature type level is set to **INSERT**, the writer mode specified by this attribute always supersedes all other values. Accepted values for the **fme_db_operation** attribute are **INSERT**, **UPDATE** or **DELETE**.

FME Raster Features

FME raster features represent raster data and use several concepts that are unlike those used in the handling of vector data. See [About FME Rasters](#).

Oracle supports rasters with an arbitrary number of bands, provided all bands are the same data type. Any number of bands may optionally have a palette.

Oracle supports reading and writing 1, 2, and 4 bit rasters. Note that the smallest data type supported by FME is 1 byte (8 bits). Thus, on reading, 1, 2, and 4 bit rasters are automatically converted to 8 bits. On writing, an option is provided to reduce the size of 8 bit rasters to 1, 2, or 4 bits.

Feature Representation

Both vector and raster features read from Oracle Spatial consist of a series of attribute values and either geometry or raster data. The feature type of each Database feature is as defined on its [DEF](#) line.

Features written to the database have the destination table as their feature type, and attributes as defined by on the [DEF](#) line.

In addition to the generic FME feature attributes that FME Workbench adds to all features (see [About Feature Attributes](#)), the Oracle Spatial module makes use of the following special attribute names:

Attribute Name	Contents
oracle_type	<p>The type of geometric entity stored within the feature. The valid values for the object model are listed below:</p> <ul style="list-style-type: none">oracle_niloracle_pointoracle_lineoracle_areaoracle_arcoracle_rectangleoracle_circleoracle_solidoracle_surfaceoracle_multipointoracle_multilineoracle_multipolyoracle_multisolidoracle_multisurfaceoracle_collectionoracle_raster
oracle_srid	<p>This attribute is defined only when reading from an Oracle table which has spatial referencing information. It contains the spatial reference identifier (SRID) for the geometry column retrieved. Its value will be the same for all features returned from a particular table when writing vector geometry. Raster features each have an individual coordinate system per feature which specified by this attribute. The SRID value is used to look up a coordinate system, which is converted to an FME coordinate system</p>

Attribute Name	Contents
	and attached to each feature returned by the reader. This attribute is not used by the Oracle Spatial writer when writing vector geometry.

Features read from, or written to, Oracle Spatial also have an attribute for each column in the database table. The feature attribute name will be the same as the source or destination column name. The attribute and column names are case-sensitive.

Note: The geometry descriptions given here are only entirely correct when using the default geometry encoding. When reading structured geometry from a table with the object model (using the STRUCTURED_GEOMETRY directive), the geometry will be formatted slightly differently. The section Structured Geometry Representation describes how geometry objects are represented when structured geometry is enabled.

SDO_POINT Field

Oracle Spatial's object model allows each geometry to have a "point" location, in addition to any ordinates defining the geometry. It stores this information in a field named **SDO_POINT** within the geometry object. If this field is not **NULL** in the Oracle Spatial geometry object, this location is stored on the FME feature using the following attributes:

Attribute Name	Contents
oracle_sdo_point.x	The first ordinate of the Oracle Spatial geometry's SDO_POINT field. This attribute is defined if this field is defined on the geometry.
oracle_sdo_point.y	The second ordinate of the Oracle Spatial geometry's SDO_POINT field. This attribute is defined if this field is defined on the geometry.
oracle_sdo_point.z	The third ordinate of the Oracle Spatial geometry's SDO_POINT field. This attribute is defined if this field is defined on the geometry, and has a "z" component.

Notice, however, that simple points are normally represented in Oracle Spatial geometry objects with an empty ordinate array, with the actual point location stored in the **SDO_POINT** field. In this case, the FME feature will have a "point" geometry and will not contain the **oracle_sdo_point** attributes. If the Oracle geometry has both an ordinate list defining a point *and* a defined **SDO_POINT** field, the FME feature will contain both a point geometry and the required **oracle_sdo_point** attributes.

Unknown Elements

Oracle Spatial (object) structures each geometry object with zero or more elements, each with a specified numeric type. Normally these types represent an actual geometry type, such as "point" or "line", but there is also a type defined as "unknown". Unknown elements are ignored by Oracle Spatial's geometric computation procedures, and may be used by applications to store whatever numeric data they see fit to store.

FME has the ability to read and write a geometry's unknown elements. It represents the unknown elements entirely with feature attributes, of the form **oracle_unknown_element{m}.attrName**, where **m** ($m \geq 0$) is the ordinal position of the element relative to the other unknown elements, and **attrName** has one of the following values.

Attribute Name	Contents
interpretation	The interpretation tells the application what kind of data is represented by the unknown element. This

Attribute Name	Contents
	may be any integral value, and is not interpreted specially by FME or Oracle in any way.
num_ordinates	This specifies the number of ordinates that make up the element. The ordinates are stored in Oracle as a one-dimensional array of numbers. A given element may have zero or more ordinates associated with it.
ordinate{m}	This is the number at the given ordinate position ($0 \leq m \leq (\text{num_ordinates}-1)$). It can be any number representable in an Oracle NUMBER type.

The unknown elements are given a sequence number among all of a given feature's unknown elements, but are not given any hint as to their position relative to other elements in a geometry. That is, FME sees the elements as an array of information added to an Oracle geometry object, and pays no attention to where the elements are placed in relation to "real" geometric elements of the feature. When writing to Oracle Spatial objects, FME always places the unknown elements before all other elements in the geometry.

No Coordinates

oracle_type: oracle_nil

Features with no coordinates are tagged with this value when reading or writing to or from Oracle Spatial.

Points

oracle_type: oracle_point

Features tagged with this value consist of a single point or an aggregate of points. When the object model is being used, an aggregate or line geometry tagged as **oracle_point** will be written as a "point cluster" with one or more coordinates stored as actual ordinates in the point, rather than being written as a single coordinate stored in the output geometry object's **SDO_POINT** field, or coerced to being an **oracle_line**.

Oracle's oriented points are also supported. In this case, the **SDO_POINT** field is set to NULL, and the orientation of the point is specified on the FME feature either as a set of three numbers describing the orientation in 3D space, or a single value describing the planar rotation, as summarized in the following table. Orientation will be read through geometry traits and attributes as per the chart below. Similarly, when writing, point orientation can be provided using either attributes or traits.

Attribute/Trait Name	Contents
oracle_orientation	The orientation of the point, expressed in degrees counter-clockwise from the positive X axis. When reading, this is an approximation of the orientation described by the (i,j,k) orientation vector. When writing, this value is used to compute an (i,j,0) unit vector on the horizontal plane, if no such (i,j,k) vector is defined on the feature being written.

Attribute/Trait Name	Contents
oracle_orient_i	The "X axis" component of a vector describing the point's orientation. This vector is normally a unit vector in 3D space. If the three components of the orientation vector are present on a feature being written, then the vector is used in place of any value present on the oracle_orientation attribute.
oracle_orient_j	The "Y axis" component of a vector describing the point's orientation. The comments above for oracle_orient_i apply to this attribute as well.
oracle_orient_k	The "Z axis" component of a vector describing the point's orientation. The comments above for oracle_orient_i apply to this attribute as well.

Lines

oracle_type: oracle_line

Linear features are tagged with this value when reading or writing to or from Oracle Spatial. Both single part and aggregate linear features are supported.

Aggregates are written out as "multiline" geometry containing several linear elements, just as if the feature had been tagged with **oracle_multiline**. Any non-linear elements contained in the aggregate are discarded.

Areas

oracle_type: oracle_area

Area features are tagged with this value when reading or writing to or from Oracle Spatial. Both single part and aggregate area features are supported. An area feature may be either a polygon or a donut polygon. Note that checking is done to ensure that the area features adhere to the geometry rules of Oracle Spatial as they are loaded.

Aggregates are written out as "multipolygon" geometry containing several polygonal elements, just as if the feature had been tagged with **oracle_multiline**. Any non-polygonal elements contained in the aggregate are discarded.

Arcs

oracle_type: oracle_arc

Arc features are tagged with this value when reading or writing to or from Oracle Spatial (object model). The arc is defined in the FME feature by a center point and a number of attributes to define the shape of the arc.

Attribute Name	Contents
oracle_primary_radius	The length of the arc's semi-major axis, measured in ground units.
oracle_secondary_radius	The length of the arc's semi-minor axis, measured in ground units.
oracle_start_angle	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of start_angle.

Attribute Name	Contents
oracle_sweep_angle	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of sweep_angle.
oracle_rotation	The rotation of the major axis. The rotation is measured in degrees counterclockwise from horizontal.

It is important to note that, as of the time of writing, there is no way to represent a non-circular arc. If oracle_primary_radius and oracle_secondary_radius are specified as different values, the FME will convert the arc to a line string approximating the arc prior to writing it to the Oracle geometry object.

It is also important to note that when reading arcs from Oracle Spatial, the FME does not always leave the arc in its original form. If the arc is part of an aggregate type – multipolygon, multiline or collection geometry, or a multipoly or multiline element within a polygon or line geometry – FME will first convert the arc to a line string approximation, in order to simplify processing of the resulting aggregate. At some point in the future, Safe Software may implement a mode in which the Oracle Spatial reader provides a more faithful representation of the exact structure of the geometry embedded in an Oracle table’s geometry columns.

Note that when in enhanced geometry mode, stroking will not take place and all the arcs will be preserved.

Rectangles

oracle_type: oracle_rectangle

Oracle Spatial rectangle objects are represented in the FME by closed polygons. When a rectangle is read, it is turned into a closed polygon feature. When a feature is written tagged with an **oracle_rectangle** type, its minimum bounding rectangle is computed, and the resulting lower-left and upper-right coordinates will form the ordinates for the Oracle geometry.

Circle

oracle_type: oracle_circle

Circle features are tagged with this value when reading or writing to or from Oracle Spatial (object model). The circle is defined in the FME feature by a center point and an attribute to define the circle’s radius:

Attribute Name	Contents
oracle_radius	The length of the circle’s semi-major axis, measured in ground units.
oracle_rotation	The rotation of the major axis. The rotation is measured in degrees counterclockwise from horizontal.

It is important to note that when reading circles from Oracle Spatial, FME does not always leave the circle in its original form. If the circle is part of an aggregate type (multipolygon or collection geometry) or defines a hole or boundary of a donut, FME will first “stroke” a linear approximation of the circle, and use that in the structure.

Note that when in enhanced geometry mode, stroking will not take place and all the arcs will be preserved.

Solids

oracle_type: oracle_solid

Solid features are tagged with this value when reading or writing to or from Oracle Spatial (object model).

Solid features are supported only when writing to an Oracle Database version 11g or later. If the Oracle Spatial (object model) writer detects a version of Oracle Database older than version 11g, solid features will automatically be downgraded to a 2D representation prior to writing.

Oracle Spatial (object model) directly supports simple solids, composite solids and optimized solids.

For writing, all other types of solid geometries (e.g., extrusions) are decomposed into simple solids prior to writing.

Surfaces

oracle_type: oracle_surface

Surface features are tagged with this value when reading or writing to or from Oracle Spatial (object model).

Surface features are supported only when writing to an Oracle Database version 11g or later. If the Oracle Spatial (object model) writer detects a version of Oracle Database older than version 11g, surface features will automatically be downgraded to a 2D representation prior to writing.

Oracle Spatial (object model) directly supports composite surfaces, 3D polygons and rectangles.

For reading, 3D polygons and rectangles are only read as surfaces if READ_3D_POLYGON_AS_FACE is set to YES.

For writing, all other types of surface geometries (e.g., triangle fans or strips) are decomposed into composite surfaces prior to writing.

Multipoints

oracle_type: oracle_multipoint

Aggregate point features are tagged with this value when reading or writing to or from Oracle Spatial. When writing to Oracle, each element of the aggregate must have a "point" geometry; others will simply be discarded before writing.

Oracle's oriented points are supported for each element of the point aggregate. On write, these can be specified on each point geometry as a trait. On read, each point geometry element in the multipoint will have the below geometry traits if the point has an orientation.

Trait Name	Contents
oracle_orientation	The orientation of the point, expressed in degrees counter-clockwise from the positive X axis. When reading, this is an approximation of the orientation described by the (i,j,k) orientation vector. When writing, this value is used to compute an (i,j,0) unit vector on the horizontal plane, if no such (i,j,k) vector is defined on the feature being written.
oracle_orient_i	The "X axis" component of a vector describing the point's orientation. This vector is normally a unit vector in 3D space. If the three components of the orientation vector are present on a feature being written, then the vector is used in place of any value present on the oracle_orientation attribute.
oracle_orient_j	The "Y axis" component of a vector describing the point's orientation. The comments above for oracle_orient_i apply to this attribute as well.
oracle_orient_k	The "Z axis" component of a vector describing the point's orientation. The comments above for oracle_orient_i apply to this attribute as well.

Multi-lines

oracle_type: oracle_multiline

Aggregate linear features are tagged with this value when reading or writing to or from Oracle Spatial. When writing to Oracle, each element of the aggregate must have a linear geometry; others will simply be discarded before writing.

When reading multiline features from Oracle, any contained arcs will be stroked out to a linear approximation before being placed into the FME feature. This results in a less accurate representation of the geometry actually contained in the Oracle table, but greatly simplifies the processing of such data. At some point in the future, Safe Software may implement a mode in which the Oracle Spatial reader provides a more faithful representation of the exact structure of the geometry embedded in an Oracle table's geometry columns.

Note that when in enhanced geometry mode, stroking will not take place and all the arcs will be preserved.

Multipolygons

oracle_type: oracle_multipoly

Aggregate polygonal features are tagged with this value when reading or writing to or from Oracle Spatial. When writing to Oracle, each element of the aggregate must have a polygonal geometry; others will simply be discarded before writing.

When reading multipolygon features from Oracle, any contained rectangles and circles will be stroked out to a linear approximation before being placed into the FME feature. This results in a less accurate representation of the geometry actually contained in the Oracle table, but greatly simplifies the processing of such data. At some point in the future, Safe Software may implement a mode in which the Oracle Spatial reader provides a more faithful representation of the exact structure of the geometry embedded in an Oracle table's geometry columns.

Note that when in enhanced geometry mode, stroking will not take place and all the arcs will be preserved.

Multi-Solids

oracle_type: oracle_multisolid

Aggregate solid features are tagged with this value when reading or writing to or from Oracle Spatial (object model). When writing to Oracle, each element of the aggregate must have a solid geometry.

Solid features are supported only when writing to an Oracle Database version 11g or later. If the Oracle Spatial (object model) writer detects a version of Oracle Database older than version 11g, solid features will automatically be downgraded to a 2D representation prior to writing.

Oracle Spatial (object model) directly supports simple solids, composite solids and optimized solids.

For writing, all other types of solid geometries (e.g., extrusions) are decomposed into simple solids prior to writing.

Multi-Surfaces

oracle_type: oracle_multisurface

Aggregate surface features are tagged with this value when reading or writing to or from Oracle Spatial (object model). When writing to Oracle, each element of the aggregate must have a surface geometry.

Surface features are supported only when writing to an Oracle Database version 11g or later. If the Oracle Spatial (object model) writer detects a version of Oracle Database older than version 11g, surface features will automatically be downgraded to a 2D representation prior to writing.

Oracle Spatial (object model) directly supports composite surfaces, 3D polygons and rectangles.

For reading, 3D polygons and rectangles are only read as surfaces if READ_3D_POLYGON_AS_FACE is set to YES.

For writing, all other types of surface geometries (e.g., triangle fans or strips) are decomposed into composite surfaces prior to writing.

Raster

oracle_type: oracle_raster

Features with raster data are unlike vector data in that they can contain various types of data arranged in blocks of pixels.

Attribute Name	Contents
oracle_raster_compression_type	The type of compression to be used. This option is the one used to determine whether or not compression will occur. Current options are NONE, DEFLATE, JPEG-B and JPEG-F. If this optional attribute is not specified, the default value is NONE and the GeoRaster object is not compressed. If the attribute value is set to NONE for an existing GeoRaster object, then the

Attribute Name	Contents
	object will be uncompressed. Lastly, any other options will cause the specified type of compression to occur in either INSERT or UPDATE operations. Note that compression is applied as a post-process.
oracle_raster_compression_update_type	<p>The type of compression to be used when updating an existing GeoRaster object. This option overrides oracle_raster_compression_type when an update is being performed.</p> <p>Current options are PRESERVE, REMOVE, DEFLATE, JPEG-B, and JPEG-F. PRESERVE preserves the existing compression on the GeoRaster object, provided the data is not being rewritten. REMOVE removes compression on the existing GeoRaster object. The other options cause the specified type of compression to be applied.</p>
oracle_raster_compression_quality	The JPEG compression quality, which is the degree of lossiness caused by the compression. Valid values are integers from 0 (lowest quality) to 100 (highest quality). This value is ignored when not using one of the JPEG compression types.
oracle_raster_default_red_band	This optional parameter specifies the index of the red band in the raster. Note that the index is zero based so that the first band is at index 0. If this parameter is not present on any the input rasters then a very simple default method is used to determine whether to treat 3 or 4 band rasters as RGB or RGBA.
oracle_raster_default_green_band	This optional parameter specifies the index of the green band in the raster. Note that the index is zero based so that the first band is at index 0. If this parameter is not present on any the input rasters then a very simple default method is used to determine whether to treat 3 or 4 band rasters as RGB or RGBA.
oracle_raster_default_blue_band	This optional parameter specifies the index of the blue band in the raster. Note that the index is zero based so that the first band is at index 0. If this parameter is not present on any the input rasters then a very simple default method is used to determine whether to treat 3 or 4 band rasters as RGB or RGBA.
oracle_raster_gcp_table_name	The Ground Control Point (GCP) table name for the external storage of GCPs that may exist in the source data. This may be unique for each raster. The default GCP table name is the RDT table name with an suffix of _GCP. If GCPs are not present on any the input rasters then this table will not be created.
oracle_raster_interleaving_type	This optional parameter specifies the type of interleaving to be applied to the data when written. Possible interleaving values include: BSQ (Band SeQuential), BIL (Band Interleaved by Line) or BIP (Band Interleaved by Pixel). The default interleaving is BSQ. If the raster contains only a single band then interleaving is irrelevant. This interleaving parameter does not apply to palette values which are always BIP

Attribute Name	Contents
	interleaved.
oracle_raster_number_of_bits_per_cell	This optional parameter specifies the bit depth of the data when written. Possible values are 1, 2, 4, and AUTO. When AUTO is selected, the bit depth will be determined by the interpretation of the input raster bands. When a specific bit depth is selected, the interpretation of the input raster bands must be one of UINT8, GRAY8, RED8, GREEN8, BLUE8, and ALPHA8. In this case, data values will be truncated to the specified number of bits. For example, if the number of bits is set to 4, a value of 201 (1100 1001 in base 2) will become 9 (1001 in base 2).
oracle_raster_pyramid_max_level	This optional parameter specifies the maximum number of pyramid levels to be generated. It is a maximum because pyramids will not be generated when the number of rows or columns in a GeoRaster is less than 64. The default for this value is to auto-calculate the correct number of pyramid levels for the raster so that the smallest pyramid has at least 64 rows and columns.
oracle_raster_pyramid_resampling	The type of resampling to be applied when generating pyramid levels. The options are NN, BILINEAR, AVERAGE4, AVERAGE16, and CUBIC. The default is NN or Nearest Neighbor.
oracle_raster_pyramid_type	This optional parameter specifies the type of pyramid to create. The options are NONE and DECREASE. If the attribute value is set to NONE for an existing GeoRaster object, the existing pyramids will be deleted.
oracle_raster_pyramid_update_type	The type of pyramid to create when updating an existing GeoRaster object. This option overrides oracle_raster_pyramid_type when an update is being performed. Current options are PRESERVE, REMOVE, and DECREASE. PRESERVE preserves the existing pyramids, provided the data is not being rewritten. REMOVE removes the existing pyramids. DECREASE generates new pyramids.
oracle_raster_table_name	The Raster Data Table (RDT) name for the given GeoRaster object. This table is where the pixel values of the input rasters will be stored. The default value is an automatically generated table name based on the raster identifier both of which are assigned by the database itself. the expected table name is 'RDT_<rasterid>\$'.
oracle_raster_lob_type	The large object (LOB) storage type used for storing raster data. This option only has an effect when creating a new Raster Data Table (RDT). The options are BASICFILE and SECUREFILE. BASICFILE specifies the original Oracle LOB storage type.

Attribute Name	Contents
	<p>SECUREFILE is a new LOB storage type introduced in version 11g which offers many advantages, including better performance.</p> <p>The default value depends on the version of the Oracle database being written to. When the version is 11g or later, SECUREFILE will be used; when the version is older than 11g, BASICFILE will be used.</p>
oracle_raster_tile_size_x	The horizontal size in pixels of a tile in the Raster Data Table (RDT). The value must be a power of 2.
oracle_raster_tile_size_y	The vertical size in pixels of a tile in the Raster Data Table (RDT). The value must be a power of 2.
oracle_raster_validate	<p>The option to validate the raster once written. If set to YES this option employs native GeoRaster validation and logs either TRUE if the GeoRaster is valid, or else it logs the Oracle error number that defines the reason for invalidity. The default value is NO indicating no validation will occur. Note that validation is checked as a post-writing process and happens before pyramiding and compression.</p>
oracle_raster_extent_srid	<p>This option is similar to the oracle_srid format attribute except that this SRID only applies to the spatial extent geometry stored inside the georaster object. This option allows users to set a different SRID for the extent of the raster that the raster itself. Valid values include the valid range of SRID values acceptable to Oracle. The default value is 0, indicating no SRID is present. Note that initially the extents are written in the same SRID as the raster and then reprojection will occur as necessary as a post-process.</p>

Collections

oracle_type: oracle_collection

Aggregates containing heterogeneous collections of point, line and polygon features are tagged with this value when reading or writing to or from Oracle Spatial.

When the global directive **FME_GEOMETRY_HANDLING** is set to **enhanced** the reader reads all the pieces in the original form (i.e., without stroking arcs or ellipses). The remainder of this section pertains to classic geometry cases.

When reading **oracle_collection** features from Oracle, any contained arcs, rectangles and circles will be stroked out to a linear approximation before being placed into the FME feature. This results in a less accurate representation of the geometry actually contained in the Oracle table, but greatly simplifies the processing of such data. At some point in the future, Safe Software may implement a mode in which the Oracle Spatial reader provides a more faithful representation of the exact structure of the geometry embedded in an Oracle table's geometry columns.

When writing **oracle_collection** features to Oracle Spatial, it may sometimes be necessary to explicitly tell it what **oracle_type** a particular element of the aggregate is. For example, a point cluster can be represented in the FME as a

line feature tagged as being an **oracle_point**; if this point cluster were contained in an **oracle_collection** without any indication that it is in fact a point, the FME would instead write it out as an **oracle_line**.

To allow for this sort of construct, the FME looks for attributes named **oracle_element{n}.oracle_type** when writing out a collection. The value of "n" is the position of the element within the aggregate, where the first element is numbered 0. Suppose that, in the above "point cluster" example, the cluster is in the third position of the aggregate, with a two lines on either side. That is, the aggregate contains five linear elements; the first two are lines, the next is the point cluster, and the remaining two are two more lines. The feature with the aggregate of these elements would contain the following attributes to tell the FME to write out the collection correctly:

Attribute Name	Attribute Value
oracle_type	oracle_collection
oracle_element{0}.oracle_type	oracle_line
oracle_element{1}.oracle_type	oracle_line
oracle_element{2}.oracle_type	oracle_point
oracle_element{3}.oracle_type	oracle_line
oracle_element{4}.oracle_type	oracle_line

Structured Geometry Representation (only applicable with classic geometry)

Using enhanced geometry removes the need for using this representation and makes it simple to translate complex oracle geometries into other formats, preserving the original geometry as it was stored in Oracle.

When reading from an Oracle 8i Spatial database using the object model, the STRUCTURED_GEOMETRY directive may be specified to keep the geometry structured in a way which mirrors the way it is represented within Oracle 8i's own structures. This accurate representation can be useful for advanced processing of Oracle geometry, but it stands in the way of normal FME feature processing, and is therefore disabled by default.

The representation of structured geometry is somewhat different from the normal feature representation, described above. The oracle_type attribute remains the same, but an additional attribute (oracle_subtype) provides more specific geometry information, and the geometry is often structured using FME aggregates.

The reason for this is that there may be several different representations for a given geometry type. For example, a "line" may be a sequence of linear segments, a sequence of connected arcs, or a combination of the two. The normal behavior of the Oracle 8i Spatial reader for such a feature would be to stroke out the arcs, and to make the resulting feature's a single line string with all of the coordinates of its parts. When the structured geometry mode is enabled, the result will be an aggregate containing line geometry to represent the lines, and points with extra attributes (the same attributes as oracle_arc features would normally have).

The oracle_subtype attribute indicates what geometry structure is used to represent the results. A complex linear feature such as the one describe above, for example, would have an oracle_subtype value of oracle_composite, indicating that it is represented by an fme_aggregate geometry.

When a structured geometry is an aggregate type, the features will often require attributes specific to a given element of the aggregate. These attributes are named oracle_element{N}.attrName, where N is the position of the element in the aggregate (starting at 0). In general, there will be an attribute named "oracle_element{N}.oracle_subtype" for every member of an aggregate that represents an oracle_line or oracle_area feature. Each feature representing a compound Oracle geometry types (oracle_multipoint, oracle_multiline, oracle_multipoly, and oracle_collection), will contain attributes "oracle_element{N}.oracle_type", because each element of the collection will be of a different type than the collection itself.

The following table summarizes all of the oracle_type and oracle_subtype values that are possible with structured geometry, and provides a description of each representation. Every structured geometry feature will have one of the oracle_type values listed in this table. Values such as oracle_arc, which are not listed in the oracle_type column, are represented as subtypes of another oracle_type.

oracle_type	oracle_subtype	Representation
oracle_nil	N/A	No geometry
oracle_point	oracle_point	Single point geometry (<i>fme_point</i>). As in the non-structured case, this is normally encoded into the SDO_POINT feature of the Oracle geometry object, but may be a single entry in the ordinate array.
	oracle_composite	Aggregate containing the points of the feature. This is used when the point's interpretation is greater than 1. (Note that it is not actually legal for an Oracle point geometry to contain more than a single point, but the SDO_GEOMETRY format technically supports this, so it representable in FME.)
oracle_line	oracle_line	Single line geometry (<i>fme_line</i>).
	oracle_arc	Point geometry (<i>fme_point</i>) representing the arc, as it would be represented when structured geometry is not being used. (i.e. The point is the centre point of the arc, and its shape is described by the attributes <i>oracle_primary_radius</i> , <i>oracle_secondary_radius</i> , <i>oracle_start_angle</i> , <i>oracle_sweep_angle</i> , and <i>oracle_rotation</i> .)
	oracle_composite	An aggregate geometry (<i>fme_aggregate</i>), each of whose elements has a subtype of <i>oracle_line</i> or <i>oracle_arc</i> .
oracle_area	oracle_polygon	A single polygon (<i>fme_polygon</i>) or donut (<i>fme_donut</i>) geometry.
	oracle_circle	Point geometry (<i>fme_point</i>) representing the circle, as it would be represented when structured geometry is not being used. (i.e. The point is the centre point of the circle, and its radius is stored in the attributes <i>oracle_radius</i> .)
	oracle_rectangle	A simple polygon (<i>fme_polygon</i>) containing five points defining the rectangle's boundary.
	oracle_line	A linear element (<i>fme_line</i>) that makes up a part of a boundary of a polygon or one of its holes. (Note that an

oracle_type	oracle_subtype	Representation
		<p>oracle_line element can only appear as a part of an oracle_composite representation, and cannot by themselves represent a whole polygon.)</p>
	oracle_arc	<p>A point element (fme_point) with the usual arc attributes, that makes up a part of a boundary of a polygon or one of its holes. (Note that an oracle_arc element can only appear as a part of an oracle_composite representation, and cannot by themselves represent a whole polygon.)</p>
	oracle_composite	<p>An aggregate (fme_aggregate) geometry containing a sequence of oracle_polygon, oracle_circle, oracle_rectangle, oracle_line, and oracle_arc elements. There will be an attribute called oracle_element{N}.oracle_subtype for each element within the aggregate, as well as attributes oracle_element{N}.attrName for any other attributes which pertain specifically to a particular element (such as a circle's radius).</p> <p>Consecutive oracle_line and oracle_arc elements which appear in an oracle_area composite are joined together to form closing "rings"; these rings form the boundaries of the polygon and its holes. (In Oracle 8.1.6 and later, outer boundaries are counter-clockwise, and hole boundaries are clockwise.)</p>
oracle_multipoint	N/A	<p>An aggregate geometry (fme_aggregate) containing one or more oracle_point elements. The feature will contain attributes named oracle_element{N}.oracle_type and oracle_element{N}.oracle_subtype for each point element of the aggregate. However, point clusters within the multipoint will not be split further into points and have oracle_type and oracle_subtype attributes.</p>

oracle_type	oracle_subtype	Representation
oracle_multiline	N/A	An aggregate geometry (fme_aggregate) containing one or more oracle_line elements. The feature will contain an attribute named oracle_element{N}.oracle_type for each element of the aggregate.
oracle_multipoly	N/A	An aggregate geometry (fme_aggregate) containing one or more oracle_area elements. The feature will contain an attribute named oracle_element{N}.oracle_type for each element of the aggregate.
oracle_collection	N/A	An aggregate geometry (fme_aggregate) containing one or more oracle_point , oracle_line , and oracle_area elements. The feature will contain an attribute named oracle_element{N}.oracle_type for each element of the aggregate.
oracle_raster	N/A	A raster geometry (fme_raster) containing pixels of various cell types.

GeoMedia Geometry Representation

Intergraph's GeoMedia stores additional information in Oracle Spatial geometry objects to represent oriented points and text features. FME's Oracle Spatial reader/writer has the ability to read and write these objects directly, without going through any GeoMedia code layers.

Both oriented points and text features appear in FME as normal Oracle point features, with additional attributes to describe the additional information. These will always be present on FME features read from Oracle tables where the GeoMedia information is present. If these attributes are defined on features being written to Oracle Spatial, the geometry written will include the GeoMedia extensions.

Each feature which has GeoMedia information includes an attribute named **geomedia_type**, whose value tells which kind of GeoMedia geometry information is attached. The following sections describe the two types of geometry.

Oriented Point

geomedia_type: geomedia_oriented_point

GeoMedia's oriented points add an orientation to the point. This can be specified on the FME feature either as a set of three numbers describing the orientation in 3D space, or a single value describing the planar rotation.

Attribute Name	Contents
geomedia_rotation	The orientation of the point, expressed in degrees counter-clockwise from the positive X axis. When reading, this is an approximation of the orientation described by the (i,j,k) orientation vector. When writ-

Attribute Name	Contents
	ing, this value is used to compute an (i,j,0) unit vector on the horizontal plane, if no such (i,j,k) vector is defined on the feature being written.
geomeadia_orient_i	The "X axis" component of a vector describing the point's orientation. This vector is normally a unit vector in 3D space. If the three components of the orientation vector are present on a feature being written, then the vector is used in place of any value present on the geomeadia_rotation attribute.
geomeadia_orient_j	The "Y axis" component of a vector describing the point's orientation. The comments above for geomeadia_orient_i apply to this attribute as well.
geomeadia_orient_k	The "Z axis" component of a vector describing the point's orientation. The comments above for geomeadia_orient_i apply to this attribute as well.

Text

geomeadia_type: geomeadia_oriented_text

GeoMedia's text features add a plaintext or RTF text string, orientation to the point. This can be specified on the FME feature either as a set of three numbers describing the orientation in 3D space, or a single value describing the planar rotation. Features read from Oracle will have both styles of orientation defined. In this case, the planar rotation is an approximation computed from the (i,j,k) orientation values.

Attribute Name	Contents
geomeadia_justification	The alignment of the text around the origin point's coordinates. Range: 0..2, 4..6, 8..10 0 centered vertically, centered horizontally 1 centered vertically, left of the origin 2 centered vertically, right of the origin 4 above the origin, centered horizontally 5 above the origin, left of the origin 6 above the origin, right of the origin 8 below the origin, centered horizontally 9 below the origin, left of the origin 10 below the origin, right of the origin Default: 9
geomeadia_text_string	The plain text version of the displayed text. Text stored as formatted text in GeoMedia will have an additional attribute on the FME feature to store the RTF version of the text, as well as one to specify the font size. When writing to Oracle, the RTF version of the text will

Attribute Name	Contents
	be used if it is supplied; otherwise an RTF string will be computed from the plain text string if a non-zero font size is specified.
geomedia_rtf_text_string	The formatted text string for the feature. Features stored as plain text in GeoMedia will not have an RTF text string attribute defined on them. When a feature with both plaintext and RTF information is written to Oracle, the RTF value will be stored into the resulting text feature; otherwise the plaintext value from <code>geomedia_text_string</code> will be combined with the font size from <code>geomedia_text_font_size</code> to compute a formatted text feature.
geomedia_text_font_size	The size of the formatted text feature, measured in text points. When reading, this value is extracted from the RTF text which defines the text feature. When writing, this value is used as the text size when creating an RTF string from the plaintext value in <code>geomedia_text_string</code> . If the feature being written contains a <code>geomedia_rtf_text_string</code> attribute, the font size is ignored. If the font size attribute has a zero (0) value, and no RTF text is supplied, the output text feature will be encoded as plain text. Note: GeoMedia features specify text in point size, whereas the rest of FME uses ground units for text size. The Oracle Spatial reader/writer does not attempt to translate between the two.
geomedia_text_font	The name of the font to be encoded into an RTF string for a text feature being written. If this isn't present, a default value of "Arial" is used. (This attribute is not supplied by the reader.)
geomedia_rotation	The rotation of the text, measured in degrees counter-clockwise from the horizontal.

Troubleshooting

Problems sometimes arise when attempting to connect to an Oracle database. This is almost always due to a misconfiguration in the user's environment. The following suggestions can often help detect and overcome such problems:

- For a table to be available to the Oracle Spatial reader, it requires an entry in the `USER_SDO_GEOM_METADATA` table defining the geometry column, spatial extents and optionally the SRID of the data.
- Ensure you can connect to the database with the service name, user name, and password using `SQL*Plus`.

- Ensure that you have the correct version of the Oracle client software installed. Oracle 8.1.5 or newer is recommended. Note that many clients have had problems if they have both 8.0.4 and 8.1.x installed on the same computer.
- Ensure that your `ORACLE_HOME` environment variable is correctly set: see the Oracle documentation for details. This is required for some specific versions of Oracle *8i*, and may be required even if `SQL*Plus` appears to operate correctly without it.
- If you have had older versions of the Oracle client software installed, make sure that your `PATH` variable has the current version's Oracle directory **first**, before any other Oracle software, including the WebDB package.
- It is sometimes helpful to define an environment variable named `ORACLE`, with the same value as the `ORACLE_HOME` variable. With some installations, it often helps to ensure that the variable named `ORACLE` is *not* defined.
- When running on UNIX, the following environment variables should be defined:

Variable	Contents	Sample Value
ORACLE_BASE	Top level of directory into which Oracle client software is installed.	<code>/opt2/oracle8i/app/oracle</code>
ORACLE_HOME	The Oracle product directory.	<code>/opt2/oracle8i/app/oracle/product.8.1.5</code>
ORACLE_SID	The system ID for the host's database instance.	<code>FME</code>
LD_LIBRARY_PATH	A list of directories which will be searched for shared objects. This list must include the <code>FME_HOME</code> path, as well as the <code>lib</code> sub-directory of <code>ORACLE_HOME</code> .	<code>\${LD_LIBRARY_PATH} : \${FME_HOME} : \${ORACLE_HOME}/lib</code>

- In most cases, the `ORACLE_SERVER_NAME` and `ORACLE_DATABASE` directives should be left with blank values, with the `ORACLE_DATASET` directive containing the Oracle service name of the database.

Mapping File Example

In this example, linear road features are extracted from Oracle Spatial and routed to MapInfo TAB files.

```
# -----
# Set up the Oracle Spatial reader

READER_TYPE ORACLE8I
ORACLE_SEARCH_ENVELOPE -128 49 -126 56.5
ORACLE_USER_NAME scott
ORACLE_PASSWORD tiger
ORACLE_DATASET worf

# -----
# Specify the relationship which must exist between
# the query region and the returned features.
# This is specified by a *type* of relationship, and a
# *result* to be tested. The relationship type is a
# string such as ANYINTERACT (the default) or TOUCH+INSIDE.
# The result is a simple test applied to determine whether
# the features exhibit the correct relationship.
```

```

# The default value of <> 'FALSE' will be true unless
# the relationship for a given pair does not exist;
# this is not the same as = 'TRUE' because (for example)
# a combined relationship like TOUCH+INSIDE will return
# a result of TOUCH, INSIDE, or FALSE.

```

```

ORACLE_INTERACTION CONTAINS
ORACLE_INTERACTION_RESULT = 'TRUE'

```

```

# -----
# Set up the mapinfo writer

```

```

WRITER_TYPE MAPINFO
MAPINFO_DATASET c:/temp/output

```

```

# -----
# Define the Oracle Spatial table we will read

```

```

ORACLE_DEF ROADS
ROADS_ID          float
NUMOFLANES       float
TYPE              varchar2(5)
UNDERCNST        varchar2(8)
DIVIDED          varchar2(8)
TRVLDIR          varchar2(6)
SDO_GID          float

```

```

MAPINFO_DEF ROADS
ROADS_ID          float
NUMOFLANES       float
TYPE              char(5)
UNDERCNST        char(8)
DIVIDED          char(8)
TRVLDIR          char(6)
SDO_GID          float

```

```

# -----
# Route the input Oracle Spatial data to the output TAB file

```

```

ORACLE ROADS
oracle_type      oracle_line
ROADS_ID         %ROADS_ID
NUMOFLANES       %NUMOFLANES
TYPE             %TYPE
UNDERCNST        %UNDERCNST
DIVIDED          %DIVIDED
TRVLDIR          %TRVLDIR
SDO_GID          %SDO_GID

```

```

MAPINFO ROADS
mapinfo_type     mapinfo_polyline
ROADS_ID         %ROADS_ID
NUMOFLANES       %NUMOFLANES
TYPE             %TYPE
UNDERCNST        %UNDERCNST
DIVIDED          %DIVIDED
TRVLDIR          %TRVLDIR
SDO_GID          %SDO_GID

```

Oracle Spatial Relational Reader/Writer

Format Notes:

This format is not supported by FME Base Edition.

ORACLE® VERSION:

- Any references to Oracle 8*i* throughout this chapter are also applicable to Oracle 9*i* and Oracle 10*g*.

Overview

The Oracle Spatial Relational Reader/Writer module enables FME to read and write geometric and attribute data stored using Oracle Spatial. This module communicates directly with Oracle Spatial for maximum throughput. Both the relational and object-relational models of Oracle Spatial are supported by FME: the relational model is discussed here and the object-relational model is discussed in a separate chapter – Oracle Spatial Object Reader/Writer.

If only attributes are to be read or written, then the Database reader and writer module of FME should be used. In addition, an OracleQueryFactory is available to extract data from an Oracle Spatial database within the FME factory pipeline.

This chapter assumes familiarity with Oracle Spatial, the geometry types it supports, and its indexing mechanisms.

Tip:

See the QueryFactory in the FME Functions and Factories manual. This factory also exploits the powerful query capabilities of Oracle Spatial.

See the @SQL function, also in the FME Functions and Factories manual. This function allows arbitrary Structured Query Language (SQL) statements to be executed against any Oracle database.

Oracle Spatial Relational Quick Facts

Format Type Identifier	ORACLE
Reader/Writer	Both
Licensing Level	Professional
Dependencies	None
Dataset Type	Database
Feature Type	Table name
Typical File Extensions	N/A
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	Optional
Spatial Index	Yes
Schema Required	Yes
Transaction Support	No
Enhanced Geometry	Yes
Encoding Support	Yes
Geometry Type	oracle_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	yes
elliptical arc	no		surface	yes
ellipses	no		text	no
line	yes		z values	yes
none	yes			

Reader Overview

FME considers an Oracle Spatial dataset to be a collection of relational tables together with their geometry. The tables must be defined in the mapping file before they can be read. Arbitrary WHERE clauses and joins are fully supported.

Reader Directives

The directives listed below are processed by the Oracle Spatial reader. The suffixes listed are prefixed by the current <ReaderKeyword> in a mapping file. By default, the <ReaderKeyword> for the Oracle Spatial reader is ORACLE .

DATASET

Required/Optional: *Required*

This specifies the SQL/Net service name for the Oracle Spatial database, which can be blank to use the default service. If it is specified, then the service must have been set up in the local SQL/Net configuration.

```
ORACLE_DATASET citySource
```

Workbench Parameter: *Source Oracle Spatial Relational Service*

USER_NAME

Required/Optional

Optional

Mapping File Syntax

```
ORACLE_USER_NAME bond007
```

If the database is configured to use an external authentication adapter (such as Windows NT or Kerberos authentication), the username may be left blank, or may be completely omitted.

If a connection cannot be established using the provided username, a second attempt will be made using the uppercase version of the username.

* Workbench Parameter

Username

PASSWORD

Required/Optional: *Required*

The password of the user accessing the database.

```
ORACLE_PASSWORD moneypenny
```

If the database is configured to use an external authentication adapter (such as Windows NT or Kerberos authentication), the password may be left blank, or may be completely omitted.

Workbench Parameter: *Password*

DEF

Required/Optional: *Optional*

The syntax of the definition is:

```
ORACLE_DEF <layerName> \  
    [oracle_envelope_min_x <xmin>] \  
    [oracle_envelope_min_y <ymin>] \  
    [oracle_envelope_max_x <xmax>] \  
    [oracle_envelope_max_y <ymax>] \  
    [oracle_interaction <interactionType>] \  
    [oracle_interaction_result <interactionQualifier>] \  
    [oracle_where_clause_encoded <whereClause>] \  
    [oracle_sql_encoded <sqlQuery>] \  
    [oracle_dim <dim>] \  
    [<fieldName> <fieldType>] +
```

The <fieldType> of each field must be given, but it is not verified against the database definition for the field. In effect, it is ignored.

The <layerName> must match an Oracle Spatial relational geometry layer in the database. This will be used as the feature type of all the features read from the table. If the <tableName> does not match a table in the database, a second attempt will be made using the uppercase version of the <tableName>.

The layer definition allows specification of separate search parameters for each layer. If any of the configuration parameters are given, they will override, for that layer, whatever global values have been specified by the reader keywords listed in the above table. If any of these parameters is not specified, the global values will be used.

The following table summarizes the definition line configuration parameters:

Parameter	Contents
oracle_envelope _minx oracle_envelope _miny oracle_envelope _maxx oracle_envelope _maxy	These specify the spatial extent of the features to be read from the layer. If these are not all specified, the values from the <ReaderKeyword>_SEARCH_ENVELOPE directive are used.
oracle_interaction	This specifies the spatial interaction type to be tested for this layer. If this is not specified, the value of the <ReaderKeyword>_INTERACTION directive is used.
oracle_interaction _result	This specifies the required result of the spatial interaction comparison performed for this layer. If this is not specified, the value of the <ReaderKeyword>_INTERACTION_RESULT directive is used.
oracle_where_clause_encoded	This specifies the SQL WHERE clause applied to the attributes of the layer's features to limit the set of features returned. If this is not specified, the value of the <ReaderKeyword>_WHERE_CLAUSE directive is used. This parameter is encoded as described in the section <i>Substituting Strings in Mapping Files</i> in FME Fundamentals help > Mapping File Syntax.
oracle_sql_encoded	This specifies an SQL SELECT query to be used as the source for the results. If this is specified, the Oracle Spatial reader will execute the query, and use the resulting rows as the features instead of reading from the table <layerName>. All returned features will have a feature type of <layerName>, and attributes for all columns selected by the query. The oracle_where_clause_encoded and all parameters which specify a spatial constraint – oracle_envelope_minx, oracle_interaction, and so on – are ignored if oracle_sql is supplied. This parameter is encoded as described in the section <i>Substituting Strings in Mapping Files</i> in FME Fundamentals help > Mapping File Syntax.
oracle_dim	This specifies the number of dimensions (2 or 3) for the table's geometry. This is required only if the oracle_sql_encoded parameter is used to specify a SELECT query.

If no `<whereClause>` is specified, all rows in the table will be read and returned as individual features. If a `<whereClause>` is specified, only those rows which are selected by the clause will be read. Note that the `<whereClause>` does not include the word **WHERE**.

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined database tables files that will be read. If no **IDs** are specified, then all defined and available tables are read. The syntax of the **IDs** keyword is:

```
ORACLE_IDS <featureType1> \  
           <featureType2> ... \  
           <featureTypeN>
```

The feature types must match those used in **DEF** lines.

The example below selects only the **ROADS** table for input during a translation:

```
ORACLE_IDS ROADS
```

Workbench Parameter: *Feature Types to Read*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

Mapping File Example

The example below selects a small area for extraction:

```
ORACLE_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The `COORDINATE_SYSTEM` directive, which specifies the coordinate system associated with the data to be read, must always be set if the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` to the reader `COORDINATE_SYSTEM` prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]

* Workbench Parameter

Clip To Envelope

INTERACTION

Required/Optional: *Optional*

This specifies the type of relationship which must exist between the search envelope and the geometry in the target layer. Any supported relationship, or combination of relationships, may be specified.

This table lists the valid geometry interaction relationships.

Search Method	Description
DISJOINT	Returns DISJOINT if the objects have no common boundary or interior points; otherwise, returns FALSE.
EQUAL	Returns EQUAL if the objects share every point of their boundaries and interior, including any holes in the objects; otherwise, returns FALSE.
INSIDE	Returns INSIDE if the geometry feature is entirely contained within the query feature; otherwise, returns FALSE.
OVERLAPBDYDISJOINT	Returns OVERLAPBDYDISJOINT if the objects overlap, but their boundaries do not interact; otherwise, returns FALSE.
OVERLAPBDYINTERSECT	Returns OVERLAPBDYINTERSECT if the objects overlap, and their boundaries intersect in one or more places; otherwise, returns FALSE.

Search Method	Description
TOUCH	Returns TOUCH if the two objects share a common boundary point, but no interior points; otherwise, returns FALSE.
ANYINTERACTION	Returns TRUE if the objects interact according to any of the above relationships; otherwise, returns FALSE.

In addition to specifying a single relationship, one may specify a combination of relationships to be tested by concatenating them with a plus sign (+). For example, if the `<ReaderKeyword>_INTERACTION` is specified as `INSIDE + TOUCH`, the result of the interaction test will be one of: `INSIDE, TOUCH`, or `FALSE`.

INTERACTION_RESULT

Required/Optional: *Optional*

This specifies the test that is applied to the results of the above geometry relationship comparison. When using the relational model, the name for type of relationship—except in the case of `ANYINTERACTION`—is returned from Oracle for a positive match, rather than a value of `TRUE`. For this reason, the default test for the interaction result is `"<>'FALSE'"` rather than (the perhaps more intuitive) `"='TRUE'"`, when using the relational model.

For combined relationships in the relational model, one might want to use a comparison such as `"='TOUCH'"` instead of the default.

WHERE_CLAUSE

Required/Optional: *Optional*

This specifies an SQL `WHERE` clause, which is applied to the table's columns to limit the resulting features. This feature is currently limited to apply only to the attributes of the target Spatial layer, and does not allow for joining multiple tables together. The effect of table joins can be achieved using the object model, by specifying the entire queries in the `DEF` line with an `oracle_sql_encoded` parameter.

By default, there is no `WHERE` clause applied to the results, so all features in the layer are returned.

Workbench Parameter: *Where Clause*

QUERY_SUFFIX

Required/Optional: *Optional*

The Oracle reader performs a spatial query when it has to find geometry which interacts with a given search envelope. When performing the spatial query in relational mode, the extent of the search is written out to an Oracle Spatial layer which is separate from the layer of geometry being queried. The name of this query layer is generated by appending a suffix to the name of the layer being queried; for example, `ROADS_QUERY`. This directive specifies the part that is appended to the geometry layer name to form the query layer name. In the preceding example, the query suffix would be specified as `_QUERY`.

If this is not specified, a default suffix of `_FMESQ` will be used.

CHUNK_SIZE

Required/Optional: *Optional*

The geometry is read from the Oracle database using a bulk reading technique to maximize performance. Normally 1000 rows of data are read from the database at a time.

This directive allows one to tune the performance of the reader. It specifies how many rows are read from the database at a time.

Workbench Parameter: *Max Features to Read*

BEGIN_SQL{n}

Occasionally you must execute some ad-hoc SQL prior to opening a table. For example, it may be necessary to ensure that a view exists prior to attempting to read from it.

Upon opening a connection to read from a database, the reader looks for the directive `<ReaderKeyword>_BEGIN_SQL{n}` (for $n=0,1,2,\dots$), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the `FME_SQL_DELIMITER` keyword, embedded at the beginning of the SQL block. The single character following this keyword will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;  
DELETE FROM instructors;  
DELETE FROM people  
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

*** Workbench Parameter**

SQL Statement to Execute Before Translation

END_SQL{n}

Occasionally you must execute some ad-hoc SQL after closing a set of tables. For example, it may be necessary to clean up a temporary view after writing to the database.

Just before closing a connection on a database, the reader looks for the directive `<ReaderKeyword>_END_SQL{n}` (for $n=0,1,2,\dots$), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the `FME_SQL_DELIMITER` directive, embedded at the beginning of the SQL block. The single character following this directive will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;  
DELETE FROM instructors;  
DELETE FROM people  
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

* Workbench Parameter

SQL Statement to Execute After Translation

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The Oracle Spatial writer module stores both geometry and attributes into an Oracle Spatial database. Only uppercase table names are supported.

The Oracle Spatial writer provides the following capabilities:

1. **Transaction Support:** The Oracle Spatial writer provides transaction support that eases the data loading process. Occasionally, a data load operation terminates prematurely due to data difficulties. The transaction support provides a mechanism for reloading corrected data without data loss or duplication.
2. **Table Creation:** The Oracle Spatial writer uses the information within the FME mapping file to automatically create database tables as needed. When the relational model of storing geometry is used, the writer will create and populate all needed supporting tables. In such a case, the `<layername>`, `<layername>_SDOGEOM`, `<layername>_SDOLAYER`, `<layername>_SDOINDEX`, and `<layername>_SDODIM` tables will all be created.
3. **Index Creation:** The Oracle Spatial writer will set up and populate all needed indexes and index tables as part of the loading process. For the relational model, indexes on `SDO_GID` columns in the `<layername>` and `<layername>_SDOGEOM` tables are created, and a compound index on the `SDO_GID` and `SDO_CODE` columns in the `<layername>_SDOINDEX` is created. The `<layername>_SDOINDEX` table will also be populated.
4. **Bulk Loading:** The Oracle Spatial writer uses a bulk loading technique to ensure speedy data load.

Writer Directives

The directives processed by the Oracle Spatial writer are listed below. The suffixes shown are prefixed by the current `<writerKeyword>` in a mapping file. By default, the `<writerKeyword>` for the Oracle Spatial writer is `ORACLE` when using the relational model.

DATASET, USER_NAME, BEGIN_SQL{ }, and END_SQL{ }

These directives operate in the same manner as they do for the Oracle Spatial reader.

DEF

Required/Optional: *Required*

Each Oracle Spatial layer (table) must be defined before it can be written. The general form of an Oracle Spatial definition statement is:

```
ORACLE_DEF <tableName> \
  [oracle_model      (relational|enhanced_relational)] \
  [oracle_min_gid    <minGid>] \
  [oracle_num_ords   <numOrds>] \
  [oracle_dim        <dim>] \
  [oracle_srid       <spatialReference>] \
  [oracle_levels     <levels>] \
  [oracle_ord1name   <ord1>] \
  [oracle_ord2name   <ord2>] \
  [oracle_ord3name   <ord3>] \
  [oracle_x_to1      <xto1>] \
  [oracle_y_to1      <yto1>] \
  [oracle_z_to1      <zto1>] \
  [oracle_x_col_type <xtype>] \
  [oracle_y_col_type <ytype>] \
  [oracle_z_col_type <ztype>] \
  [oracle_code_size  <codeSize>] \
  [oracle_min_x      <xmin>] \
  [oracle_min_y      <ymin>] \
  [oracle_min_z      <zmin>] \
  [oracle_max_x      <xmax>] \
  [oracle_max_y      <ymax>] \
  [oracle_max_z      <zmax>] \
  [oracle_create_indices (yes|no|incremental)] \
  [oracle_index_commit_interval <interval>] \
  [oracle_gid_name   <gidName>] \
  [<fieldName>      <fieldType>]*
```

The table definition allows complete control of the layer that will be created. If the layer already exists, the majority of the oracle_ parameters will be ignored and need not be given. As well, if the table already exists in the database, then it is not necessary to list the fields and their types – FME will use the schema information in the database to determine this. If the fields and types are listed, they must match those in the database, however, not all fields must be listed.

If the table does not exist, then the field names and types are used to first create the table. In any case, if a **<field-Type>** is given, it may be any field type supported by the target database.

The configuration parameters present on the definition line are described in the following table:

Parameter	Contents
oracle_model	This indicates which model for storing geometry should be used. This parameter should normally be specified with its default value of "relational" when working with the relational model. It may optionally be specified as "enhanced_relational", which places the writer into a mode which supports the additional features described in the section entitled Enhanced Relational Operation.
oracle_min_gid	This specifies the minimum geometric identifier (GID) that should be used. The default is the greater of 0, or the largest GID present in the layer. This is only used by

Parameter	Contents
	the relational model of Oracle Spatial.
oracle_num_orcs	This specifies the number of ordinates that will be present in the <code><layername>_SDOGEOM</code> table. The default is 64. This is only used by the relational model of Oracle Spatial.
oracle_dim	This specifies the dimension of the layer, which can be 2 or 3. The default is 2.
oracle_levels	This specifies the number of tessellation levels used to create the spatial index for the layer. The larger the number, the longer spatial index creation will take but the finer the granularity of the index. The range is any integer between 1 and 64 for the quadtree algorithm. If you want to use the R-Tree algorithm, set the value to 0. For Oracle 11g users, it is highly recommended to use the R-Tree algorithm. Oracle 10g users should use the quadtree algorithm. The default is 1.
oracle_ord1name	This specifies the name to use for the first ordinate. This name is used when the <code><layername>_SDOGEOM</code> table is created. The default is X. This is only used by the relational model of Oracle Spatial.
oracle_ord2name	This specifies the name to use for the second ordinate. This name is used when the <code><layername>_SDOGEOM</code> table is created. The default is Y. This is only used by the relational model of Oracle Spatial.
oracle_ord3name	This specifies the name to use for the third ordinate. This name is used when the <code><layername>_SDOGEOM</code> table is created. The default is Z. This is only used by the relational model of Oracle Spatial.
oracle_x_tol	This specifies the comparison tolerance for the x coordinates. Coordinates in x that are closer than this value are considered equal. The default is 0.000000005.
oracle_y_tol	This specifies the comparison tolerance for the y coordinates. Coordinates in x that are closer than this value are considered equal. The default is 0.000000005.
oracle_z_tol	This specifies the comparison tolerance for the z coordinates. Coordinates in x that are closer than this value are considered equal. The default is 0.000000005.

Parameter	Contents
oracle_x_col_type	This specifies the column type in the <layername>_SDOGEOM for the x coordinates. The default is float. This is only used by the relational model of Oracle Spatial.
oracle_y_col_type	This specifies the column type in the <layername>_SDOGEOM for the y coordinates. The default is float. This is only used by the relational model of Oracle Spatial.
oracle_z_col_type	This specifies the column type in the <layername>_SDOGEOM for the z coordinates. The default is float. This is only used by the relational model of Oracle Spatial.
oracle_code_size	<p>This specifies the size of the sdo code columns in the <layername>_SDOINDEX table. The default is 30.</p> <p>The optimal value for this can be calculated by executing this SQL procedure in SQLPlus:</p> <pre> set serveroutput on declare sz integer; begin sz := sdo_admin.sdo_code_size ('FOREST'); dbms_output.put_line ('VALUE is ' sz); end; </pre> <p>This is only used by the relational model of Oracle Spatial.</p>
oracle_min_x	<p>The minimum x value expected in the dataset. If any x values are present that are less than this value, the spatial index will give undefined results. For best spatial search performance, this value should be as close to the true minimum x as possible.</p> <p>This parameter must be specified.</p>
oracle_min_y	<p>The minimum y value expected in the dataset. If any y values are present that are less than this value, the spatial index will give undefined results. For best spatial search performance, this value should be as close to the true minimum y as possible.</p> <p>This parameter must be specified.</p>
oracle_min_z	<p>The minimum z value expected in the dataset. In the current release of Oracle Spatial, no indexing is done on the z axis, so the value can be arbitrarily assigned.</p> <p>This parameter must be specified if the dimension of the layer is 3.</p>
oracle_max_x	<p>The maximum x value expected in the dataset. If any x values are present that are greater than this value, the</p>

Parameter	Contents
	<p>spatial index will give undefined results. For best spatial search performance, this value should be as close to the true maximum x as possible.</p> <p>This parameter must be specified.</p>
oracle_max_y	<p>The maximum y value expected in the dataset. If any y values are present that are greater than this value, the spatial index will give undefined results. For best spatial search performance, this value should be as close to the true maximum y as possible.</p> <p>This parameter must be specified.</p>
oracle_max_z	<p>The maximum z value expected in the dataset. In the current release of Oracle Spatial, no indexing is done on the z axis, so the value can be arbitrarily assigned.</p> <p>This parameter must be specified if the dimension of the layer is 3.</p>
oracle_create_indices	<p>This indicates whether or not indices are to be created as part of the data load. The valid choices are yes, no or incremental.</p> <p>If yes or incremental is specified, attribute indices on SDO_GID columns in the <layename> and <layename>_SDOGEOM tables are created, and a compound index on the SDO_GID and SDO_CODE columns in the <layename>_SDOINDEX is created.</p> <p>If yes is specified, the <layename>_SDOINDEX table will be populated using the SDO_ADMIN.POPULATE_INDEX function.</p> <p>If incremental is specified, the <layename>_SDOINDEX table will be populated for only the features just loaded using the SDO_ADMIN.UPDATE_INDEX function.</p> <p>If no is specified, no index creation is done.</p> <p>The default is no.</p>
oracle_index_commit_interval	<p>When incremental index creation is used, a commit will be performed each time the number of features specified here have been indexed.</p> <p>The default is 50.</p>
oracle_gid_name	<p>When the FME opens up MYTABLE for writing, it performs an SQL query to determine the highest value of MY_ID in the table MYTABLE. If this value is 34, the next row written by the FME will place a value of 35 into MY_ID and then 36 for the next row, etc. (If the table contains no data, the first row for the oracle_gid_name column will be given a value of 1.) It is important that the variable named by oracle_gid_name be an actual row on the DEF line. If the above</p>

Parameter	Contents
	<p>DEF line read:</p> <pre>ORACLE_DEF MYTABLE \ oracle_gid_name MY_ID \ GEOM GEOMETRY \ NAME VARCHAR2(20)</pre> <p>then MY_ID would never actually be assigned a column in the table.</p> <p>Note: If a feature to be written to an Oracle Spatial relational table contains an attribute with the same name as the GID column, the Oracle writer will assume that the attribute's value should be used when writing the feature to the database. No check will be made to determine whether this value conflicts with any existing values in the table's GID column.</p>

START_TRANSACTION

Required/Optional: *Optional*

This statement tells the Oracle Spatial writer module when to start actually writing features into the database. The Oracle Spatial writer does not write any features until the feature is reached that belongs to **<last successful transaction> + 1**. Specifying a value of zero causes every feature to be output. Normally, the value specified is zero—a non-zero value is only specified when a data load operation is being resumed after failing partway through.

Parameter	Contents
<last successful transaction>	The transaction number of the last successful transaction. When loading data for the first time, set this value to 0 .

Example:

```
ORACLE_START_Transaction 0
```

TRANSACTION_INTERVAL

Required/Optional: *Optional*

This statement informs the FME about the number of features to be placed in each transaction before a transaction is committed to the database.

If the **ORACLE_TRANSACTION_INTERVAL** statement is not specified, then a value of 2000 is used as the transaction interval.

Parameter	Contents
<transaction_interval>	The number of features in a single transaction.

Example:

```
ORACLE_Transaction_INTERVAL 5000
```

Feature Representation

Features read from Oracle Spatial consist of a series of attribute values and geometry. The feature type of each Database feature is as defined on its **DEF** line.

Features written to the database have the destination table as their feature type, and attributes as defined by on the **DEF** line.

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), the Oracle Spatial module makes use of the following special attribute names:

Attribute Name	Contents
oracle_type	The type of geometric entity stored within the feature. The valid values for both the relational and object model are listed below: oracle_nil oracle_point oracle_line oracle_area

Features read from, or written to, Oracle Spatial also have an attribute for each column in the database table. The feature attribute name will be the same as the source or destination column name.

The attribute and column names are case-sensitive.

No Coordinates

oracle_type: oracle_nil

Features with no coordinates are tagged with this value when reading or writing to or from Oracle Spatial.

Points

oracle_type: oracle_point

Features tagged with this value consist of a single point or an aggregate of points. When the object model is being used, an aggregate or line geometry tagged as **oracle_point** will be written as a "point cluster" with several coordinates, rather than being written as a single coordinate, or coerced to being an **oracle_line**.

Lines

oracle_type: oracle_line

Linear features are tagged with this value when reading or writing to or from Oracle Spatial. Both single part and aggregate linear features are supported.

When the object model is being used, aggregates are written out as "multiline" geometry containing several linear elements, just as if the feature had been tagged with oracle_multiline. Any non-linear elements contained in the aggregate are discarded.

Areas

oracle_type: oracle_area

Area features are tagged with this value when reading or writing to or from Oracle Spatial. Both single part and aggregate area features are supported. An area feature may be either a polygon or a donut polygon. Note that checking is done to ensure that the area features adhere to the geometry rules of Oracle Spatial as they are loaded.

When the object model is being used, aggregates are written out as “multipolygon” geometry containing several polygonal elements, just as if the feature had been tagged with `oracle_multiline`. Any non-polygonal elements contained in the aggregate are discarded.

Enhanced Relational Operation

FME normally writes to Oracle Spatial relational databases according to Oracle’s definition of the format. In this definition, user attributes for the layer `<layerName>` go into the table named `<layerName>`, and the geometry for a given feature is stored in one or more rows in the table named `<layerName>_SDOGEOM`.

Some GIS systems, such as Intergraph’s G/Technology, store user attributes in the `<layerName>_SDOGEOM` table in addition to the normal geometry definition. To accommodate such systems, the Oracle Spatial (Relational) writer can operate in an “enhanced” mode, which is triggered by setting the table’s `DEF` line’s `oracle_model` parameter to a value of “`enhanced_relational`”. In the “enhanced” mode, each feature written is searched for attributes which match the names of the additional columns in the `<layerName>_SDOGEOM` table. Any matching attributes are written to their respective columns of the table, on every row used to define the geometry.

One complication when operating in the `enhanced_relational` mode is that there may be an attribute in the `<layerName>_SDOGEOM` table that has the same name as an attribute in `<layerName>`’s user attribute table, but is to have a different value. To overcome this complication, the enhanced relational writer searches each feature for attributes named `SDOGEOM.<attrName>`; if found, each such attribute’s value is placed into the corresponding column `<attrName>` of the `<layerName>_SDOGEOM` table.

Troubleshooting

Problems sometimes arise when attempting to connect to an Oracle database. This is almost always due to a misconfiguration in the user’s environment. The following suggestions can often help detect and overcome such problems.

- Ensure you can connect to the database with the service name, user name, and password using `SQL*Plus`.
- Ensure that you have the correct version of the Oracle client software installed. Oracle 8.1.5 or newer is recommended. Note that many clients have had problems if they have both 8.0.4 and 8.1.x installed on the same computer.
- Ensure that your `ORACLE_HOME` environment variable is correctly set—see the Oracle documentation for details on this. This is required for some specific versions of Oracle 8i, and may be required even if `SQL*Plus` appears to operate correctly without it.
- If you have had older versions of the Oracle client software installed, make sure that your `PATH` variable has the current version’s Oracle directory **first**, before any other Oracle software, including the WebDB package.
- It is sometimes helpful to define an environment variable named `ORACLE`, with the same value as the `ORACLE_HOME` variable. With some installations, it often helps to ensure that the variable named `ORACLE` is *not* defined.
- When running on UNIX, the following environment variables should be defined:

Variable	Contents	Sample Value
<code>ORACLE_BASE</code>	Top level of directory into which Oracle client software is installed.	<code>/opt2/oracle8i/app/oracle</code>
<code>ORACLE_HOME</code>	The Oracle product directory.	<code>/opt2/oracle8i/app/oracle/product.8.1.5</code>
<code>ORACLE_SID</code>	The system ID for the host’s database instance.	<code>FME</code>

Variable	Contents	Sample Value
LD_LIBRARY_PATH	A list of directories which will be searched for shared objects. This list must include the FME_HOME path, as well as the lib sub-directory of ORACLE_HOME .	<code>\${LD_LIBRARY_PATH}:\${FME_HOME}:- \${ORACLE_HOME}/lib</code>

- In most cases, the **ORACLE_SERVER_NAME** and **ORACLE_DATABASE** keywords should be left with blank values, with the **ORACLE_DATASET** keyword containing the Oracle service name of the database.

Mapping File Examples

In this example, a Shapefile containing linear features representing rivers is imported into Oracle Spatial, using the relational model to store the geometry.

```
# -----
# Set up the shape reader

READER_TYPE SHAPE
SHAPE_DATASET /usr/data/shapes

# -----
# Set up the ORACLE writer, starting at transaction 0

WRITER_TYPE ORACLE
ORACLE_USER_NAME scott
ORACLE_PASSWORD tiger
ORACLE_DATASET worf
ORACLE_TRANSACTION 0

# -----
# Define the original shape file

SHAPE_DEF RIVER \
  SHAPE_GEOMETRY      shape_polyline \
  MOEPCODE            char(10) \
  DEPTH              number(5,0)

# -----
# Define the output Oracle spatial table

ORACLE_DEF RIVER \
  oracle_create_indices incremental \
  oracle_index_commit_interval 50 \
  oracle_levels        5 \
  oracle_min_x         -128 \
  oracle_min_y         49 \
  oracle_max_x         -126 \
  oracle_max_y         60 \
  oracle_model         relational \
  oracle_x_col_type    FLOAT \
  oracle_y_col_type    FLOAT \
  oracle_z_col_type    FLOAT \
  oracle_dim           2 \
  oracle_gid_name      SDO_GID \
  MOEPCODE             varchar2(10) \
  DEPTH               float

# -----
# Route the input shape data to the output Oracle database
```

```

SHAPE RIVER \
  MOEPCODE %1 \
  DEPTH %2
ORACLE RIVER \
  oracle_type oracle_line \
  MOEPCODE %1 \
  DEPTH %2

```

In this example, linear road features are extracted from Oracle Spatial and routed to MapInfo TAB files.

```

# -----
# Set up the Oracle spatial reader

READER_TYPE ORACLE
ORACLE_SEARCH_ENVELOPE -128 49 -126 56.5
ORACLE_USER_NAME scott
ORACLE_PASSWORD tiger
ORACLE_DATASET worf

# -----
# Specify the relationship which must exist between
# the query region and the returned features.
# This is specified by a *type* of relationship, and a
# *result* to be tested. The relationship type is a
# string such as ANYINTERACT (the default) or TOUCH+INSIDE.
# The result is a simple test applied to determine whether
# the features exhibit the correct relationship.
# The default value of <> 'FALSE' will be true unless
# the relationship for a given pair does not exist;
# this is not the same as = 'TRUE' because (for example)
# a combined relationship like TOUCH+INSIDE will return
# a result of TOUCH, INSIDE, or FALSE.

ORACLE_INTERACTION CONTAINS
ORACLE_INTERACTION_RESULT <> 'FALSE'

# -----
# Set up the mapinfo writer

WRITER_TYPE MAPINFO
MAPINFO_DATASET c:/temp/output

# -----
# Define the Oracle spatial table we will read

ORACLE_DEF ROADS
ROADS_ID float \
NUMOFLANES float \
TYPE varchar2(5) \
UNDERCNST varchar2(8) \
DIVIDED varchar2(8) \
TRVLDIR varchar2(6) \
SDO_GID float

MAPINFO_DEF ROADS
ROADS_ID float \
NUMOFLANES float \
TYPE char(5) \
UNDERCNST char(8) \
DIVIDED char(8) \
TRVLDIR char(6) \
SDO_GID float

# -----
# Route the input Oracle spatial data to the output TAB file

```

ORACLE ROADS
oracle_type
ROADS_ID
NUMOFLANES
TYPE
UNDERCNST
DIVIDED
TRVLDIR
SDO_GID

oracle_line
%ROADS_ID
%NUMOFLANES
%TYPE
%UNDERCNST
%DIVIDED
%TRVLDIR
%SDO_GID

MAPINFO ROADS
mapinfo_type
ROADS_ID
NUMOFLANES
TYPE
UNDERCNST
DIVIDED
TRVLDIR
SDO_GID

mapinfo_polyline
%ROADS_ID
%NUMOFLANES
%TYPE
%UNDERCNST
%DIVIDED
%TRVLDIR
%SDO_GID

Oracle SQL Loader Writer

Overview

Oracle® SQL Loader files are text files formatted for loading into relational databases. In particular, the format of these files works well with the SQL Loader utility supplied by Oracle for use with Oracle databases.

For more information, see the *Relational Table* chapter, DEF section.

Notes:

This type of file can only be written – it is not possible to read these files.

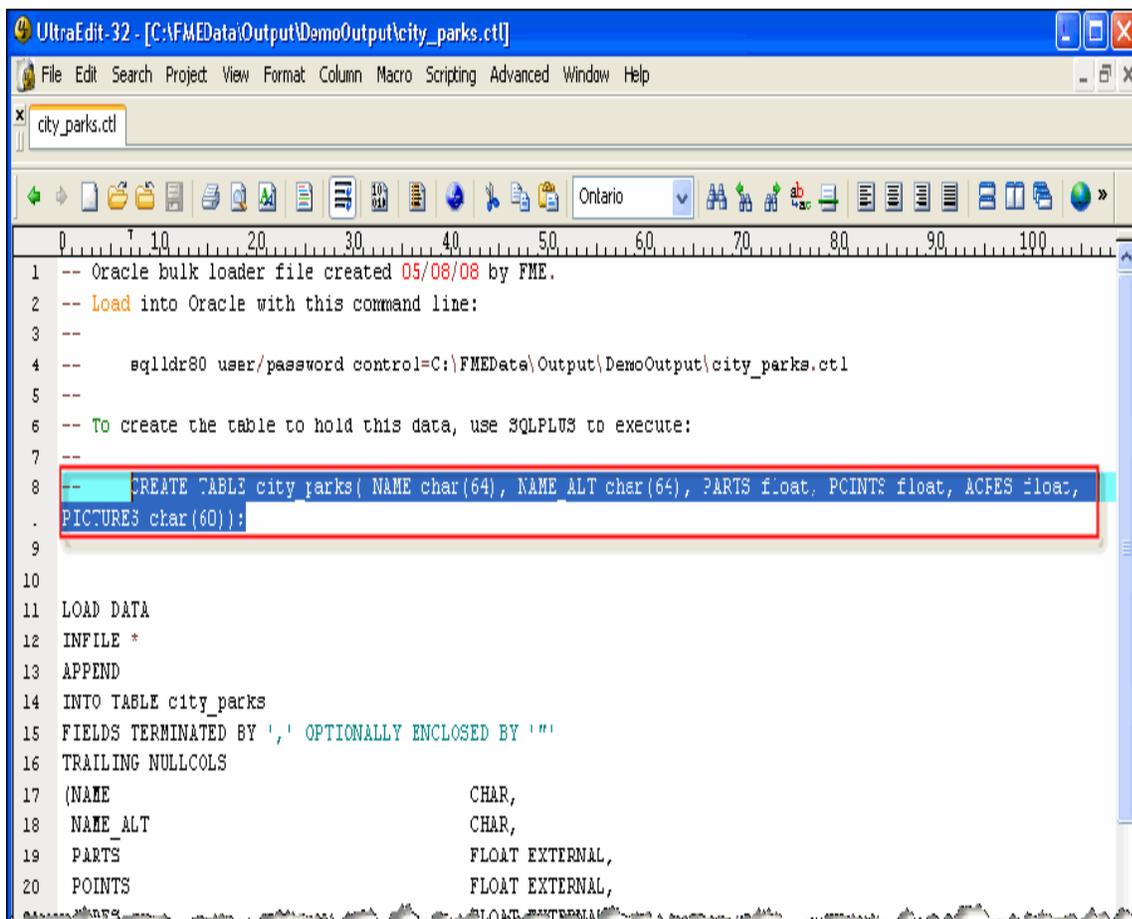
The Oracle SQL Loader writer does not support spatial data.

The Oracle SQL Loader writer does not create the required table when the ctl file is run. See the paragraph below for instructions on creating a table.

Creating a Table via the CTL File

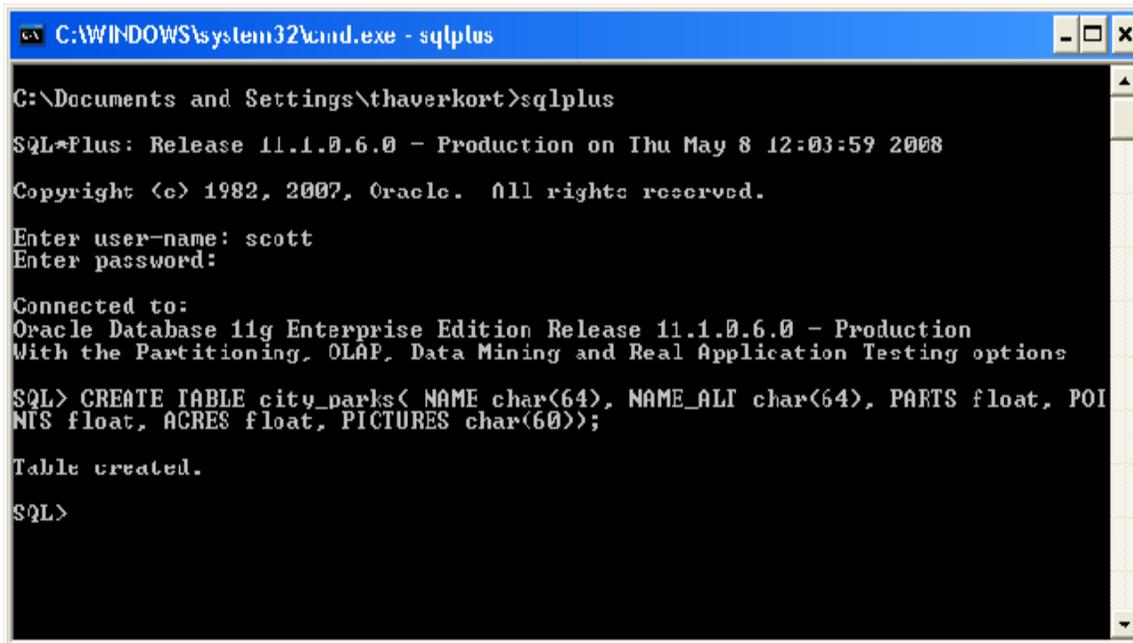
The destination Oracle table must exist prior to using SQLLoader to load the CTL file. FME will not create the required table.

If the table does not exist, then it can be created by opening the CTL file that FME generated. In the comments section is the SQL statement that will create the table with the correct structure.



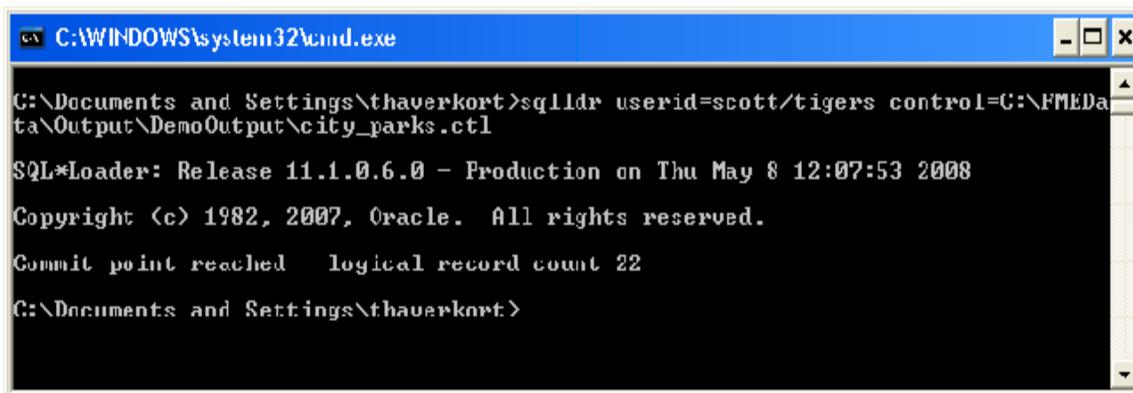
```
UltraEdit-32 - [C:\FMEData\Output\DemoOutput\city_parks.ctl]
File Edit Search Project View Format Column Macro Scripting Advanced Window Help
x city_parks.ctl
0 10 20 30 40 50 60 70 80 90 100
1 -- Oracle bulk loader file created 05/08/08 by FME.
2 -- Load into Oracle with this command line:
3 --
4 --   sqlldr80 user/password control=C:\FMEData\Output\DemoOutput\city_parks.ctl
5 --
6 -- To create the table to hold this data, use SQLPLUS to execute:
7 --
8 -- CREATE TABLE city_parks( NAME char(64), NAME_ALT char(64), PARTS float, PCINTS float, ACRES float,
9 -- PICTURES char(60));
10
11 LOAD DATA
12 INFILE *
13 APPEND
14 INTO TABLE city_parks
15 FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
16 TRAILING NULLCOLS
17 (NAME CHAR,
18 NAME_ALT CHAR,
19 PARTS FLOAT EXTERNAL,
20 POINTS FLOAT EXTERNAL,
```

To use this SQL, log into SQLPlus. Then copy the SQL from the CTL file (do not include the comment marks, but do include the semi-colon at the end). Then paste the string into SQLPlus.



```
C:\WINDOWS\system32\cmd.exe - sqlplus
C:\Documents and Settings\thaverkort>sqlplus
SQL*Plus: Release 11.1.0.6.0 - Production on Thu May 8 12:03:59 2008
Copyright (c) 1982, 2007, Oracle. All rights reserved.
Enter user-name: scott
Enter password:
Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
SQL> CREATE TABLE city_parks( NAME char(64), NAME_ALI char(64), PARTS float, POI
NTS float, ACRES float, PICTURES char(60));
Table created.
SQL>
```

The next step is to load the data from the CTL file into the newly created table. This is done through Oracle's SQLLoader interface. This image is one example:



```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\thaverkort>sqlldr userid=scott/tigers control=C:\FMEDA\
ta\Output\DemoOutput\city_parks.ctl
SQL*Loader: Release 11.1.0.6.0 - Production on Thu May 8 12:07:53 2008
Copyright (c) 1982, 2007, Oracle. All rights reserved.
Commit point reached logical record count 22
C:\Documents and Settings\thaverkort>
```

OS (GB) NTF Reader

This format is not supported by FME Base Edition.

Overview

The Ordnance Survey (Great Britain) National Transfer Format (OS(GB) NTF) Reader module provides FME with access to data in the United Kingdom (UK). This format is documented in the British Standard BS 7567.

More information on data products available in the OS(GB) NTF format, and their detailed product specifications, are found at this website:

<http://www.ordnancesurvey.co.uk/oswebsite/>

OS(GB) NTF Quick Facts

Format Type Identifier	NTF
Reader/Writer	Reader
Licensing Level	Professional
Dependencies	None
Dataset Type	Directory
Feature Type	Geometry type
Typical File Extensions	.ntf
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Never
Schema Required	Not applicable
Transaction Support	No
Geometry Type	ntf_type
Encoding Support	No

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	point	yes
circles	no	polygon	yes
circular arc	no	raster	no
donut polygon	yes	solid	no
elliptical arc	no	surface	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
ellipses	no		text	yes
line	yes		z values	no
none	yes			

Reader Overview

FME considers a single **.NTF** file, or a directory containing a set of **.NTF** files, to be a dataset. Groups of closely related records within the files are grouped together and transformed into FME features. In addition to geographic features such as points, lines, polygons, and text features, this reader also produces features representing the relationship between short feature code values and longer feature class names.

Reader Directives

The directives that are processed by the OS(GB) NTF reader are listed below. The suffixes shown are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the OS(GB) NTF reader is **NTF**.

DATASET

Required/Optional: *Required*

This is the name of a directory containing one or more NTF files. Each NTF file must have an extension of **.NTF**.

The example below identifies an NTF dataset:

```
NTF_DATASET D:\DATA\LANDFORM
```

Workbench Parameter: *Source OS (GB) NTF File(s)*

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available NTF files read. If no IDs are specified, then all available NTF files are read.

The syntax of the **IDs** keyword is:

```
<ReaderKeyword>_IDs <baseName1> \
    <baseName2> ... \
    <baseNameN>
```

The example below selects only the **SS68.NTF** file for input during a translation:

```
NTF_IDS SS68
```

FORCE_GENERIC

Required/Optional: *Optional*

The FME is programmed to recognize a set of standard Ordnance Survey NTF products and to provide a feature schema tailored to the data product, with field names appropriate to the data product. However, any NTF file not recognized as being a specifically targeted product profile, using the **DBNAME** field of the **DBHREC** record, is generically handled, at some cost to efficiency and tailoring of feature schema.

The **FORCE_GENERIC** keyword can be used to force recognized products to be treated using the generic rules, rather than the product-specific rules. This could be useful if new versions of standard products are inappropriately treated by product-specific rules or if you desire that all features from different products be returned using a common, or generic, schema.

NTF_FORCE_GENERIC ON

DEM_SAMPLE

Required/Optional: *Optional*

The FME can read Land-Form PANORAMA™ and Land-Form PROFILE™ raster Digital Terrain Model (DTM) products, translating each pixel into a point feature. This can produce a lot of point features—160000 to 250000 per file. Sometimes it is desirable to extract only a sub-sample of the features, while maintaining a regular grid pattern of points. The **DEM_SAMPLE** keyword is used to specify a decimation factor that is applied in the horizontal and vertical directions. A **DEM_SAMPLE** value of **3**, for instance, returns only one in three points in both horizontal and vertical directions from the reader, resulting in an 8/9^{ths} reduction in the total number of features.

NTF_DEM_SAMPLE 3

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the **mitab.dll** in the FME home directory to **mapinfo.dll**.

The syntax of the **MAPINFO_SEARCH_ENVELOPE** directive is:

MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The **COORDINATE_SYSTEM** directive, which specifies the coordinate system associated with the data to be read, must always be set if the **SEARCH_ENVELOPE_COORDINATE_SYSTEM** directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the **SEARCH_ENVELOPE_COORDINATE_SYSTEM** to the reader **COORDINATE_SYSTEM** prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the **SEARCH_ENVELOPE** directive.

Values

YES | NO (default)

Mapping File Syntax

<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Feature Representation

Features read from NTF consist of their geometry and a series of attribute values. Some features such as **FEATURE_CLASSES** have no geometry. The FME feature type is determined by the type of NTF record and the specific product from which the feature is extracted.

Product Schemas

The NTF reader considers a directory of NTF files to be a single dataset. All files in the directory are scanned on open to determine which NTF product they contain. For each particular product listed above, a set of layers are created. However, these layers may be extracted from several files of the same product.

The layers are based on a low-level feature type in the NTF file, and generally contain features of many different feature codes (**FEAT_CODE** attribute). Different features within a given layer may have a variety of attributes in the file, however, the schema is established based on the union of all possible attributes within features of a particular type.

If an NTF product is read that doesn't match one of the known schemas, it will go through a generic handler that has only layers of the **GENERIC_POINT** and **GENERIC_LINE** types. In such a case, the features will only have a **FEAT_CODE** attribute.

Product Types

The following is a list of product types and the FME feature types that they contain. Note that the feature type does not indicate which specific file the feature came from. For instance, if several Landline files are translated at once, features from all the files will be generically treated as **LANDLINE_POINT**, **LANDLINE_LINE**, or **LANDLINE_NAME**. When necessary, the source tile of a feature can be identified via the **TILE_REF** attribute attached to almost all NTF features.

NTF Product Type	Feature Type
Landline and Landline Plus	LANDLINE_POINT LANDLINE_LINE LANDLINE_NAME FEATURE_CLASSES
Panorama Contours	PANORAMA_POINT PANORAMA_CONTOUR FEATURE_CLASSES HEIGHT attribute holds elevation.
Strategi	STRATEGI_POINT STRATEGI_LINE STRATEGI_TEXT STRATEGI_NODE FEATURE_CLASSES
Meridian	MERIDIAN_POINT MERIDIAN_LINE MERIDIAN_TEXT MERIDIAN_NODE FEATURE_CLASSES
Boundary Line	BOUNDARYLINE_LINK BOUNDARYLINE_POLY BOUNDARYLINE_COLLECTIONS FEATURE_CLASSES The _POLY layer has links to links that allow true polygons to be formed, otherwise the _POLYs only have a seed point for geometry. The collections are collections of polygons, also without geometry as read. This is the only product from which polygons can be constructed.
Boundary Line 2000	BOUNDARYLINE_LINK BOUNDARYLINE_POLY BOUNDARYLINE_COLLECTIONS FEATURE_CLASSES The _POLY layer has links to links that

NTF Product Type	Feature Type
	<p>allow true polygons to be formed, otherwise the _POLYs only have a seed point for geometry.</p> <p>The collections are collections of polygons, also without geometry as read. This is the only product from which polygons can be constructed.</p>
BaseData.GB	BASEDATA_POINT BASEDATA_LINE BASEDATA_TEXT BASEDATA_NODE FEATURE_CLASSES
OSCAR Asset/Traffic	OSCAR_POINT OSCAR_LINE OSCAR_NODE FEATURE_CLASSES
OSCAR Network	OSCAR_NETWORK_POINT OSCAR_NETWORK_LINE OSCAR_NETWORK_NODE OSCAR_COMMENT FEATURE_CLASSES
OSCAR Route	OSCAR_ROUTE_POINT OSCAR_ROUTE_LINE OSCAR_ROUTE_NODE OSCAR_COMMENT FEATURE_CLASSES
Address Point	ADDRESS_POINT
Code Point	CODE_POINT
Code Point Plus	CODE_POINT_PLUS
Generic—only a subset of these appears in any given generic dataset	GENERIC_POINT GENERIC_LINE GENERIC_TEXT GENERIC_NAME GENERIC_NODE GENERIC_COLLECTION GENERIC_POLY

NTF Product Type	Feature Type
	FEATURE_CLASSES

Specific Feature Type Notes

The following list provides information specific to each of the feature types read by the NTF reader.

- ***_POINT**: Contains a point feature with a **POINT_ID** attribute containing the identifier (id) for the feature.
- ***_LINE**: Contains a line feature with a **LINE_ID** attribute containing the id for the feature.
- ***_CONTOUR**: Same as **_LINE**, but specific to contour products. Elevation is in the **HEIGHT** field and the id is in the **LINE_ID** field.
- ***_NAME**: Contains a textual feature with positioning, size, orientation, and font information. The feature id is in the **NAME_ID** field.
- ***_TEXT**: Similar to **_NAME** features, but the id is in the **TEXT_ID** field.
- ***_NODE**: A point feature with a list of **_LINE** feature ids starting or ending at the node in the **GEOM_ID_OF_LINK** list field. The **DIR** field indicates the direction of each line. This could potentially be used for routing, but is generally ignored.
- ***_COMMENT**: Contains indication of a feature (**RECORD_ID**) and type (**RECORD_TYPE**) that have been updated in this product release, as well as an indication of the change (**CHANGE_TYPE**).
- ***_POLY**: Contains a polygon feature. Note that the polygon geometry for these features is generated by the processing pipeline, and that the "uncooked" features have no geometry or, in some cases, just an inside point.
- **BOUNDARYLINE_COLLECTIONS**: An aggregate of polygons representing an administrative region.
- **GENERIC_COLLECTIONS**: The feature contains references to other features and some attributes of the grouping. Due to the nature of these collections, it isn't possible to generically aggregate them.
- **FEATURE_CLASSES**: These features relate a feature code string (**FEAT_CODE**) such as 4001 with a feature class description of **string**. The standard processing pipeline automatically uses these features to add an **FC_NAME** attribute to all features with the long description corresponding to their feature code.
- **DTM_***: Raster DTM pixels are translated into point features. The FME feature type is established by appending the tile name, normally part of the file name, to **DTM_***. That is, unlike all other NTF features, the feature type of raster DTM points is based on the file name. The point elevation is in the **HEIGHT** attribute.

Special Attributes

Several feature types use special attributes to hold source data information. The table below lists the special attribute names used and provides a description of their contents.

Special Attribute Name	Description
FEAT_CODE	This general feature code integer can be used to look up a name in the FEATURE_CLASSES layer or table.

Special Attribute Name	Description
TEXT_ID POINT_ID LINE_ID NAME_ID COLL_ID POLY_ID GEOM_ID	This is the unique identifier for a feature of the appropriate type.
TILE_REF	All layers except FEATURE_CLASSES contain a TILE_REF attribute that indicates from which tile, or file, the features came. Generally speaking, the id numbers are only unique within the tile, so the TILE_REF can be used to restrict id links within features from the same file.
FONT TEXT_HT DIG_POSTN ORIENT	This provides the detailed information on the font, text height, digitizing position, and orientation of text, or name, objects. Review the Ordnance Survey (OS) product manuals to understand the units and the meaning of these codes.
GEOM_ID_OF_POINT	For _NODE features, this defines the POINT_ID of the point layer object to which this node corresponds. Generally speaking, the nodes don't carry a geometry of their own. The node must be related to a point to establish its position.
GEOM_ID_OF_LINK	This is a _list_ of _LINK or _LINE features to end or start at a node. Nodes and this field are generally only of value when establishing connectivity of line features for network analysis. Note that this should be related to the target features GEOM_ID , not its LINE_ID . On the BOUNDARYLINE_POLY layer, this attribute contains the GEOM_IDs of the lines that form the edge of the polygon.
POLY_ID	This is a list of POLY_IDs from the BOUNDARYLINE_POLY layer associated with a given collection in the BOUNDARYLINE_COLLECTIONS layer.

Points

ntf_type: ntf_point

Features with **ntf_point** as their **ntf_type** contain a two-dimensional (2D) point. There are no other attributes specific to this feature type.

ntf_type: ntf_point3d

Features with **ntf_point3d** as their **ntf_type** contain a three-dimensional (3D) point. There are no other attributes specific to this feature type.

Nodes

ntf_type: ntf_node

Features with **ntf_node** as their **ntf_type** contain a 2D point, which is a node in the dataset. There are no other attributes specific to this feature type.

Lines

ntf_type: ntf_line

Features with **ntf_line** as their **ntf_type** contain a 2D line and there are no other attributes specific to this feature type.

ntf_type: ntf_line3d

Features with **ntf_line3d** as their **ntf_type** contain a 3D line. There are no other attributes specific to this feature type.

Polygons

ntf_type: ntf_polygon

Features with **ntf_polygon** as their **ntf_type** contain a 2D closed polygon. There are no other attributes specific to this feature type.

Collections

ntf_type: ntf_collection

Features with **ntf_collection** as their **ntf_type** contain information pertaining to a collection of features.

Annotations

ntf_type: ntf_text

Features with **ntf_text** as their **ntf_type** contain a 2D insert point for an annotation feature. Such features have these attributes:

Attribute Name	Contents
TEXT	The annotation string to appear at the insert point
ORIENT	The orientation of the text, measured in degrees counterclockwise from horizontal
TEXT_HT_GROUND	The text height, measured in ground units

PenMetrics GRD Reader/Writer

The PenMetrics GRD Reader/Writer provides the Feature Manipulation Engine (FME) with access to PenMetrics Graphical Drawing (GRD) 32-bit format files. GRD files consist of drawing settings and configuration, as well as a series of vectors, or graphic elements, organized into layers. The FME provides broad support for GRD vector types and options. In addition, when GRD data is output, header information may be copied from a supplied template or prototype file.

Overview

The GRD 32-bit file format is the native file format for PenMetrics vector drawings used with applications such as FieldNotes.

GRD files may contain both two-dimensional (2D) and three-dimensional (3D) features. GRD files store feature geometries as well as user-defined attributes. There are 11 kinds of features as follows: points, lines, polylines (including polygons), rectangles, circles, ellipses, arcs, inserts, text, ink, and multi-segmented polylines.

The FME looks for an extension of `.grd` for the input GRD files, but accepts any GRD file as input regardless of the file name or extension. The FME considers a GRD dataset to be a single PenMetrics GRD file. GRD files are binary files that consist of a combination of fixed and variable-length binary records.

The information held within the GRD file itself is contained in five separate sections:

- Header
- Linetypes
- Layers
- Blocks
- Layer Vectors

The organization of GRD files closely parallels that of AutoCAD files.

GRD Entity Types and Descriptions

The GRD reader and writer use symbolic names for different entity types stored within a data file. This simplifies feature type specification. The following table gives a brief description of each of the different GRD entity types currently supported by the reader and writer. The entities are described in detail in subsequent sections.

FME <code>grd_entity</code>	Description
<code>grd_line</code>	Linear features stored within the GRD file as a line or unclosed polyline.
<code>grd_point</code>	Point features.
<code>grd_ellipse</code>	Features with an elliptical representation.
<code>grd_circle</code>	Features with an circular representation.
<code>grd_polygon</code>	Features whose geometry is represented by a closed polyline.
<code>grd_arc</code>	Features whose geometry represents a portion of a circular arc.
<code>grd_rectangle</code>	Features with a closed rectangular geometry. The edges are vertical and horizontal only — no rotation.

FME grd_entity	Description
grd_text	Text features.
grd_ink	Ink features that store images in binary form in a text attribute.
grd_mspline	A group of associated lines are stored as a multi-segmented polyline feature.
grd_insert	Point features that carry insert entity, or block, data.

Layers

GRD files use a layers concept to organize all features they contain. Every feature lies on one of the many layers that may be defined. Each layer has a unique name and defines colors, line styles, font styles, etc. for use with all features on that layer that do not have overriding settings. Layers may be either Drawing or Markup types. Every GRD file has the following two mandatory layers:

- Layer "0" — the drawing layer
- Layer "Markup 0" — the markup, or inking, layer

Any number of other layers may be defined by the user.

Linetypes

Linetypes are used to define the way line work is meant to be displayed. When reading from GRD files, several linetype specific attributes are given to each feature. Together they define the linetype that it is meant to be displayed with. When writing GRD files, these linetype specific attributes, if present, are stored as the linetype to use. By default, a linetype of **CONTINUOUS** is used when writing GRD files.

A linetype definition has a name and a description that appears to the user. Usually the description shows what the linetype looks like by using underscores ('_') and spaces (' '), for example: "_____. ____". The actual linetype definition consists of a series of dashes. There are a maximum of 12 dashes that can make up a linetype. Here is how the linetypes are represented:

- **Dash:** Positive dash length where the length is the length of the dash.
- **Space:** Negative space length where the length is the absolute value of the space.
- **Dot:** Dash length of zero.

To represent this linetype "_____. ____", the following dashes would be necessary: 2, -2, 2, -2, 4, -2, 0, -2, 2, -2, 2, -2

The details of how this information is represented in FME attributes is found in this section under the heading Linetypes Representation.

GRD Numeric Color Associations

The numeric color values referred to in GRD files have the following associations:

GRD Numeric Color Value	Description
0	black
1	blue
2	green
3	cyan

GRD Numeric Color Value	Description
4	red
5	magenta
6	brown
7	light gray
8	gray
9	light blue
10	light green
11	light cyan
12	light red
13	light magenta
14	yellow
15	white
16	visible (pen) or transparent (brush)
17	use layer's color
18	use block's color (valid only for block entities)

GRD Quick Facts

Format Type Identifier	GRD
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	File
Feature Type	Layer name
Typical File Extensions	.grd
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	Yes
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	grd_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	yes		polygon	yes
circular arc	yes		raster	no
donut polygon	yes		solid	no
elliptical arc	yes		surface	no
ellipses	yes		text	yes
line	yes		z values	yes
none	no			

Reader Overview

The GRD reader opens the input file, immediately starts reading features, and returns them to the rest of the FME for processing. The reader has no requirement for definition statements as the user-defined attributes are defined completely within the GRD file itself.

Each returned feature has its feature type set to either the layer name or the geometric type of the feature, as follows: **point**, **line**, **polygon**, **rectangle**, **circle**, **ellipse**, **arc**, **insert**, **text**, **ink**, or **mspline**.

Reader Directives

The directives processed by the GRD reader are listed below. The suffixes shown are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the GRD reader is **GRD**.

DATASET

The value for this keyword is the file containing the GRD dataset to be read.

Required/Optional

Required

Mapping File Syntax

```
GRD_DATASET /usr/data/PenMetrics/test.grd
```

* Workbench Parameter

Source PenMetrics GRD File(s)

EXPAND_INSERTS

This setting determines whether insert features are output as one or several separate features. With YES, each element of the insert blocks are output as separate features. With NO, each insert block is output as a single point feature.

When the reader expands inserts — also referred to as resolving blocks — it outputs one feature for each of the GRD vector entities that are part of the block definition. The original insert is not output. This results in the full graphical representation of the insert transferred through the FME, but the exact insertion point of the insert is lost.

Each insert member feature is given the attribute *grd_insert_number* set to the same value for each block so the features that comprise each insert may be combined in subsequent processing.

If the exact insertion point of the insert is desired, then insert expansion should be turned off. This results in each insert block being translated into a point feature in the output system.

Required/Optional

Optional

Mapping File Syntax

A typical mapping file fragment specifying that linked features should not be broken looks like:

```
GRD_EXPAND_INSERTS NO
```

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The GRD writer creates and writes feature data to the GRD file specified by the **DATASET** keyword. Any GRD file with the same name is overwritten with the new feature data.

The GRD writer provides the following capabilities when writing GRD files.

- **User-defined Linetypes:** New linetypes can be defined on attributes attached to features being written to the GRD file.
- **User-defined Layers:** Users must define the layers into which features are stored. The layers can also define the attributes to be stored within the feature.
- **Copy Block Definitions:** Often users have existing GRD data files that contain block definitions they want the translated data to carry. Specifying the **TEMPLATE_FILE** keyword in the mapping file results in block definitions being copied from the existing file to the output GRD file. These blocks can then be referred to by insert entities.
- **Copy Linetypes:** Predefined linetypes within existing GRD files are copied making them available for use by features being written to the destination file. Specifying the **TEMPLATE_FILE** keyword in the mapping file results in the predefined linetypes being copied from the template file to the output drawing file. Feature entities can then refer to these linetype definitions.
- **Copy Layer Definitions:** Layer definitions within an existing GRD file identified by **TEMPLATE_FILE** enables layer definitions to be copied to the destination dataset and then referenced.
- **Automatic Block Creation:** When a feature that cannot be written as a single GRD vector entity is passed to the writer—such as a donut polygon—the writer automatically defines a GRD block and inserts entities necessary to represent the feature.

Writer Directives

The directives that are processed by the SDF3 writer are listed below. The suffixes shown are prefixed by the current **<WriterKeyword>_** in a mapping file. By default, the **<WriterKeyword>** for the SDF3 writer is **SDF3**.

DATASET

The value for this keyword is the file containing the GRD dataset to write.

Required/Optional

Required

Mapping File Syntax

```
GRD_DATASET /usr/data/PenMetrics/output.grd
```

* Workbench Parameter

Destination PenMetrics GRD File(s)

TEMPLATE_FILE

This setting gives the name of the file or files used as templates.

All layer styles, line styles, and block definitions are taken directly from the template files and used in the output GRD file that the FME produces. Multiple template file names may be listed after the keyword on a single line, or multiple template file names may be listed on separate lines, each beginning with the **TEMPLATE_FILE** directive. New layers can be defined during the translation using the **DEF** lines and added to those brought in from the template files. If there are duplicate definitions for the same layer, the **DEF** line definitions prevail.

Tip: LINETYPE definitions found in the mapping file override any linetype definitions found in the template file.

Required/Optional

Optional

Mapping File Syntax

```
GRD_TEMPLATE_FILE /usr/data/penmetrics/map.grd
```

AUTO_CREATE_LAYERS

Required/Optional: *Optional*

This statement tells the writer to create layers as needed. Normally, all layers must either be defined by **_DEF** lines or the template file before they can be used. If **AUTO_CREATE_LAYERS** is specified as **yes**, and a feature with a feature type not previously defined as a layer is sent to the writer, then a new layer will be created. This layer is created with the properties of the last **_DEF** line found in the mapping file, if any, or it uses other defaults.

This example sets the writer into a mode where it creates layers as needed. Each created layer has a color of **4** (red) and a linetype of **CONTINUOUS**.

```
GRD_AUTO_CREATE_LAYERS yes
GRD_DEF_DEFAULT \
  grd_color      10 \
  grd_linetype   CONTINUOUS
```

LINETYPE

Required/Optional: *Optional*

The GRD writer enables linetypes to be defined within the FME mapping file. This lets the user control how output lines look in the destination dataset. The linetype definition takes the following form:

```
<writerKeyword>_LINETYPE <linetype name> \
  [grd_linetype_description <picture>] \
  [grd_dash_type <dash type>] \
  [<segment values>+]
```

where:

- <linetype name> is the name used throughout the mapping file to refer to the linetype being defined by this statement. If this is not set, "" (By Layer) is used.
- <picture> is the text or name displayed in FieldNotes when linetypes are displayed. If this is not set, "" is used.
- <dash type> can have the value **0** or **1**, where **0=PIXEL** and **1=VIRTUAL**. This indicates whether the dash lengths are in real world coordinates or if they represent a length in screen pixels. Virtual linetypes spread out when you zoom in on them. Pixel linetypes are the same no matter what scale you are at. If this is not set, **0 (PIXEL)** is used.
- <segment values> are the length of each of the segments within the linetype segment. There is a maximum of 12 segments to each linetype. If no segment values are set then by default none are used. The segment values obey the following rules:
 - negative value — pen up length, used to create spaces of varying lengths
 - positive value — pen down length, used to make dashes of varying lengths
 - zero — used to create a dot

The following example creates a linetype called *dash-dot* which appears as " _ . _ . _ . " and so on when displayed on the screen.

```
GRD_LINETYPE dash-dot \
  grd_linetype_description " _ . _ . _ . " \
```

```
grd_dash_type 0 \  
2.5 -2.25 0 -2.25
```

DEF

Required/Optional: *Optional if AUTO_CREATE_LAYERS is used*

The GRD writer requires that every feature written to the GRD file is stored within a predefined GRD layer. In GRD, the layers are used to store collections of logically related attributes. Within the FME, the GRD layer and the type of feature are treated synonymously as there is a one-to-one correspondence between FME feature type and GRD layer.¹ The layer statement has the following form:

```
<writerKeyword>_DEF <layer name> \  
[grd_layer_type <layer type>] \  
[grd_is_fixed <boolean>] \  
[grd_is_visible <boolean>] \  
[grd_pen_color <default color>] \  
[grd_brush_color <default color>] \  
[grd_pen_width <default width>] \  
[grd_pen_width_type <default width type>] \  
[grd_brush_type <default brush type>] \  
[grd_linetype <default linetype>] \  
[<attribute name> <attribute type>]
```

where:

- <layer name> is the name of the layer being defined and is used throughout the remainder of the FME mapping files.
- <layer type> is the type of layer being defined. The default is **DRAWING**. The values are associated with the following types:
 - 0 = MARKUP
 - 1 = DRAWING
- <boolean> is either **True** or **False**. By default, layers are visible and not fixed.
- <default color> is the color number used for all features stored within the layer unless explicitly overridden on the correlation lines below. Valid values are between 0 and 18. Refer to the discussion under the heading *Colors* for the color description. By default, **color = 0** (black).
- <default pen width> is the actual width of the pen. This expects a numeric value and is interpreted differently depending on the pen width type. The default is **1**.
- <default pen width type> is the pen width type used for all features stored within the layer unless explicitly overridden on the correlation lines below. The default value is **1** (Hairline). The values are associated with the following types:
 - 1 = HAIRLINE
 - 2 = VIRTUAL
 - 3 = PIXEL
- <default brush type>: is the numeric brush type used for all features stored within the layer unless explicitly overridden on the correlation lines below. The default is **0** (Solid). The values are associated with the following types:
 - 0 = SOLID
 - 1 = DIAGONAL #1
 - 2 = CROSS
 - 3 = DIAGONAL COORD

¹Layers can also be defined through the use of a **TEMPLATE_FILE**.

- 4 = DIAGONAL #2
- 5 = HORIZONTAL
- 6 = VERTICAL
- **<default linetype>** is the name of the linetype used for the layer if no linetype is specified on the correlation line. The default value is **CONTINUOUS**. The linetype specified must be:
 - defined in the mapping file,
 - copied from a specified template file, or
 - be the predefined linetype named **CONTINUOUS**
- **<attribute name> <attribute type>** is the definition of an attribute to be stored within the extra entity data of features for the layer. If no attributes are defined, then all feature attributes except those that start with **grd_** are stored. The storing of attributes can be turned off by specifying an attribute type of **SKIP**.

The example below defines a layer called **boundary** in which entities are drawn using pen color 13 unless otherwise specified, and a linetype called dash-dot unless otherwise specified, etc. The feature also has several specified attributes that are written to the extra entity data of each feature within the layer.

```
GRD_DEF boundary \
  grd_pen_color 13 \
  grd_brush_color 4 \
  grd_brush_type 2 \
  grd_pen_width_type 3 \
  grd_linetype dash-dot \
      FEATCODE      char(12)
      PPID          char(10) \
      DATECHNG     date \
      SURVEYDIST   number(8,2)
```

Feature Representation

GRD features consist of geometry and attribute information. All GRD FME features contain the **grd_type** attribute that identifies the geometric type, as well as many common attributes.

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), and depending on the geometric type, the feature may contain additional attributes specific to the geometric type. These are described in subsequent sections.

Attribute Name	Contents
grd_type	<p>The GRD geometric type of this entity.</p> <p>Range:</p> <p>grd_point grd_line grd_polygon grd_rectangle grd_ellipse grd_circle grd_arc grd_insert grd_text grd_ink grd_mspline</p>

Attribute Name	Contents
	Default: No default

GRD Attributes

The following table lists all common GRD attributes returned on all features produced by the FME reader. If these attributes are present on any feature written out to GRD, they will be used to set the appropriate values in the output file.

Field Name	Description
grd_vector_type	This is the type of vector. Maximum size is 10 characters. When reading GRD files, the value is one of the following: POINT, LINE, PLINE, RECT, CIRCLE, ELLIPSE, ARC, INSERT, TEXT, INK, or MS-PLINE
grd_pen_width_type	This is the type of pen width used to plot the vector. Maximum size is 10 characters. The default is 0. The values are associated with the following types: 0 = USE LAYER 1 = HAIRLINE 2 = VIRTUAL 3 = PIXEL
grd_pen_width	This is the width of the pen used to plot the vector. This value is interpreted differently depending on the pen width type. The default is 1 . Range: Any real number.
grd_pen_color	This is the color of the pen used to plot the vector. Refer to the information under the heading <i>Colors</i> for a description of each color. The default is 0 (black). Range: 0-18
grd_brush_color	This is the color of the brush used to plot the vector. Refer to the information under the heading <i>Colors</i> for a description of each color. Note that color 16 is TRANSPARENT. The default is 0 (black). Range: 0-18
grd_brush_type	This is the type of brush used for this vector. The default is 0 (solid). The values are associated with the following types: 0 = SOLID 1 = DIAGONAL #1 2 = CROSS 3 = DIAGONAL COORD 4 = DIAGONAL #2 5 = HORIZONTAL

Field Name	Description
	6 = VERTICAL 7 = USE LAYER'S BRUSH TYPE 8 = USE BLOCK'S BRUSH TYPE Range: 0-8
grd_reserved	Boolean flag indicating whether the Reserved bit on this vector is set. The default is False. Range: True False
grd_added	Boolean flag indicating whether the IsAdded bit on this vector is set. The default is False. Range: True False
grd_has_extra_data	Boolean flag indicating whether the HasExtraData bit on this vector is set. The default is False. Range: True False
grd_visible	Boolean flag indicating if the IsVisible bit on this vector is set. The default is True. Range: True False
grd_selected	Boolean flag indicating whether the IsSelected bit on this vector is set. Note that this setting is unreliable within the FieldNotes application. The default is False. Range: True False
grd_changed	Boolean flag indicating if the IsChanged bit on this vector is set. The default is False. Range: True False
grd_has_annotation	Boolean flag indicating if the HasAnnotation bit on this vector is set. The default is False. Range: True False
grd_vector_handle	This integer is the vector handle. The default is 0. Range: 0 - (2 ³³ -1)
grd_annotation_string	This string holds the annotation for the vector. Maximum size is 254 characters. There is no annotation by default.
grd_annotation_x grd_annotation_y grd_annotation_z	This is the location of the annotation for the vector. The defaults are all zero. Range: Any real number.

Field Name	Description
grd_linetype_name	This is the name of the linetype used to display the vector. The linetype must be defined in the mapping file or the template file being used. The default linetype is to use the layer's linetype. Maximum size is 50 characters.
grd_database_key	This is a key string that links the vector to a database. The format of, and relationship between, the database and the feature is beyond the scope of the GRD reader or writer. Maximum size is 254 characters.
grd_layers_pen_width_type	When reading GRD files, this boolean flag indicates whether the pen width was taken from the layer or if it was specifically set on the feature. When writing GRD files, this boolean flag indicates if the feature should use the layer's pen width or specifically set its own. The default is False. Range: True False
grd_layers_pen_color	When reading GRD files, this boolean flag indicates whether the pen color was taken from the layer or if it was specifically set on the feature. When writing GRD files, this boolean flag indicates if the feature should use the layer's pen color or specifically set its own. The default is False. Range: True False
grd_blocks_pen_color	When reading GRD files, this boolean flag indicates if the pen color was taken from the block or if it was specifically set on the feature. When writing GRD files, this boolean flag indicates whether the feature should use the block's pen color or specifically set its own. The default is False. Range: True False
grd_layers_brush_color	When reading GRD files, this boolean flag indicates if the brush color was taken from the layer or if it was specifically set on the feature. When writing GRD files, this boolean flag indicates if the feature should use the layer's brush color or specifically set its own. The default is False. Range: True False
grd_blocks_brush_color	When reading GRD files, this boolean flag indicates if the brush color was taken from the block or if it was specifically set on the feature. When writing GRD files, this boolean flag indicates if the feature should use the block's brush color or specifically set its own. The

Field Name	Description
	default is False. Range: True False
grd_layers_brush_type	When reading GRD files, this boolean flag indicates if the brush type was taken from the layer or if it was specifically set on the feature. When writing GRD files, this boolean flag indicates if the feature should use the layer's brush type or specifically set its own. The default is False. Range: True False
grd_blocks_brush_type	When reading GRD files, this boolean flag indicates if the brush type was taken from the block or if it was specifically set on the feature. When writing GRD files, this boolean flag indicates if the feature should use the block's brush type or specifically set its own. The default is False. Range: True False
grd_layers_linetype	When reading GRD files, this boolean flag indicates if the linetype was taken from the layer or if it was specifically set on the feature. When writing GRD files, this boolean flag indicates if the feature should use the layer's linetype or specifically set its own. The default is False. Range: True False
grd_linetype_description	This is a text description of the linetype used for this vector. Refer to the discussion under the heading <i>Linetypes</i> for detailed descriptions. The linetype must be defined in either the mapping file or the template file being used. By default, the layer's linetype is used. Maximum size is 80 characters, with a maximum of 12 entries. Example: " _ _ _ _ _ . _ _ _ _ _ "

Points

grd_type: grd_point

GRD point features represent single point features and may be either 2D or 3D. These features have the following special attributes associated with them.

Attribute Name	Contents
grd_virtual_size	This is the size the point should appear when plotted. Range: Any real number
grd_point_type	This is the symbology of the point when plotted.

Attribute Name	Contents
	<p>The value here may be the sum of any one of the internal types and any one of the outline types.</p> <p>Internal Types:</p> <p>0 = dot 1 = none 2 = plus 3 = cross 4 = vertical line</p> <p>Outline Types:</p> <p>0 = none 32 = circle 64 = square 96 = cirsquare</p> <p>Range: 0 - 100</p>

grd_type: grd_ink

GRD ink features represent a raster picture or image. Their geometry is a point that indicates the lower-left point where the ink should be located. Ink features may have the following special attributes associated with them.

Attribute Name	Contents
grd_ink_data	This attribute contains the ink data. This is usually binary data stored in a string attribute, so typical string manipulation with this attribute may not be appropriate.
grd_width	This is the width of the ink data. Range: Any real number
grd_height	This is the height of the ink data. Range: Any real number

grd_type: grd_insert

GRD insert features represent a block of features linked together. The geometry of this point consists of a point that indicates the location of the insert block. Inserts may also have any number of attributes associated with them (as defined in the block definition of the file). These attributes will appear on the feature with the attribute names as defined. When writing insert attributes to GRD files, attributes with the prefix `grd_insert_attribute{N}` are first searched for necessary information. If these attributes are not found, then attributes with the prefix `fme_attr_info{N}` are sought. Insert features may also have the following special attributes associated with them.

Attribute Name	Contents
grd_block_name	This attribute contains the name of the insert that holds all associated vectors together. If used with a <code><reader keyword>_TEMPLATE_FILE</code> setting when writing GRD files, inserts may use blocks that have already been defined elsewhere. Max-

Attribute Name	Contents
	imum size is 254 characters.
grd_rotation	<p>This is the rotation of the insert block, in degrees counterclockwise.</p> <p>Note: Ellipses must have a rotation that is a multiple of 90 degrees. Therefore, when writing out to GRD, if an insert refers to a block that has an ellipse in its definition, the rotation of the insert is rounded to the nearest multiple of 90 degrees when it is output.</p> <p>Range: 0 .. 360.0</p>
grd_scale_x grd_scale_y grd_scale_z	<p>This is the scale of the insert block—a scaling factor applied to the ground units that block is defined in. Either the scale or the size of inserts must be specified.</p> <p>Range: Any real number</p>
grd_size_x grd_size_y grd_size_z	<p>This is the desired size of the insert block’s bounding box, in ground units. Either the scale or the size of inserts must be specified. If both are specified, the size settings are ignored.</p> <p>Range: Any real number</p>
grd_insert_attribute{N}.field_name fme_attrib_info{N}.field_name	<p>These list attributes hold the name of the Nth attribute.</p> <p>Range: text string</p>
grd_insert_attribute{N}.field_type fme_attrib_info{N}.field_type	<p>These list attributes hold the type of the Nth attribute.</p> <p>1=DOUBLE 2=LONG 3=STRING 4=LOGICAL 5=DATE 6=TIME (unsupported) 7=MONEY (unsupported) 8=MEMO 9=BLOB 10=DATETIME (unsupported)</p> <p>Range: integer Default: 3</p>

Attribute Name	Contents
grd_insert_attribute{N}.field_size fme_attrib_info{N}.field_size	These list attributes hold the size of the N th attribute. Range: integer
grd_insert_attribute{N}.num_decimal_places fme_attrib_info{N}.num_decimal_places	These list attributes hold the number of decimal places in the value of the N th attribute (if appropriate). Range: text string
grd_insert_attribute{N}.default_value fme_attrib_info{N}.default_value	These list attributes hold the default value of the N th attribute. Range: text string
grd_insert_attribute{N}.isVisible fme_attrib_info{N}.isVisible	These list attributes indicates whether the N th attribute should be displayed or not. Range: TRUE FALSE
grd_insert_attribute{N}.color fme_attrib_info{N}.color.red fme_attrib_info{N}.color.green fme_attrib_info{N}.color.blue	These list attributes hold the color of the N th attribute. The GRD attribute holds the color based on GRD color codes. The FME attributes hold the color in RGB values, ranged between 0.0 and 1.0. Range: integer (GRD color); real number 0.0-1.0 (FME color)
grd_insert_attribute{N}.locationX grd_insert_attribute{N}.locationY grd_insert_attribute{N}.locationZ fme_attrib_info{N}.location_x fme_attrib_info{N}.location_y fme_attrib_info{N}.location_z	These list attributes hold the plotting location of the N th attribute. Range: real number
grd_insert_attribute{N}.height fme_attrib_info{N}.height	These list attributes hold the display height of the N th attribute. Range: real number
grd_insert_attribute{N}.rotation fme_attrib_info{N}.rotation	These list attributes hold the display rotation of the N th attribute. Range: real number
grd_insert_attribute{N}.horizontal_align	This list attribute holds the display horizontal alignment of the N th attribute. 0=CENTER 1=LEFT 2=RIGHT Range: integer

Attribute Name	Contents
	Default: 0
grd_insert_attribute{N}.vertical_align	This list attribute holds the display vertical alignment of the N th attribute. 0=CENTER 1=TOP 2=BOTTOM 3=BASELINE Range: integer Default: 3
fme_attrib_info{N}.justification	This list attribute holds the display justification of the N th attribute. The value it can have is one of the following: baseline_middle baseline_right bottom_right middle_right top_right baseline_left bottom_left middle_left top_left Range: text string Default: baseline_middle

Lines

grd_type: grd_line

GRD line features represent linear features and may be either 2D or 3D. These features do not have any special attributes associated with them.

When writing lines out to GRD, all lines are written as PLINES. The only exception is if any line to be written out has exactly two points and also has an attribute `grd_vector_type` with a value of LINE; in this case, a LINE is written out to GRD.

grd_type: grd_mspline

GRD multi-segmented polyline features represent an aggregate consisting of linear features, all of which are the same — either 2D or 3D. **M spline** features may have the following special attribute associated with them.

Attribute Name	Contents
grd_ordered	This indicates whether or not multiple lines are ordered. Range: 0 1

Arcs (circles)

grd_type: grd_arc

GRD arc features represent circular arc features and are either 2D or 3D. The coordinate on the feature contains the location of the centre. Arc features may have the following special attributes associated with them.

Attribute Name	Contents
grd_rotation	The rotation of the arc, in degrees counter-clockwise. Range: 0 .. 360.0
grd_primary_axis	The radius of the arc. Range: Any real number
grd_secondary_axis	The radius of the arc. Range: Any real number
grd_start_angle	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of start_angle. Range: 0 .. 360.0
grd_sweep_angle	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of sweep_angle. Range: Any real number

grd_type: [grd_circle](#)

GRD circle features represent closed circular arc features and may be either 2D or 3D. The coordinate on the feature contains the location of the centre. Circle features may have the following special attribute associated with them.

Attribute Name	Contents
grd_primary_axis	The radius of the circle. Range: Any real number

grd_type: [grd_ellipse](#)

GRD ellipse features represent closed elliptical arc features and may be either 2D or 3D. The coordinate on the feature contains the location of the centre. Ellipse features may have the following special attributes associated with them.

Attribute Name	Contents
grd_rotation	The rotation of the ellipse, in degrees counterclockwise. Note: All ellipses must have a rotation that is a multiple of 90 degrees. If an invalid rotation is passed in to the GRD writer, the ellipse is written out as a polygon instead. Range: 0 .. 360.0
grd_primary_axis	The radius of the longest axis of the ellipse. Range: Any real number
grd_secondary_axis	The radius of the shortest axis of the ellipse. Range: Any real number

Polygons

grd_type: [grd_polygon](#)

GRD polygon features represent closed polygonal features and may be either 2D or 3D. These features do not have any special attributes associated with them.

grd_type: grd_rectangle

GRD rectangle features represent closed polygonal features, containing only four points, that make up a polygon. These rectangles cannot have any rotation—their edges must be horizontal or vertical. These features may be either 2D or 3D, and do not have any special attributes associated with them.

Text

grd_type: grd_text

GRD text features hold text information. A single 2D or 3D position is associated with the text block. Text features may have the following special attributes associated with them.

Attribute Name	Contents
grd_width	The width of the text string. Range: Any real number
grd_height	The height of the text string. Range: Any real number
grd_rotation	The angle of the string, in degrees counter-clockwise from horizontal. Range: 0 .. 360.0
grd_italic	Boolean flag indicating if the text is to be displayed in italic type. Range: True False
grd_bold	Boolean flag indicating if the text is to be displayed in bold type. Range: True False
grd_font_name	The name of the font used to display the text. Maximum size is 254 characters.
grd_text_font_type	The specific kind of type the text is to be displayed in. The default is 0 (TrueType Font). The values are associated with the following types: 0 = TrueType Font 1 = NonTrueType Font 3 = Specific Font
grd_text_string	The text string of the vector. Maximum size is 254 characters.

PHOCUS PHODAT Reader/Writer

The PHOCUS PHODAT (PHOCUS) Reader module provides the Feature Manipulation Engine (FME) with the ability to translate PHOCUS data in and out of any FME format. PHOCUS PHODAT is a published ASCII format, output by the Carl-Zeiss PHOCUS system.

Overview

The PHOCUS files may contain both two-dimensional (2D) and three-dimensional (3D) features. The PHOCUS files do not explicitly store attribute values but rather use a feature coding approach whereby unique feature codes are assigned to different types of features stored within the dataset. The FME looks for an extension of **.pdt** for the input PHOCUS files, but will accept any PHOCUS file as input regardless of the file name or extension.

The PHOCUS reader module supports the reading of point, line, area, and text geometric data in PHOCUS files.

Each geometric entity present in a PHOCUS file is assigned an object code and an item code. Together these codes define the type of feature being read.

The FME considers a PHOCUS dataset to be a single **PHOCUS PHODAT** file.

PHOCUS PHODAT Quick Facts

Format Type Identifier	PHOCUS
Reader/Writer	Reader
Licensing Level	Base
Dependencies	None
Dataset Type	File
Feature Type	Object code
Typical File Extensions	.pdt
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	phocus_type
Encoding Support	No

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	point	yes
circles	yes	polygon	yes
circular arc	no	raster	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	yes		text	yes
line	yes		z values	yes
none	no			

Reader Overview

The PHOCUS reader opens the input file and immediately starts reading features, returning them to the rest of the FME for processing. The reader has no requirement for definition statements as there are no user-defined attributes.

Each feature returned has its feature type set to the value of the features object code as defined by PHOCUS.

Reader Directives

The directive processed by the PHOCUS reader are listed below. The suffix shown is prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the PHOCUS reader is **PHOCUS**.

DATASET

Required/Optional: *Required*

The value for this keyword is the file containing the **PHOCUS** dataset to be read. A typical mapping file fragment specifying an input **PHOCUS** dataset looks like:

```
PHOCUS_DATASET /usr/data/phocus/db84.pdt
```

Workbench Parameter: *Source PHOCUS PHODAT File(s)*

SPLIT_INVISIBLE_LINES

Required/Optional: *Optional*

This directive indicates whether lines that have visible and invisible components are to be split apart and returned as visible and invisible lines. If not specified, or if the value is NO, then lines with visible and invisible components are not split up.

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the **mitab.dll** in the FME home directory to **mapinfo.dll**.

The syntax of the **MAPINFO_SEARCH_ENVELOPE** directive is:

```
MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

```
MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

DATASET

Required/Optional: *Required*

The value for this keyword is the file containing the PHOCUS dataset to be read. A typical mapping file fragment specifying an input PHOCUS dataset looks like:

```
PHOCUS_DATASET /usr/data/phocus/db84.pdt
```

Workbench Parameter: *Destination PHOCUS PHODAT File(s)*

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Attribute Name	Contents
phocus_type	<p>The PHOCUS geometric type of this entity.</p> <p>Range:</p> <p>phocus_point phocus_line phocus_area phocus_text</p> <p>Default: No default</p>
phocus_visibility	<p>This attribute is specified when the PHOCUS reader is run with the directive <code>SPLIT_INVISIBLE_LINES</code> set to <code>yes</code>. The attribute indicates if the line is either <code>visible</code> or <code>invisible</code>.</p> <p>If this attribute is specified when features are written and the value is <code>invisible</code>, then the feature is written to the file such that it is not displayed.</p>

Point Cloud XYZ (POINTCLOUDXYZ) Reader/Writer

The Point Cloud XYZ (POINTCLOUDXYZ) Reader/Writer allows FME to read and write point clouds from features into files in the xyz format.

Overview

XYZ files are ASCII database files, where each column in a row is delimited by some separator character. Each row represents a point within a point cloud. Each column represents a point component for the point. The data from the Point Cloud XYZ file is read from or written to a point cloud geometry on an FME feature.

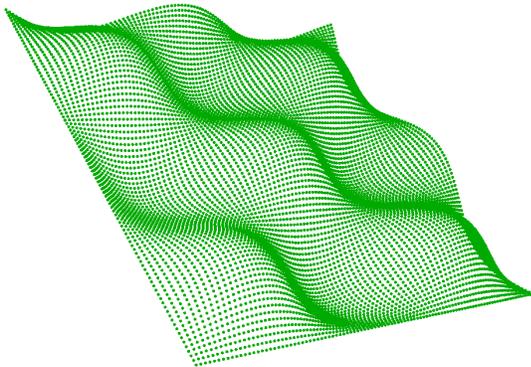
By convention, these files use the .xyz filename extension, but the Point Cloud XYZ reader and writer can use any extension. The writer will write a gzipped file if the extension of the destination file ends with ".gz"

More Information

About Point Clouds

About Point Clouds

A point cloud is a type of geometry that is useful for storing large amounts of data, typically gathered from LIDAR applications. The use of LIDAR allows for fast and accurate collection of data, such as for forestry canopy measurements, or landscape modeling. Point cloud geometry allows for quick and efficient processing of a large collection of vertices in 3D space that represent the external surfaces of objects. Together, these vertices form a model which can be transformed, and visualized. Some operations of the point cloud geometry involve thinning, splitting, and combining to produce a more useable set of vertices.



Associated with each vertex are a number of properties called components, which contains a value describing the point. These component values can be used to classify different sections of the collection of points contained in the point cloud geometry. The specific set of components stored by the point cloud is referred to as the interpretation.

Interpretation	Allowed Values	Description
Intensity	1.7E +/- 308 (15 digits)	The magnitude of the intensity of the pulse return.
Color	0 to 65,535	The color of the object at the point, in RGB color.
Classification	0 to 65,535	The classification value categorizes the points into fields, such as ground, building, water, etc.
Returns	1 - 5	The return value is the return number from a pulse.
Number of returns	1 - 5	The total number of detected returns from a single pulse.
Angle	-90 to 90	The angle of the pulse that the point was scanned at.

Flight line	0 to 4,294,967,295	The flight line number the point was detected in.
Scan Direction	0 and 1	The direction in which a scanning mirror was directed when the point was detected.
Point ID	1 to 65,535	This point ID is indicative of the point origin.
POSIX time	1.7E +/- 308 (15 digits)	Used to express the time, as the number of seconds elapsed since UTC January 1 st , 1970.
User data	0 to 65,535	The user data value is for the user to use.
GPS time and GPS week	GPS Week: 1.7E +/- 308 (15 digits)	Together, these two values express the time since January 6th, 1980. The GPS Week represents a week number, and the GPS time represents the number of seconds into a week.
Flight line Edge	GPS Time: 0 to 65,535 1 for points on the edge, 0 otherwise.	The flight line edge value is a flag for points that lie on the edge of the scan, along the flight line.

Point Cloud XYZ Quick Facts

Format Type Identifier	POINTCLOUDXYZ
Reader/Writer	Both
Licensing Level	Professional
Dependencies	None
Dataset Type	Directory or File
Feature Type	File base name
Typical File Extensions	.xyz
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	No
Transaction Support	No
Geometry Type	pointcloudxyz_type
Encoding Support	Yes

Geometry Support

Geometry Supported?

aggregate	no
circles	no
circular arc	no
donut poly- gon	no
elliptical arc	no
ellipses	no
line	no
None	no

Geometry Supported?

point	no
polygon	no
raster	no
solid	no
surface	no
text	no
z values	yes
point cloud	yes

Point Cloud Components

Point Cloud Component	Data Type	Notes
fmepc_angle	REAL64	
fmepc_classification	UINT8	
fmepc_color_r	UINT16	
fmepc_color_g	UINT16	
fmepc_color_b	UINT16	
fmepc_flight_line_edge	UINT8	
fmepc_flight_line	UINT32	
fmepc_gps_time	REAL64	
fmepc_gps_week	UINT16	
fmepc_intensity	REAL64	
fmepc_number_of_returns	UINT8	
fmepc_point_source_id	UINT32	
fmepc_posix_time	REAL64	
fmepc_return	UINT8	
fmepc_scan_direction	UINT8	
fmepc_user_data	UINT16	

Reader Overview

FME considers a single POINTCLOUDXYZ file to be a dataset. Each dataset contains a single FME point cloud feature.

Reader Directives

The directives listed below are processed by the POINTCLOUDXYZ reader. The suffixes shown are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the Point Cloud XYZ reader is **POINTCLOUDXYZ**.

DATASET

This is the name of one or more XYZ files.

The default extension for Point Cloud XYZ files is **.xyz**

Required/Optional

Required

Mapping File Syntax

POINTCLOUDXYZ_DATASET /usr/data/filename.xyz

* Workbench Parameter

Point Cloud XYZ File(s)

FIELD_NAMES

If the field or column names of the table are specified in the file, set this value to yes and the names will be extracted from the file. Otherwise, the columns of the table are given default names (that is, col0, col1, ... , colN) with the setting no.

Note: If FIELD_NAMES is set to yes, skip_lines should also be set to skip at least one row. You can also set FIELD_NAMES_AFTER_HEADER to yes. See FIELD_NAMES_AFTER_HEADER below for details.

Values

yes | no (default)

Required/Optional

Optional

FIELD_NAMES_AFTER_HEADER

If the column/field names are AFTER the header information instead of BEFORE, then you can set **FIELD_NAMES_AFTER_HEADER** to **yes**. Otherwise, by default, the first line of the file will be used as the column/field names.

This parameter is ignored if FIELD_NAMES is not set, or it is set to no.

Note: If FIELD_NAMES_AFTER_HEADER is set to yes, SKIP_LINES should also be set to skip at least one row.

Values

yes | no

Required/Optional

Optional

SEPARATOR

A special field is listed to identify the separator used to divide the fields in the file.

By default, a space is used; however you can also use any of these one-character separators:

- Comma: ,
- Semicolon: ;
- Vertical bar: |
- Tab: indicated by a backslash (\) followed by a "t"; for example:

POINTCLOUDXYZ_SEPARATOR \t

Note: There must be a space between POINTCLOUDXYZ_SEPARATOR and <separator>. The opening and closing angle brackets are optional.

Required/Optional

Optional

Values

<separator>

SKIP_LINES

This field can be listed to indicate the number of lines to skip at the top of the file. By default, no lines are skipped. This parameter is useful if the Point Cloud XYZ file contains a header line of field names or other descriptive material that should be skipped.

Required/Optional

Optional

Values

<number>

*** Workbench Parameter**

Number of Lines to Skip

SKIP_FOOTER

This field can be listed to indicate the number of footer lines to skip at the bottom of the file. By default, no footer lines are skipped. This parameter is useful if the Point Cloud XYZ file contains a footer line of descriptive material that should be skipped.

Required/Optional

Optional

Values

<number>

DUPLICATE_DELIMS

This field can be listed to indicate if duplicate delimiters are to be treated as a single delimiter. If set to yes then multiple contiguous delimiters are treated as a single de-limiter; otherwise, each delimiter is treated as if it delimits a different field.

Required/Optional

Optional

Values

yes | no

*** Workbench Parameter**

Skip Duplicate Delimiters

COLUMN_TO_PCCOMPONENT

This directive maps each data column in the Point Cloud XYZ file to a component of a point within the point cloud.

Values

<component, columnname pairwise list>

Required/Optional

Required

* Workbench Parameter

Component Mapping

ENCODING

This directive specifies the file encoding to use when reading.

Values

<encoding>

Encodings

UTF-8

UTF-16LE

UTF-16BE

ANSI

BIG5

SJIS

CP437

CP708

CP720

CP737

CP775

CP850

CP852

CP855

CP857

CP860

CP861

CP862

CP863

CP864

CP865

CP866

CP869

CP932

CP936

CP950

CP1250

CP1251

CP1252

CP1253

CP1254

CP1255

CP1256

CP1257

CP1258

ISO8859-1

ISO8859-2

ISO8859-3

ISO8859-4

ISO8859-5
ISO8859-6
ISO8859-7
ISO8859-8
ISO8859-9
ISO8859-13
ISO8859-15

Required/Optional

Optional

* Workbench Parameter

Character Encoding

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the `mitab.dll` in the FME home directory to `mapinfo.dll`.

The syntax of the `MAPINFO_SEARCH_ENVELOPE` directive is:

```
MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

```
MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

Writer Overview

The Point Cloud XYZ Writer writes a point cloud geometry to a XYZ file. Optionally, the point components of the cloud are also output.

Writer Directives

The suffixes shown are prefixed by the current <WriterKeyword> in a mapping file. By default, the <WriterKeyword> for the Point Cloud XYZ reader is POINTCLOUDXYZ.

DATASET

This is the name of a directory containing one or more XYZ files.

The default extension for Point Cloud XYZ files is .xyz. To write gzipped files, use .xyz.gz as the destination file extension.

Required/Optional

Required

Mapping File Syntax

```
POINTCLOUDXYZ_DATASET /usr/data/xyz/output
```

* Workbench Parameter

Destination Point Cloud XYZ Directory

DEF

Defines a Point Cloud XYZ file. The definition contains the file's base name without any of the extensions, followed by the definitions of the attributes. There may be many DEF lines, one for each file to be written.

The syntax of a Point Cloud XYZ DEF line is:

```
<writerKeyword>_DEF <baseName> \  
[<attrName> <attrType>]+
```

The following DEF line directives are supported by the writer:

DEF Line Directives	Value	Required / Optional
POINTCLOUDXYZ_INCLUDE_FIELD_NAME <yes no>	See INCLUDE_FIELD_NAME below for details.	Optional
POINTCLOUDXYZ_SEPARATOR <separator>	See SEPARATOR below for details.	Required
POINTCLOUDXYZ_EXTENSION <extension>	See EXTENSION below for details.	Optional
POINTCLOUDXYZ_ENCODING <encoding>	See ENCODING for details.	Optional
POINTCLOUDXYZ_END_OF_LINE <yes no>	See END_OF_LINE for details.	Optional
POINTCLOUDXYZ_COLUMN_TO_PCCOMPONENT <pc,colName pairwise list>	See COLUMN_TO_PCCOMPONENT for details.	Required
POINTCLOUDXYZ_QUOTE_FIELD_NAMES <yes no if_needed>	See QUOTE_FIELD_NAMES for details	Optional
POINTCLOUDXYZ_WRITE_UTF8_BOM <yes no>	See WRITE_UTF8_BOM for details	Optional

Each of these directives has the same meaning as the global Point Cloud XYZ writer keyword with the same suffix. Any value specified on a DEF line will override values defined for equivalent global directives, as they apply to the table being defined.

Required/Optional

Required

INCLUDE_FIELD_NAME

If set this value to **yes**, the field names will be written as the first line in the output file. If it is set to **no**, column names will not be written to file.

Required/Optional

Optional

Values

Values: <yes (default) | no>

* Workbench Parameter

Output Field Names

SEPARATOR

A special field is listed to identify the separator used to divide the fields in the file. By default, a space is used; however you can also use any of these one-character separators:

- Comma: ,
- Semicolon: ;
- Vertical bar: |
- Tab: indicated by a backslash (\) followed by a "t"; for example:

```
POINTCLOUDXYZ_SEPARATOR \t
```

Note: There must be a space between POINTCLOUDXYZ_SEPARATOR and <separator>. The opening and closing angle brackets are optional.

Required/Optional

Required

Values

<separator>

COLUMN_TO_PCCOMPONENT

This directive specifies the point components to be written and the order of the fields in the output XYZ file.

Values

<component, columnname pairwise list>

Required/Optional

Required

* Workbench Parameter

Component Mapping

EXTENSION

This direction specifies the file extension to be written.

Values

<extension>

Default: **.xyz**

Note: If the extension ends in .gz (e.g., **xyz.gz**) the writer will output gzipped files.

Required/Optional

Optional

*** Workbench Parameter**

Extension

QUOTE_FIELD_NAMES

This directive specifies whether the field names written on the first row of the file are quoted.

Values

yes | no | if_needed (default)

If set to yes, then field names will be quoted. If set to no, field names will not be quoted. If set to if_needed, field names will be quoted only if they contain a separator character.

Required/Optional

Optional

*** Workbench Parameter**

Quote Field Names

ENCODING

This directive specifies the file encoding to use when writing.

Values

<encoding>

Default: System

Encodings

UTF-8

UTF-16LE

UTF-16BE

ANSI

BIG5

SJIS

CP437

CP708

CP720

CP737

CP775

CP850

CP852

CP855

CP857

CP860

CP861

CP862

CP863

CP864

CP865

CP866

CP869
CP932
CP936
CP950
CP1250
CP1251
CP1252
CP1253
CP1254
CP1255
CP1256
CP1257
CP1258
ISO8859-1
ISO8859-2
ISO8859-3
ISO8859-4
ISO8859-5
ISO8859-6
ISO8859-7
ISO8859-8
ISO8859-9
ISO8859-13
ISO8859-15

Required/Optional

Optional

*** Workbench Parameter**

Character Encoding

END_OF_LINE

This directive specifies the end of line character to use when writing.

Values

Macintosh | Windows | Unix | System (default)

Required/Optional

Optional

*** Workbench Parameter**

Line Termination

WRITE_UTF8_BOM

This directive specifies whether the byte order mark for UTF-encoded files should be written at the beginning.

This option only has an effect when the encoding is set to a UTF encoding.

Values

yes (default) | no

Required/Optional

Optional

*** Workbench Parameter**

Write UTF-8 Byte Order Mark

PostGIS Reader/Writer

Format Notes:

This format is not supported by FME Base Edition.

Overview

PostGIS is a geometric layer over a PostgreSQL Object-Relational Database Management System (ORDBMS) that provides geometry and Spatial Reference System (SRS) handling. The PostGIS reader/writer module enables FME to read geometric PostGIS data as well as underlying attribute data stored in PostgreSQL.

The PostGIS reader/writer is specifically designed to handle the geometric and SRS portions of the data. When reading attribute-only tables from PostgreSQL, the PostgreSQL reader/writer should be used instead. The PostGIS reader/writer communicates directly with the PostgreSQL libpq interface for maximum throughput.

This chapter assumes familiarity with PostGIS and PostgreSQL, the attribute and geometry types supported, and its indexing mechanisms.

For more information, please see the PostgreSQL home at

<http://www.postgresql.org/>

and the PostGIS home at

<http://postgis.refractory.net/>

PostGIS Quick Facts

Format Type Identifier	PostGIS
Reader/Writer	Both
Licensing Level	Professional
Dependencies	None
Dataset Type	Database
Feature Type	Table name
Typical File Extensions	None
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Always
Schema Required	Yes
Transaction Support	Yes
Geometry Type	postgis_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	yes		polygon	yes
circular arc	yes		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	yes		z values	yes
none	yes			

Reader Overview

FME considers a PostGIS dataset to be a database containing a collection of relational tables together with their corresponding geometries. The tables must be defined in the mapping file before they can be read. Arbitrary **WHERE** clauses and joins are fully supported, as well as an entire arbitrary SQL **SELECT** statement; however, the user then assumes responsibility for the correctness of the statement or clause including quoting where necessary. Support for **@SQL** and **@Relate** functions has also been added.

When reading from the PostGIS/PostgreSQL database, each table is considered a feature type in FME and each row of a table at least one feature in FME. In the case of heterogeneous geometry collections, they may become more than one FME feature.

The basic reading process involves opening a connection to the database, querying metadata, and querying data. The data is read using a text cursor and rows are fetched to the client machine in batches of 10000 by default. There is one cursor per input table.

If NULL geometries are read, they are treated as non-geometry features and the attributes are preserved.

Table and column names are truncated at 64 characters. If duplicate names are produced by truncation, the behavior is undetermined. Please ensure that table names comply with PostgreSQL naming conventions.

Spaces and special characters are permissible in both table and column names. Case sensitivity has also been implemented, so table and column names are no longer changed to lowercase.

Table listing support when using the PostGIS settings boxes has been improved to avoid errors with schemas and tables that do not exist, or are inconsistent with the PostGIS metadata.

UNICODE support has been added to work with a client's system encoding. Although there is no way to explicitly specify the encoding, the client is assumed to have entered data and created tables and columns in the encoding of their operating system. Multiple system encodings are now supported via the native PostgreSQL conversions between client and server, particularly if the server encoding is set to UNICODE.

Older schema directives have been removed and qualified table naming is now supported in the form **<schema-name>.<tablename>**. Additionally, the schema search path is now read and interpreted to determine a user's default schema when writing and the available schema to read from when reading. Failing a valid schema search path, the default public schema will be used for newer databases.

Reader Directives

The directives that are processed by the PostGIS reader are listed below. The suffixes shown are prefixed by the current **<ReaderKeyword>**_ in a mapping file. By default, the **<ReaderKeyword>** for the PostGIS reader is **POST-GIS_IN**.

DATASET/DATABASE

This directive specifies the name of the PostGIS-enabled PostgreSQL database. The database must exist in the ORDBMS.

Required/Optional

Required

Mapping File Syntax

```
POSTGIS_DATASET testdb
```

*** Workbench Parameter**

Source PostGIS Dataset

HOST

This directive specifies the machine running the PostGIS/PostgreSQL ORDBMS as either an IP address or host name. The database must have proper permissions and be set up to accept TCP/IP connections if connecting from a remote machine.

Required/Optional

Required

Mapping File Syntax

```
POSTGIS_IN_HOST myserver
```

*** Workbench Parameter**

Host

PORT

When connecting remotely, this directive specifies the TCP/IP port on which to connect to the ORDBMS service. The default port is 5432.

Required/Optional

Required

Mapping File Syntax

```
POSTGIS_IN_PORT 5432
```

*** Workbench Parameter**

Port

USER_NAME

The name of user who will access the database. The named user must exist with appropriate PostgreSQL permissions.

The default user name is `postgres`.

Required/Optional

Required

Mapping File Syntax

```
POSTGIS_IN_USER_NAME postgres
```

* Workbench Parameter

Username

PASSWORD

The password of the user accessing the database. This directive is optional when using a trusted connection.

Other authentication types such as password or MD5 require this parameter to be set.

Required/Optional

Optional

Mapping File Syntax

```
POSTGIS_IN_PASSWORD secret
```

* Workbench Parameter

Password

DEF

The syntax of the definition is:

```
POSTGIS_DEF <tableName> \  
    [postgis_where_clause <whereClause>] \  
[<fieldName> <fieldType>] +
```

OR

```
POSTGIS_DEF <queryName> \  
    [postgis_sql_statement <sqlQuery>]
```

The **<tableName>** must match a PostGIS table in the database. This will be used as the feature type of all the features read from the table. The exception to this rule is when using the `sql_statement` directive. In this case, the **DEF** name may be any valid alphabetic identifier; it does not have to be an existing table name – rather, it is an identifier for the custom SQL query. The feature type of all the features returned from the SQL query are given the query name.

The **<fieldType>** of each field must be given, but it is not verified against the database definition for the field. In effect, it is ignored.

The exception to this is the **geometry** field type which is not placed in the **DEF**. This is stored separately in the **geometry_columns** table of the PostgreSQL database which maps geometry information to the database and table name.

The definition allows specification of separate search parameters for each table. If any of the configuration parameters are given, they will override, for that table, whatever global values have been specified by the reader directives listed above. If any of these parameters is not specified, the global values will be used.

The following table summarizes the definition line configuration parameters:

Parameter	Contents
where_clause	This specifies the SQL WHERE clause applied to the

Parameter	Contents
	attributes of the layer's features to limit the set of features returned. If this is not specified, then all the tuples are returned. This directive will be ignored if the <code>sql_statement</code> is present.
<code>sql_statement</code>	This specifies an SQL SELECT query to be used as the source for the results. If this is specified, the PostGIS reader will execute the query, and use the resulting rows as the features instead of reading from the table <code><queryName></code> . All returned features will have a feature type of <code><queryName></code> , and attributes for all columns selected by the query. All parameters that specify a spatial constraint are ignored if an <code>sql_statement</code> is supplied. If selecting a geometry column from a PostGIS table, avoid the use of geometry column format modifiers such as <code>AsBinary()</code> , <code>AsText()</code> , <code>AsWKT()</code> , or <code>ASWKB()</code> since this obscures the fact that we have a geometry column and not just some text or byte attribute column.

If no `<whereClause>` is specified, all rows in the table will be read and returned as individual features. If a `<whereClause>` is specified, only those rows that are selected by the clause will be read. Note that the `<whereClause>` does not include the word "where".

The PostGIS reader allows one to use the `sql_statement` parameter to specify an arbitrary SQL `SELECT` query on the DEF line. If this is specified, FME will execute the query, and use each row of data returned from the query to define a feature. Each of these features will be given the feature type named in the DEF line, and will contain attributes for every column returned by the `SELECT`. In this case, all DEF line parameters regarding a `WHERE` clause or spatial querying are ignored, as it is possible to embed this information directly in the text of the `<sqlQuery>`.

The following example selects rows from the table `ROADS`, placing the resulting data into FME features with a feature type of `MYROADS`. Imagine that `ROADS` defines the geometry for the roads, and has a numeric field named `ID`, a text field named `NAME` and a geometry column named `GEOM`.

```
POSTGIS_DEF MYROADS \
  sql_statement 'SELECT id, name, geom FROM ROADS'
```

Required/Optional

Required

IDs

This optional specification is used to limit the available and defined database tables files that will be read. If no IDs are specified, then no tables are read. The syntax of the IDs directive is:

```
POSTGIS_IDS <featureType1> \
  <featureType2> \
  <featureTypeN>
```

The feature types must match those used in DEF lines.

The example below selects only the `ROADS` table for input during a translation:

POSTGIS_IDS ROADS

Required/Optional

Optional

MINX, MINY, MAXX, MAXY

These directives when used together specify the spatial extent of the feature retrieval. Only features that interact with the bounding box defined by these directive values are returned.

If this is not supplied, all features will be returned. If either **min** value is greater than the corresponding **max** value, the values will be swapped. If less than the entire set of four values are supplied, the supplied values will be ignored and all features will be returned.

The syntax of the directives is:

```
POSTGIS_IN_MINX <minX>
POSTGIS_IN_MINY <minY>
POSTGIS_IN_MAXX <maxX>
POSTGIS_IN_MAXY <maxY>
```

The example below selects a small area for extraction:

```
POSTGIS_IN_MINX 25.6
POSTGIS_IN_MINY 59.0
POSTGIS_IN_MAXX 79.2
POSTGIS_IN_MAXY 124.5
```

Required/Optional

Optional

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

Mapping File Example

The example below selects a small area for extraction:

```
POSTGIS_IN_SEARCH_ENVELOPE 25.6 59.0 79.2 124.5
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The `COORDINATE_SYSTEM` directive, which specifies the coordinate system associated with the data to be read, must always be set if the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` to the reader `COORDINATE_SYSTEM` prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the `SEARCH_ENVELOPE` directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

SEARCH_METHOD

This directive is used to specify the spatial relationship between the provided bounding box and the geometries in the geometry column of the table.

There are two types of operation:

- Maximum Bounding Rectangle (MBR) operations determine adherence to a given spatial relationship using only the bounding box of the geometry.
- Full spatial operations will use the actual geometry itself.

Full spatial relationship comparisons are only available if GEOS is enabled on the PostGIS server. If not, all envelope comparisons will be made using the default MBR operation `MBR_OVERLAPS`.

Required/Optional

Optional

Mapping File Syntax

```
POSTGIS_IN_SEARCH_METHOD <spatial_relationship>
```

* Workbench Parameter

Search Method

FEATURES_PER_FETCH

To avoid loading all the features in memory at once when reading a large dataset, cursors are used to retrieve the rows from the database. This optional directive specifies the number of rows to be read at one time from the cursor for a given query.

The default is 10000 rows and should be sufficient in most cases. However this may need to be lowered or raised depending on the capabilities of the specific hardware in use and the data being read.

Required/Optional

Optional

Mapping File Syntax

The example below selects a small set of features per extraction:

```
POSTGIS_IN_FEATURES_PER_FETCH 5000
```

*** Workbench Parameter**

Number of Records to Fetch at a Time

RETRIEVE_ALL_SCHEMAS

This specification tells the reader to retrieve the names and the schemas of all the tables in the source database.

This directive is only applicable when generating a mapping file, generating a workspace, or when retrieving schemas in a FME Objects application.

Values

YES | NO

If this value is not specified, then it is assumed to be No. When set to Yes, indicates to the reader to return all the schemas of the tables in the database.

Required/Optional

Optional

Mapping File Syntax

```
POSTGIS_RETRIEVE_ALL_SCHEMAS Yes
```

RETRIEVE_ALL_TABLE_NAMES

This specification tells the reader to retrieve only the table names of all the tables in the source database.

This directive is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

Note: If RETRIEVE_ALL_SCHEMAS is also set to Yes, then RETRIEVE_ALL_SCHEMAS takes precedence.

Required/Optional

Optional

Values

YES | NO

If this value is not specified, then it is assumed to be No.

Required/Optional

Optional

Mapping File Syntax

`POSTGIS_RETRIEVE_ALL_TABLE_NAMES` Yes

DISABLE_COLLECTION_SPLITTING

Specifies that the reader should not split *single item multis* or *geometrycollections* with any number of parts.

If geometrycollections are split, each part is tagged with `postgis_collection_id` and `postgis_collection_part_id` attributes.

Required/Optional

Optional

Values

YES (default) | NO

Not setting this directive is equivalent to setting it to NO.

Mapping File Syntax

`POSTGIS_DISABLE_COLLECTION_SPLITTING` Yes

USE_TRUE_POSTGIS_TYPES

Required/Optional

Optional

Values

YES (default) | NO

If not present or not set to YES, only the legacy postgis types (*postgis_point*, *postgis_line*, *postgis_area*, *postgis_geometrycollection*) will be attached to features.

If set to YES *postgis_line* is replaced by *postgis_linestring*, *postgis_area* is replaced by *postgis_polygon*, and types representing curve supporting geometries are also available.

Mapping File Syntax

`POSTGIS_USE_TRUE_POSTGIS_TYPES` Yes

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The PostGIS writer module stores both geometry and attributes into an PostgreSQL database. Note that attribute case is preserved, unless the option to lowercase attributes is set to 'yes'. The PostGIS writer provides the following capabilities:

- **Transaction Support:** The PostGIS writer provides transaction support that eases the data loading process. Occasionally, a data load operation terminates prematurely due to data difficulties. The transaction support provides a mechanism for reloading corrected data without data loss or duplication.
- **Index Creation:** The PostGIS writer can set up and populate indexes as part of the loading process. By default, a GiST index is created on the primary geometry column of a table and not on any other columns. Additional columns can be individually indexed. Composite column indexes are not supported at this time.
- **Bulk Loading:** By default, the PostGIS writer uses a bulk loading technique to ensure speedy data loading.

Writer Directives

The directives that are processed by the PostGIS writer are listed below. The suffixes shown are prefixed by the current `<WriterKeyword>_` in a mapping file. By default, the `<WriterKeyword>` for the PostGIS writer is `POSTGIS_OUT`.

DATASET/DATABASE, HOST, PORT, USER_NAME, PASSWORD

These directives operate in the same manner as they do for the PostGIS reader.

DEF

Each PostGIS table must be defined before it can be written. The general form of a PostGIS definition statement is:

```
POSTGIS_DEF <tableName> \  
  postgis_type <postgis_type> \  
  [postgis_mode (insert|update|delete|inherit_from_writer)] \  
  [postgis_spatial_column <column>] \  
  [postgis_spatial_column_type (geometry|geography)] \  
  [postgis_srid <srid>] \  
  [postgis_drop_table (yes|no)] \  
  [postgis_truncate_table (yes|no)] \  
  [postgis_create_with_oids (yes|no)] \  
  [postgis_create_with_gist_index (yes|no)] \  
  [postgis_vacuum_analyze (yes|no)] \  
  [<fieldName> <fieldType>][,<indexType>]*
```

The table definition allows control of the table that will be created. If the table already exists, the majority of the `postgis_` parameters will be ignored and need not be given. If the fields and types are listed, they must match those in the database.

If the table does not exist, then the field names and types are used to first create the table. In any case, if a `<field-
Type>` is given, it may be any field type supported by the target database.

Required/Optional

Required

Configuration Parameters

The configuration parameters present on the definition line are described in the following table:

Parameter	Contents
tableName	<p>The name of the table to be written. If a table with the specified name exists, it will be overwritten if either the postgis_overwrite_table DEF line parameter set to YES or if the global writer directive type postgis_out_overwrite is set to YES. Otherwise the table will be appended. Valid values for table names include any character string devoid of SQL-offensive characters and less than 32 characters in length.</p>
postgis_type	<p>The type of geometric entity to which the created table will be constrained. The valid values for the type are listed below:</p> <ul style="list-style-type: none"> postgis_point postgis_multipoint postgis_linestring postgis_multilinestring postgis_circularstring postgis_compoundcurve postgis_multicurve postgis_polygon postgis_multipolygon postgis_curvepolygon postgis_multisurface postgis_geometrycollection postgis_none postgis_geometry (allow anything) <p>If a collection is written to a table constrained to single pieces of geometry, the collection will automatically be broken apart for writing. Conversely, if a single piece of geometry is written to a table constrained to one of the collection types, it will automatically be wrapped in a collection.</p>
postgis_mode	<p>The the default operation mode of the feature type in terms of the types of SQL statements sent to the database. Valid values are INSERT, UPDATE, DELETE and INHERIT_FROM_WRITER. Note that INSERT mode allows for only INSERT operations where as UPDATE and DELETE can be overwritten at the feature levels. INHERIT_FROM_WRITER simply indicates to take this value from the writer level and not to override it at the feature type level.</p> <p>Default: INHERIT_FROM_WRITER</p>
postgis_spatial_column	<p>This specifies the name of the column to be created that will hold the spatial data when creating a new PostGIS table.</p> <p>Default: geom</p>

Parameter	Contents
postgis_spatial_column_type	<p>This specifies the name of the column to be created that will hold the spatial data when creating a new PostGIS table. Valid values are geography and geometry. Default: geometry</p>
postgis_srid	<p>This specifies the spatial referencing information for the geometry in the table. By default, this value is INHERIT_FROM_WRITER which uses the conversion of the FME coordinate system of the writer into an SRID as the SRID for the given table.</p> <p>Alternatively, a specific integer SRID value may be specified. Specified SRID values should correspond to an existing the spatial reference identifier value stored in the (SRID) column in the global table spatial_ref_sys.</p> <p>All geometry within a given table must have the same spatial referencing.</p> <p>If postgis_srid is not specified, tables will be created with the SRID of the writer coordinate system.</p> <p>If empty SRIDs are desired, the value for the SRID field can be set to -1 indicating no spatial reference system.</p>
postgis_drop_table	<p>This specifies that if the table exists by this name, it should be dropped and recreated before any features are written to it.</p> <p>This parameter, along with postgis_truncate_table, deprecates the older postgis_overwrite_table parameter.</p>
postgis_truncate_table	<p>This specifies that if the table exists by this name, it should be truncated before any features are written to it.</p> <p>This parameter, along with postgis_drop_table, deprecates the older postgis_overwrite_table parameter.</p>
postgis_create_with_oids	<p>Create the table including a system OID column as a unique identifier. If set to no, then the OID column is not created. Default: yes</p>
postgis_create_with_gist_index	<p>Create a GiST index on the geometry column of the table (as long as one exists).The indexing of the geometry column is required for spatial query performance. Default: yes</p>
postgis_vacuum_analyze	<p>Perform the database function to vacuum and analyze the table once successfully written. This will build statistics for the table. Default: yes</p>
fieldName	<p>The name of the field to be written. Valid values for field</p>

Parameter	Contents
	name include any character string devoid of SQL-offensive characters and less than 32 characters in length.
fieldType	<p>The type of a column in a table. The valid values for the field type are listed below:</p> <ul style="list-style-type: none"> bool char(width) bpchar(width) varchar(width) int2 int4 int8 text bytea oid serial float4 float8 money date time timetz timestamp timestampz
indexType	<p>The type of index to create on the given field. The valid values for the index type are listed below:</p> <ul style="list-style-type: none"> BTREE (default attribute index) RTREE HASH PRIKEY (primary key)

START_TRANSACTION

This statement tells the PostGIS writer module when to start actually writing features into the database.

The PostGIS writer does not write any features until the feature number of features are skipped, and then it begins writing the following features. Normally, the value specified is zero – a non-zero value is only specified when a data load operation is being resumed after failing partway through.

Required/Optional

Optional

Mapping File Syntax

POSTGIS_OUT_START_TRANSACTION 0

* Workbench Parameter

Starting Feature

TRANSACTION_INTERVAL

This directive determines the number of features that FME will place in each transaction before a transaction is committed to the database.

If the `POSTGIS_OUT_TRANSACTION_INTERVAL` statement is not specified, then a value of 1000 is used as the transaction interval.

Required/Optional

Optional

Mapping File Syntax

```
POSTGIS_OUT_TRANSACTION_INTERVAL 2000
```

Workbench Parameter: *Features Per Transaction*

BULK_COPY

This statement tells the PostGIS writer module to insert data into the database using either SQL `INSERT` statements or the SQL `COPY` command. The default option is the bulk copy using the `COPY` command, which yields the best performance.

Required/Optional

Optional

Mapping File Syntax

```
POSTGIS_OUT_BULK_COPY YES
```

Note: The bulk delimiter escaping has improved, and therefore it is no longer necessary as a backup measure. However, if individual inserts are desired, this option can be set to NO.

* Workbench Parameter

Bulk COPY Insert

WRITER_MODE

This directive informs the PostGIS writer which SQL operations will be performed by default by this writer.

This operation can be set to `INSERT`, `UPDATE` or `DELETE`. The default writer-level value for this operation can be overwritten at the feature type or table level. The corresponding feature type `DEF` parameter name is called `POSTGIS_MODE`. It has the same valid options as the writer-level mode, as well as the value `INHERIT_FROM_WRITER` (which causes the writer-level mode to be inherited by the feature type as the default for features contained in that table).

The operation can also be set specifically for individual features. Note that when the writer mode is set to `INSERT`, this prevents the mode from being interpreted from individual features and all features are inserted unless otherwise marked as update or delete features. These are skipped.

Required/Optional

Optional

Mapping File Syntax

`POSTGIS_OUT_WRITER_MODE INSERT`

If this directive is not specified, then a value of `INSERT` is given.

* Workbench Parameter

Writer Mode

Note: For more information on this directive, see the *Database Writer Mode*.

GENERIC_GEOMETRY

This directive applies at generation time, not at translation time.

The default value of `NO` indicates that we want the previous behavior of creating geometrically constrained spatial columns on the destination tables. For example, a `POINT` spatial table would be restricted only to points.

Now we have the option to create generic or non-geometrically constrained spatial columns. This means you can insert multiple geometry types into one table. Specifically the spatial column is created to have the generic type `GEOMETRY` and there are no constraints placed on the geometry types allowed.

Required/Optional

Optional

If the `POSTGIS_OUT_GENERIC_GEOMETRY` statement is not specified, then a value of `NO` is given.

Mapping File Syntax

`POSTGIS_OUT_GENERIC_GEOMETRY YES`

SPATIAL_COLUMN_TYPE

The default value of `geometry` indicates that we want to create planar (geometry) spatial columns on the destination tables by default. Setting its value to `geography`, however, would mean that we want to create geodetic (geography) spatial columns. This value can also be set specifically for individual feature types.

This directive applies only at generation time.

Required/Optional

Optional

SPATIAL_COLUMN_NAME

The default value of `geom` indicates that we want to create a spatial column with the name "geom" on the destination tables by default. This value can also be set specifically for individual feature types.

This directive only applies at generation time.

Feature Representation

Features read from PostGIS consist of a series of attribute values and geometry. The feature type of each feature is as defined on its DEF line. The geometry object model in PostGIS follows the *OGIS Simple Features Specification 1.1*. For more information see <http://www.opengis.org>.

Features written to the database have the destination table as their feature type, and attributes as defined on the `DEF` line.

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Attribute Name	Contents
postgis_type	<p>The type of geometric entity stored within the feature. The valid values for the object model are listed below:</p> <ul style="list-style-type: none"> postgis_point postgis_multipoint postgis_linestring postgis_multilinestring postgis_circularstring postgis_compoundcurve postgis_multicurve postgis_polygon postgis_multipolygon postgis_curvepolygon postgis_multisurface postgis_geometrycollection postgis_none

Features read from, or written to, PostGIS also have an attribute for each column in the database table. The feature attribute name will be the same as the source or destination column name. The attribute and column names are case-sensitive.

No Coordinates

postgis_type: postgis_none

Features with no coordinates are tagged with this value when reading from PostGIS. Note that when reading or writing attribute-only data tables, the PostgreSQL reader/writer should be used instead. Note also that this is not a valid OGC type.

Point

postgis_type: postgis_point

Features tagged with this value consist of a single point.

MultiPoint

postgis_type: postgis_multipoint

Features tagged with this value consist of a collection of points.

LineString

postgis_type: postgis_linestring

Features tagged with this value consist of a single line.

MultiLineString

postgis_type: postgis_multilinestring

Features tagged with this value consist of a collection of lines.

CircularString

postgis_type: postgis_circularstring

Features tagged with this value consist of a path of circular arcs.

CompoundCurve

postgis_type: postgis_compoundcurve

Features tagged with this value consist of a path of lines and/or circular arcs.

MultiCurve

postgis_type: postgis_multicurve

Features tagged with this value consist of a collection of lines and paths of circular arcs.

Note that paths containing both lines and circular arcs may not be stored in a MultiCurve.

Polygon

postgis_type: postgis_polygon

Features tagged with this value consist of a linear polygon or donut.

MutiPolygon

postgis_type: postgis_multipolygon

Features tagged with this value consist of a collection of linear polygons and/or donuts.

CurvePolygon

postgis_type: postgis_curvepolygon

Features tagged with this value consist of a linear or curved polygon or donut.

MultiSurface

postgis_type: postgis_multisurface

Features tagged with this value consist of a collection of linear and/or curved polygons and/or donuts.

GeometryCollection

postgis_type: postgis_geometrycollection

Features tagged with this value consist of a possibly heterogeneous aggregate.

When writing to PostGIS, the `postgis_type` can be manually set to `postgis_geometrycollection` and features will be combined into collections based on the `postgis_collection_id` attribute.

Geometry

postgis_type: postgis_geometry

Although not a valid geometry type on an individual feature, this type may be set for the destination geometry column type to indicate that any geometry is allowable in that column. If the writer directive `GENERIC_GEOMETRY` is specified at generation time, all destination feature types will have geometry columns of this type. Alternatively, although it will not happen by default, this type can be specified on any one or more destination feature types manually to create generic geometry columns on those specific tables.

Note that although the geometry column is explicitly determined by the user when writing, the reader will attempt to determine the geometry column name and type using the PostGIS metadata tables.

In cases where the PostGIS reader is applied to databases lacking the proper metadata tables or entries, then the first column whose type would allow for geometry storage is taken as the geometry column. This decision may cause the translation to fail if the column does not indeed contain geometry. To work around this case, please use the PostgreSQL reader instead, which ignores geometry columns. Alternatively, if there are multiple geometry columns or a mixture of blob columns and geometry columns and the geometry is still desired, try formatting a custom SQL query that selects the geometry column the first column.

Note also that geometry columns can be stored in either geometry or blob columns but these may contain other data as well. Geometry columns that do not correspond to the geometry of the feature will be read as hex-encoded strings and blobs will be read as raw bytes.

Troubleshooting Tips

Problems sometimes arise when attempting to connect to a PostGIS/PostgreSQL database. This is almost always due to a misconfiguration in the user's environment.

The following suggestions can often help detect and overcome such problems.

- Ensure you can connect to the database with the host, port, database, user name, and password using psql or pgAdmin. See PostgreSQL documentation for proper security and connection information, and for the usage of the psql utility.
- If you try to list the tables and nothing happens, check the log file. There may have been an underlying error that didn't generate a dialog. Usually this means a parameter does not exist or permissions are not sufficient to access the requested resource.
- In most cases, the POSTGIS_DATABASE directive should be left with blank values, with the POSTGIS_DATASET directive containing the name of the PostGIS database.
- When using a UNIX operating system, the environment variables PGHOST, PGPORT, PGDATABASE, PGUSER and PGPASSWORD can be used to specify the PostgreSQL connection parameters.
- If the table list in the PostGIS reader input settings box does not display your table, try typing the name with the schema prefix (e.g., public.mytable). If this works, then your table may not be properly registered in the PostGIS metadata tables or it may not have a geometry column.
- If the table list in the PostGIS reader input settings box lists your table, but you receive an error message that the table does not exist when you run the translation, then it is likely that the PostgreSQL table has been deleted without updating the PostGIS metadata tables. Orphaned metadata may continue to exist in the PostGIS metadata tables. It is suggested that the PostGIS metadata table for the geometry columns be corrected to match only existing PostgreSQL tables.
- If your data ends up looking garbled using a given encoding, it may be because the encoding of the data does not match your system encoding. These must match because FME uses the system encoding to set the encoding of the PostgreSQL client, and then allows the database to convert encodings if necessary between the client and server.

Connecting to PostgreSQL/PostGIS tables in another user's schema

FME uses the Postgres search path to determine which schemas' tables to show in the table list. To set a user's search path for a session:

```
SET search_path TO "$user",public;  
SET search_path TO "$user",public,schema2,schema3;
```

To set a user's search path for all future sessions:

```
ALTER USER <username> SET search_path TO "$user",public;  
ALTER USER <username> SET search_path TO "$user",public,schema2,schema3;
```

To see the current search path:

```
show search_path;
```

PostgreSQL Reader/Writer

Format Notes:

This format is not supported by FME Base Edition.

Overview

PostgreSQL is an Object-Relational Database Management System (ORDBMS) that stores attribute information. The PostgreSQL reader/writer module enables FME to read and write PostgreSQL attribute data.

The PostgreSQL reader/writer is specifically designed to handle the attribute portion of the data in the database. When reading geometric or Spatial Reference System (SRS) data stored in a PostGIS layer over PostgreSQL, the PostGIS reader/writer module should be used instead. The PostgreSQL reader/writer communicates directly with the PostgreSQL **libpq** interface for maximum throughput.

This chapter assumes familiarity with PostgreSQL, SQL, the attribute types supported, and its indexing mechanisms.

For more information, please see the PostgreSQL home at

<http://www.postgresql.org/>

PostgreSQL Quick Facts

Format Type Identifier	PostgreSQL
Reader/Writer	Both
Licensing Level	Professional
Dependencies	None
Dataset Type	Database
Feature Type	Table name
Typical File Extensions	None
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	Yes
Geometry Type	postgis_type

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	point	no
circles	no	polygon	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
circular arc	no		raster	no
donut polygon	no		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	no		z values	no
none	yes			

Reader Overview

FME considers a PostgreSQL dataset to be a database containing a collection of relational tables together with their corresponding geometries. The tables must be defined in the mapping file before they can be read. Arbitrary **WHERE** clauses and joins are fully supported, as well as an entire arbitrary SQL **SELECT** statement; however, the user then assumes responsibility for the correctness of the statement or clause including quoting where necessary. Support for the **@SQL** and **@Relate** FME functions has also been added.

When reading from the PostgreSQL database, each table is considered a feature type in FME and each row of a table at least one feature in FME.

The basic reading process involves opening a connection to the database, querying metadata, and querying data. The data is read using a text cursor and rows are fetched to the client machine in batches of 10000 by default. There is one cursor per input table.

Table and column names are truncated at 64 characters. If duplicate names are produced by truncation, the behavior is undetermined. Please ensure that table names comply with PostgreSQL naming conventions.

Spaces and special characters are permissible in both table and column names. Case sensitivity has also been implemented, so table and column names are no longer changed to lowercase.

Table listing support when using the PostgreSQL settings boxes has been improved to avoid errors with schemas.

UNICODE support has been added to work with a client's system encoding. Although there is no way to specify the encoding explicitly, the client is assumed to have entered data and created tables and columns in the encoding of their operating system. Multiple system encodings are now supported via the native PostgreSQL conversions between client and server, particularly if the server encoding is set to UNICODE.

Older schema keywords have been removed and qualified table naming is now supported in the form **<schema-name>.<tablename>**. Additionally, the schema search path is now read and interpreted to determine a user's default schema when writing and the available schema to read from when reading. Failing a valid schema search path, the default public schema will be used for newer databases.

Reader Directives

The directives that are processed by the PostgreSQL reader are listed below. The suffixes shown are prefixed by the current **<ReaderKeyword>_** in a mapping file. By default, the **<ReaderKeyword>** for the PostgreSQL reader is **POSTGRES_IN**.

DATASET/DATABASE

Required/Optional: *Required*

This specifies the name of the PostgreSQL database. The database must exist in the ORDBMS.

POSTGRES_DATASET testdb

Workbench Parameter: *Source PostgreSQL Dataset*

HOST

Required/Optional: *Required*

This specifies the machine running the PostgreSQL ORDBMS as either an IP address or host name. The database must have proper permissions and be set up to accept TCP/IP connections if connecting from a remote machine.

```
POSTGRES_IN_HOST myserver
```

Workbench Parameter: *Host*

PORT

Required/Optional: *Required*

When connecting remotely, this specifies the TCP/IP port on which to connect to the ORDBMS service. The default port is 5432.

```
POSTGRES_IN_PORT 5432
```

Workbench Parameter: *Port*

USER_NAME

Required/Optional: *Required*

The name of user who will access the database. The named user must exist with appropriate PostgreSQL permissions. The default user name is `postgres`.

```
POSTGRES_IN_USER_NAME postgres
```

Workbench Parameter: *User Name*

PASSWORD

Required/Optional: *Optional*

The password of the user accessing the database. This parameter is optional when using a trusted connection. Other authentication types such as password or MD5 require this parameter to be set.

```
POSTGRES_IN_PASSWORD secret
```

Workbench Parameter: *Password*

DEF

Required/Optional: *Required*

The syntax of the definition is:

```
POSTGRES_DEF <tableName> \  
    [postgres_where_clause <whereClause>] \  
[<fieldName> <fieldType>] +
```

OR

```
POSTGRES_DEF <queryName> \  
    [postgres_sql_statement <sqlQuery>] \  
[<fieldName> <fieldType>] +
```

The `<tableName>` must match a PostgreSQL table in the database. This will be used as the feature type of all the features read from the table. The exception to this rule is when using the `sql_statement` keyword. In this case, the `DEF` name may be any valid alphabetic identifier; it does not have to be an existing table name – rather, it is an identifier for the custom SQL query. The feature type of all the features returned from the SQL query are given the query name.

The `<fieldType>` of each field must be given, but it is not verified against the database definition for the field. In effect, it is ignored.

The definition allows specification of separate search parameters for each table. If any of the configuration parameters are given, they will override, for that table, whatever global values have been specified by the reader keywords listed above. If any of these parameters is not specified, the global values will be used.

The following table summarizes the definition line configuration parameters:

Parameter	Contents
where_clause	This specifies the SQL WHERE clause applied to the attributes of the layer's features to limit the set of features returned. If this is not specified, then all the tuples are returned. This keyword will be ignored if the sql_statement is present.
sql_statement	This specifies an SQL SELECT query to be used as the source for the results. If this is specified, the PostgreSQL reader will execute the query, and use the resulting rows as the features instead of reading from the table <queryName>. All returned features will have a feature type of <queryName>, and attributes for all columns selected by the query.

If no **<whereClause>** is specified, all rows in the table will be read and returned as individual features. If a **<whereClause>** is specified, only those rows that are selected by the clause will be read. Note that the **<whereClause>** does not include the word **WHERE**.

The PostgreSQL reader allows one to use the **sql_statement** parameter to specify an arbitrary SQL **SELECT** query on the DEF line. If this is specified, FME will execute the query, and use each row of data returned from the query to define a feature. Each of these features will be given the feature type named in the **DEF** line, and will contain attributes for every column returned by the **SELECT**. In this case, all **DEF** line parameters regarding a **WHERE** clause or spatial querying are ignored, as it is possible to embed this information directly in the text of the **<sqlQuery>**.

The following example selects rows from the table **ROADS**, placing the resulting data into FME features with a feature type of **MYROADS**. Imagine that **ROADS** defines the geometry for the roads, and has a numeric field named **ID**, a text field named **NAME** and a geometry column named **GEOM**.

```
POSTGRES_DEF MYROADS \
    sql_statement 'SELECT id, name FROM ROADS'
```

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined database tables files that will be read. If no **IDs** are specified, then no tables are read. The syntax of the **IDs** keyword is:

```
POSTGRES_IDS <featureType1> \
    <featureType2> \
    <featureTypeN>
```

The feature types must match those used in **DEF** lines.

The example below selects only the **ROADS** table for input during a translation:

```
POSTGRES_IDS ROADS
```

FEATURES_PER_FETCH

Required/Optional: *Optional*

In order to avoid loading all the features in memory at once when reading a large dataset, cursors are used to retrieve the rows from the database. This optional keyword specifies the number of rows to be read at one time from the cursor for a given query. The default is 10000 rows and should be sufficient in most cases. However this may need to be lowered or raised depending on the capabilities of the specific hardware in use and the data being read.

The example below selects a small set of features per extraction:

```
POSTGRES_IN_FEATURES_PER_FETCH 5000
```

Workbench Parameter: *Number Of Features To Fetch At A Time*

RETRIEVE_ALL_SCHEMAS

Required/Optional: *Optional*

This specification is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

This optional specification is used to tell the reader to retrieve the names and the schemas of all the tables in the source database. If this value is not specified, then it is assumed to be No. When set to Yes, indicates to the reader to return all the schemas of the tables in the database.

The syntax of the **RETRIEVE_ALL_SCHEMAS** directive is:

```
POSTGRES_RETRIEVE_ALL_SCHEMAS Yes
```

RETRIEVE_ALL_TABLE_NAMES

Required/Optional: *Optional*

This specification is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

Similar to **RETRIEVE_ALL_SCHEMAS**: this optional specification is used to tell the reader to only retrieve the table names of all the tables in the source database. If **RETRIEVE_ALL_SCHEMAS** is also set to Yes, then **RETRIEVE_ALL_SCHEMAS** takes precedence. If this value is not specified, then it is assumed to be No.

The syntax of the **RETRIEVE_ALL_TABLE_NAMES** directive is:

```
POSTGRES_RETRIEVE_ALL_TABLE_NAMES Yes
```

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

*** Workbench Parameter**

Additional Attributes to Expose

Writer Overview

The PostgreSQL writer module stores both geometry and attributes into an PostgreSQL database. Note that attributes are always written as lowercase. The PostgreSQL writer provides the following capabilities:

- **Transaction Support:** The PostgreSQL writer provides transaction support that eases the data loading process. Occasionally, a data load operation terminates prematurely due to data difficulties. The transaction support provides a mechanism for reloading corrected data without data loss or duplication.
- **Index Creation:** The PostgreSQL writer can set up and populate indexes as part of the loading process. By default, no indexes are created. Additional columns can be individually indexed. Composite column indexes are not supported at this time.
- **Bulk Loading:** The PostgreSQL writer uses a bulk loading technique to ensure speedy data load.

Writer Directives

The directives that are processed by the PostgreSQL writer are listed below. The suffixes shown are prefixed by the current `<writerkeyword>_` in a mapping file. By default, the `<writerkeyword>` for the PostgreSQL writer is `POSTGRES_OUT`.

DATASET/DATABASE, HOST, PORT, USER_NAME, PASSWORD

These directives operate in the same manner as they do for the PostgreSQL reader.

DEF

Each PostgreSQL table must be defined before it can be written. The general form of a PostgreSQL definition statement is:

```
POSTGRES_DEF <tableName> \  
  [postgres_type <postgis_type>] \  
  [postgres_writer_mode (insert|update|delete|inherit_from_writer)] \  
  [postgres_drop_table (yes|no)] \  
  [postgres_truncate_table (yes|no)] \  
  [postgres_create_with_oids (yes|no)] \  
  [postgres_vacuum_analyze (yes|no)] \  
 [<fieldName> <fieldType>][,<indexType>]*
```

The table definition allows control of the table that will be created. If the table already exists, the majority of the `postgis_` parameters will be ignored and need not be given. If the fields and types are listed, they must match those in the database.

If the table does not exist, then the field names and types are used to first create the table. In any case, if a `<fieldType>` is given, it may be any field type supported by the target database.

Required/Optional

Required

Configuration Parameters

The configuration parameters present on the definition line are described in the following table:

Parameter	Contents
tableName	The name of the table to be written. If a table with the specified name exists, it will be overwritten if either the postgres_overwrite_table DEF line parameter set to YES or if the global writer keyword type postgres_out_overwrite is set to YES. Otherwise the table will be appended to. Valid values for table names include any character string devoid of SQL offensive characters and less than 32 characters in length.
postgres_type	The type of geometric entity stored within the feature. The valid values for the type are listed below: postgres_none
postgres_mode	The the default operation mode of the feature type in terms of the types of SQL statements sent to the database. Valid values are INSERT, UPDATE, DELETE and INHERIT_FROM_WRITER. Note that INSERT mode allows for only INSERT operations whereas UPDATES and DELETE can be overwritten at the feature levels. INHERIT_FROM_WRITER simply indicates to take this value from the writer level and not to override it at the feature type level. Default: INHERIT_FROM_WRITER
postgres_drop_table	This specifies that if the table exists by this name, it should be dropped and recreated before any features are written to it. This parameter, along with postgres_truncate_table, deprecates the older postgres_overwrite_table parameter. Default: NO
postgres_truncate_table	This specifies that if the table exists by this name, it should be truncated before any features are written to it. This parameter, along with postgres_drop_table, deprecates the older postgres_overwrite_table parameter. Default: NO
postgres_create_with_oids	Create the table including a system OID column as a unique identifier. If no, then the OID column is not created. Default: yes
postgres_vacuum_analyze	Perform the database function to vacuum and analyze the table once successfully written. This will build statistics for the table. Default: yes
fieldName	The name of the field to be written. Valid values for field

Parameter	Contents
	name include any character string devoid of SQL offensive characters and less than 32 characters in length.
fieldType	<p>The type of a column in a table. The valid values for the field type are listed below:</p> <ul style="list-style-type: none"> bool char(width) bpchar(width) varchar(width) int2 int4 int8 text bytea oid serial float4 float8 money date time timetz timestamp timestampz
indexType	<p>The type of index to create on the given field. The valid values for the index type are listed below:</p> <ul style="list-style-type: none"> BTREE (default attribute index) RTREE HASH PRIKEY (primary key)

START_TRANSACTION

Required/Optional: *Optional*

This statement tells the PostgreSQL writer module when to start actually writing features into the database. The PostgreSQL writer does not write any features until the feature number of features are skipped, and then it begins writing the following features. Normally, the value specified is zero – a non-zero value is only specified when a data load operation is being resumed after failing partway through.

POSTGRES_OUT_START_TRANSACTION 0

Workbench Parameter: *Starting Feature*

TRANSACTION_INTERVAL

Required/Optional: *Optional*

This statement informs the FME about the number of features to be placed in each transaction before a transaction is committed to the database.

If the `POSTGRES_OUT_TRANSACTION_INTERVAL` statement is not specified, then a value of 1000 is used as the transaction interval.

```
POSTGRES_OUT_TRANSACTION_INTERVAL 2000
```

Workbench Parameter: *Features Per Transaction*

BULK_COPY

Required/Optional: *Optional*

This statement tells the PostgreSQL writer module to insert data into the database using either SQL `INSERT` statements or the SQL `COPY` command. The default option is the bulk copy using the `COPY` command, which yields the best performance. The bulk delimiter is no longer user-adjustable – the escaping of it has improved and is no longer necessary as a backup measure. However, if individual inserts are desired, this option can be set to `NO`.

```
POSTGRES_OUT_BULK_COPY YES
```

Workbench Parameter: *Bulk COPY Insert*

WRITER_MODE

Required/Optional: *Optional*

Note: For more information on this directive, see the chapter *Database Writer Mode*.

This directive informs the Postgres writer which SQL operations will be performed by default by this writer. This operation can be set to `INSERT`, `UPDATE` or `DELETE`. The default writer-level value for this operation can be overwritten at the feature type or table level. The corresponding feature type `DEF` parameter name is called `POSTGRES_MODE`. It has the same valid options as the writer-level mode, as well as the value `INHERIT_FROM_WRITER` (which causes the writer level mode to be inherited by the feature type as the default for features contained in that table).

The operation can be set specifically for individual feature as well. Note that when the writer mode is set to `INSERT` this prevents the mode from being interpreted off individual features and all features are inserted unless otherwise marked as update or delete features. These are skipped.

If the `POSTGRES_WRITER_MODE` statement is not specified, then a value of `INSERT` is given.

```
POSTGRES_OUT_WRITER_MODE INSERT
```

Workbench Parameter: *Writer Mode*

GENERIC_GEOMETRY

Required/Optional: *Optional*

This directive is unique in that it only applies at generation time and not at translation time. The default value of `NO` indicates that we want the previous behavior of creating geometrically constrained geometry columns on the destination tables. For example, a `POINT` geometry table would be restricted only to points. Now we have the option to create generic or non-constrained geometry column types.

Effectively this means you can insert multiple geometry types into one table. Specifically the geometry column is created to have the generic type `GEOMETRY` and there are no constraints placed on the geometry types allowed.

If the `POSTGIS_OUT_GENERIC_GEOMETRY` statement is not specified, then a value of `NO` is given.

```
POSTGIS_OUT_GENERIC_GEOMETRY YES
```

Feature Representation

Features read from PostgreSQL consist of a series of attribute values only and no geometry. The feature type of each feature is as defined on its **DEF** line but the only type used is `postgis_none`. Underlying PostgreSQL geometries are not read as geometries but are interpreted as strings.

Features written to the database have the destination table as their feature type, and attributes as defined on the **DEF** line.

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Attribute Name	Contents
<code>postgres_type</code>	The type of geometric entity stored within the feature. The valid values for the object model are listed below: <code>postgres_none</code>

Features read from, or written to, PostgreSQL have an attribute for each column in the database table. The feature attribute name will be the same as the source or destination column name. The attribute and column names are case-sensitive.

No Coordinates

postgres_type: `postgres_none`

Features with no coordinates are tagged with this value when reading or writing to or from PostgreSQL. This is the default.

Troubleshooting

Problems sometimes arise when attempting to connect to a PostgreSQL database. This is almost always due to a mis-configuration in the user's environment.

The following suggestions can often help detect and overcome such problems.

- Ensure you can connect to the database with the host, port, database, user name, and password using `psql`. See PostgreSQL documentation for proper security and connection information, and for the usage of the `psql` utility.
- In most cases, the `POSTGRES_DATABASE` keyword should be left with blank values, with the `POSTGRES_DATASET` keyword containing the name of the PostgreSQL database.
- When using a UNIX operating system, the environment variables `PGHOST`, `PGPORT`, `PGDATABASE`, `PGUSER` and `PGPASSWORD` can be used to specify the PostgreSQL connection parameters.
- If the table list in the PostgreSQL reader input settings box does not display your table, try typing the name with the schema prefix, i.e. `public.mytable`. If this works, then your search path for schemas may not be set to the desired values.
- If your data ends up looking garbled using a given encoding, it may be because the encoding of the data does not match your system encoding. These must match because FME uses the system encoding to set the encoding of the PostgreSQL client, and then allows the database to convert encodings if necessary between the client and server.

Connecting to PostgreSQL/PostGIS tables in another user's schema

FME uses the `Postgres` search path to determine which schemas' tables to show in the table list. To set a user's search path for a session:

```
SET search_path TO "$user",public;  
SET search_path TO "$user",public,schema2,schema3;
```

To set a user's search path for all future sessions:

```
ALTER USER <username> SET search_path TO "$user",public;
```

```
ALTER USER <username> SET search_path TO "$user",public,schema2,schema3;
```

To see the current search path:

```
show search_path;
```

Regional Geographic Information System (REGIS) Reader/Writer

The REGIS Reader and Writer provides FME with access to the REGIS file format.

Overview

The following files are associated with the REGIS Reader and Writer. The topology file, however, is supported only by the Reader.

The following extensions are added to the basename of the REGIS files.

File Name Extension	Contents
.fea	The feature file contains the Geometric data.
.top	This file contains the topological information of the feature file. This is supported only by the Reader.

The REGIS Reader and Writer support the storage of point, line, text and polygon in its data files. Additional user-defined attributes are not supported.

FME considers a REGIS dataset to be a collection of REGIS files in a single directory.

REGIS Quick Facts

Format Type Identifier	REGIS
Reader/Writer	Both
Licensing Level	Professional
Dependencies	None
Dataset Type	Directory or File
Feature Type	File base name
Typical File Extensions	.fea
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	regis_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	yes
line	yes		z values	yes
none	no			

Reader Overview

The REGIS Reader first scans the directory it is given for the requested REGIS files. For each feature file found, the Reader checks to see if the corresponding topology file exists. If it does not exist, a warning message is given. If it does exist, all the features in the file are processed. The Reader extracts features from the file one at a time, and passes them on to the rest of the FME for further processing. When the file is exhausted, the REGIS Reader moves on to the next file in the directory. Optionally a single REGIS file can be specified. If this is the case, only that REGIS file is read.

Reader Directives

The directives processed by the REGIS reader are listed below. The suffixes shown are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the REGIS reader is **REGIS**.

DATASET

Required/Optional: *Required*

The value for this keyword is the directory containing the REGIS files to be read, or the name of a single REGIS file to be read. A typical mapping file fragment specifying an input REGIS dataset looks like:

```
REGIS_DATASET /usr/data/REGIS
```

Workbench Parameter: *Source Regional Geographic Information System (REGIS) File(s)*

DEF

Required/Optional: *Required*

Each REGIS file must be defined before it is read. The definition specifies only the base name of the file, the type of geometry it contains, and the names and the types of all attributes. The syntax of a REGIS **DEF** line is:

```
<ReaderKeyword>_DEF <baseName> \
  [<attrName> <attrType>]+
```

Field Type	Description
char(<width>)	Character fields store fixed-length strings. The width parameter controls the maximum characters that can be stored by the field. When a character field is written, it is right-padded with blanks, or truncated, to fit the width. When a character field is retrieved, any padding blank characters are stripped

Field Type	Description
	away.
date	Date fields store dates as character strings with the format YYYYMMDD .
logical	Logical fields store TRUE/FALSE data. Data read to or written from such fields must always have a value of either true or false.
number(<width>,<decimals>)	Number fields store single and double precision floating point values. The width parameter is the total number of characters allocated to the field, including the decimal point. The decimals parameter controls the precision of the data and is the number of digits to the right of the decimal.

The following mapping file fragment defines a sample REGIS line file.

```
REGIS_DEF landcover regis_type regis_line
```

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available REGIS files read. If no **IDs** are specified, then all defined and available REGIS files are read. The syntax of the **IDs** keyword is:

```
<ReaderKeyword>_IDS <baseName1> \
                    <baseName2> ... \
                    <baseNameN>
```

The base names correspond to those used in the **DEF** lines. The example below selects only the **roads** REGIS file for input during a translation:

```
REGIS_IDS roads
```

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the **mitab.dll** in the FME home directory to **mapinfo.dll**.

The syntax of the **MAPINFO_SEARCH_ENVELOPE** directive is:

```
MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

```
MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

*** Workbench Parameter**

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

*** Workbench Parameter**

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

*** Workbench Parameter**

Additional Attributes to Expose

Writer Overview

The REGIS Writer creates and writes features to REGIS files in the directory specified by the **DATASET** keyword. As with the Reader, the directory must exist before the translation occurs. Any existing REGIS files in the directory are overwritten by the new data. As features are routed to the Writer, it determines the file into which the features are written based on the feature type of the feature. Many REGIS files can be written during a single FME session.

Topology files are not supported by the Writer.

Writer Directives

The directives that are processed by the REGIS writer are listed below. The suffixes shown are prefixed by the current `<WriterKeyword>_` in a mapping file. By default, the `<WriterKeyword>` for the REGIS writer is **REGIS**.

DATASET, DEF

These directives are processed as described in the Reader Directives section.

Feature Representation

REGIS features consist of geometry and predefined attributes only. It does not make use of any user-defined attribution. All REGIS features contain a `regis_type` attribute, which identifies its geometric type. In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), REGIS features also contain additional attributes specific to the geometric type. These are described in subsequent sections.

Attribute Name	Contents
regis_type	The type of the geometry supported by REGIS file. This attribute will contain one of: <code>regis_point</code> <code>regis_line</code> <code>regis_polygon</code> <code>regis_text</code> Default: No default

Point

regis_type: regis_point

This is the point feature. There are no additional attributes for this geometric type.

Line

regis_type: regis_line

This specifies a line feature. At least two distinct coordinates have to be specified in order for FME to process the feature correctly. There are no additional attributes for this geometric type.

Polygon

regis_type: regis_polygon

This is a polygon feature. In this version of the REGIS Reader and Writer, only simple polygons are supported (donut features are not supported). REGIS polygons consist of at least four coordinates, where the first and last coordinates are exactly the same. There are no additional attributes for this geometric type.

Text

regis_type: regis_text

This is a text feature. Exactly one coordinate must be specified for this text feature.

The following table lists the special FME attribute names used to control the REGIS text settings.

Attribute Name	Contents
regis_rotation	Degrees by which the text string is rotated. Range: 0...360 Default: 0
regis_text_string	The text string of the text feature. Range: Maximum 176 characters Default: Blank
regis_text_size	The size of the text string. Range: 32-bit floating point number Default: 10

S-57 (ENC) Hydrographic Data Reader

Format Notes

- This format is not supported by FME Base Edition.
- This chapter also contains information applicable to the Additional Military Layers (AML) reader.

Overview

The S-57 Reader module provides the Feature Manipulation Engine (FME) with access to data in International Hydrographic Organization (IHO) S-57 formatted file sets. While any S-57 dataset should be supported, this reader has only been fully tested with S-57 Electronic Navigational Chart (ENC) profile products. Note that S-57 Editions 3.0 and 3.1 are supported.

The S-57 format is a standard published by IHO and more information can be found at:

<http://www.iho.shom.fr/>

An online, browsable web interface to the S-57 Object and the Attribute Catalog can be found on the Universal Systems website at:

<http://www.universal.ca/S-57/frames/S57catalog.htm>

Some aspects of an S-57 transfer, such as data quality information, is not accessible via the S-57 reader, however, a user well-versed in the S-57 format can extract it using the ISO8211 reader.

S-57 Quick Facts

Format Type Identifier	S57
Reader/Writer	Reader
Licensing Level	Professional
Dependencies	None
Dataset Type	File/Catalog
Feature Type	Object class
Typical File Extensions	.000, 030
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	Yes
Generic Color Support	Yes
Spatial Index	Never
Schema Required	Not applicable
Transaction Support	No
Geometry Type	s57_type
Encoding Support	Yes

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	yes		z values	no
none	yes			

Reader Overview

The S-57 Reader module produces FME features for all S-57 features in one or more related S-57 data files. An S-57 dataset can be an S-57 catalog file in which case, all files referred to from the catalog are selected, or an individual S-57 data file.

S-57 feature objects are translated into FME features. S-57 geometry objects are automatically collected and formed into geometries on the features. Geometry objects are not separately accessible with the S-57 reader.

The FME S-57 reader supports S-57 update files. S-57 update files contain information on how to update a distributed S-57 data file. Update files have a file extension of 001, 002, etc.

Reader Directives

The directives processed by the S-57 reader are listed below. The suffixes shown are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the S-57 reader is **s57**.

DATASET

Required/Optional: *Required*

The dataset may be specified as an S-57 data file or S-57 catalog file. If a single data file is selected, only that file will be in the dataset. If an S-57 catalog file – normally called **CATALOG.030** – is selected, all S-57 data files listed in it will be selected.

For example:

S57_DATASET NEWFILES\I

or

S57_DATASET NEWFILES\I\CA39995I.000

or

S57_DATASET NEWFILES\I\CATALOG.030

Workbench Parameter: *Source S-57 (ENC) Hydrographic Data File(s)*

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined S-57 files read. If no IDs are specified, then all available S-57 files in the dataset are read. The syntax of the **IDs** keyword is:

```
<ReaderKeyword>_IDS <baseName1> \  
  <baseName2> ... \  
  <baseNameN>
```

The example below selects only the **CA39995I.000** file for input during a translation:

```
S57_IDS CA39995I
```

Workbench Parameter: *Feature Types to Read*

FORCE_GENERIC

Required/Optional: *Optional*

The FME is programmed to recognize all feature object classes defined as part of the S-57 standard and to provide a feature schema for each object class with the set of attributes defined in the standard. However, in some cases it may be convenient to discard object class specific attributes and group all features in a small set of feature types based on the geometry type, rather than the S-57 object class.

The **FORCE_GENERIC** keyword can be used to force all features to be treated as one of the feature types **Point**, **Line**, **Area** or **Meta** depending on their geometry. In this case, object-class-specific attributes are discarded, but the attributes **GRUP**, **OBJL**, **RVER**, **AGEN**, **FIDN**, **DSNM**, **LNAM**, and **LNAM_REFS** common to all features are still generated.

For example:

```
S57_FORCE_GENERIC ON
```

If the \$FME_HOME/s57/*.csv files used to define the S-57 object classes cannot be found at run-time, the **FORCE_GENERIC** flag will automatically be turned on and an appropriate warning will be generated in the reader's log output.

Workbench Parameter: *<WorkbenchParameter>*

UPDATES

Required/Optional: *Optional*

The S-57 reader will by default apply all updates available for the datasets read. That is, if there are files ending in **.001**, **.002** and so on, in the same directory with base datasets (ending in **.000**), these update files will be read and applied to the base feature set in accordance with S-57 update rules. The **UPDATES** directive in the mapping file may be set to **IGNORE** to ignore all updates. The default value is **APPLY** indicating that updates should be applied.

Workbench Parameter: *Action to take on update files*

FULL_STRUCTURE

Required/Optional: *Optional*

This keyword allows primitives to be read as individual features whereby each feature has some extra information which could be used in future for writing to a S-57 dataset (not available yet). By default, this keyword is **OFF**.

For example:

```
S57_FULL_STRUCTURE ON
```

Workbench Parameter: *<WorkbenchParameter>*

PROFILE

Required/Optional: *Optional*

This specifies which enhanced version of S-57 dataset to read. This is used only during schema generation (mapping file or workspace generation) and has no effect during normal reading. The original specifications for S-57 could be modified by adding additional object classes or adding more attributes therefore marking it as a different flavor of

original S-57. By setting this keyword to either **Default**, **Additional_Military_Layers** or **Inland_Waterways** the reader can then process the schema accordingly. By default, this keyword is set to **Default** which means the dataset is interpreted as the original S-57.

For example:

```
Generate S57 NULL "<source dataset>" "<mapping file name>" ----  
Source_PROFILE_IN "Inland_Waterways"
```

Note: PROFILE_IN is the macro to use to set the value for the keyword PROFILE.

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the `mitab.dll` in the FME home directory to `mapinfo.dll`.

The syntax of the **MAPINFO_SEARCH_ENVELOPE** directive is:

```
MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

```
MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The **COORDINATE_SYSTEM** directive, which specifies the coordinate system associated with the data to be read, must always be set if the **SEARCH_ENVELOPE_COORDINATE_SYSTEM** directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the **SEARCH_ENVELOPE_COORDINATE_SYSTEM** to the reader **COORDINATE_SYSTEM** prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the **SEARCH_ENVELOPE** directive.

Values

YES | NO (default)

Mapping File Syntax

<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Feature Representation

Normally, all features read from S-57 are assigned a feature type based on the name of the object class (OBJL) to which they belong. For instance, with an **OBJL** value of **2**, the feature is an Airport/airfield and has a short name of AIRARE which is used as the FME feature type. A typical S-57 transfer may have in excess of 100 feature types.

Each feature type has a predefined set of attributes as defined by the S-57 standard. For instance, the airport (AIRARE) object class can have the AIRARE, CATAIR, CONDTN, CONVIS, NOBJNM, OBJNAM, STATUS, INFORM, NIN-FOM, NTXTDS, PICREP, SCAMAX, SCAMIN, TXTDSC, RECDAT, RECIND, SORDAT, and SORIND attributes. These short names can be related to longer, more meaningful names using an S-57 object/attribute catalog, such as the S-57 standard document itself or the files in the **fme/s57** directory. Such a catalog can also be used to establish all available object classes and their attributes.

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*) the following common attributes are also added – these include generic attributes that appear on all features, regardless of whether object class is turned on.

Attribute Name	Description	Defined On
s57_type	Geometry type of this feature. One of the S57_point, s57_point3d, s57_line, s57_polygon or s57_no_geom files. Note that this relates closely to the S-57 PRIM field.	All features
s57_update_file{}	List of file names that have been used to update the current feature. These list entries are paired with the s57_update_type field so that together they specify a list update operations from a list of files. For example, s57_update_file{0} =	Updated features

Attribute Name	Description	Defined On
	mys57file.001 s57_update_type{0} = INSERT	
s57_update_type{}	The list of update types that have been performed on the feature. Possible values are INSERT, MODIFY or DELETE. These list entries are paired with the s57_update_type field so that together they specify a list update operations from a list of files. For example, s57_update_file{0} = mys57file.001 s57_update_type{0} = MODIFY s57_update_file{1} = mys57file.002 s57_update_type{1} = MODIFY	Updated features
COLOUR	FME will interpret the color value and set the fme_color attribute. COLOUR is a list type attribute (i.e. the value may be "2,6,2"). FME will select the first color value in the list as the default color for the feature.	Some features
GRUP	Group number	All features
OBJL	Object label code This number indicates the object class of the features.	All features
RVER	Record Version	All features
AGEN	Numeric agency code, such as 50 for the Canadian Hydrographic Service. A potentially outdated list is available in \$FMEHOME/s57/agencode.txt .	All features
FIDN	Feature identification number	All features
FIDS	Feature identification subdivision	All features
LNAM	Long name. An encoding of AGEN , FIDN , and FIDS used to uniquely identify these features within an S-57 file.	All features
LNAM_REFS{}	List of LNAM values of other features related to this feature.	Some features
DSNM	Dataset name. The file name where the feature came from. Used with LNAM to form a unique dataset wide identifier for a feature.	All features
INFORM	Informational text	Some features
NINFOM	Informational text in national language	Some features
OBJNAM	Object name	Some features

Attribute Name	Description	Defined On
NOBJNM	Object name in national language	Some features
SCAMAX	Maximum scale for display	Some features
SCAMIN	Minimum scale for display	Some features
SORDAT	Source date	Some features

The S-57 reader also depends on CSV text files with definitions of S-57 object classes, and their attributes. These are located in the files `s57attributes.csv`, `s57objectclasses.csv`, and `s57expectedinput.csv`. These CSV files are installed in `$FME_HOME/s57`. If, for some reason, they aren't found, the reader will default to reading all objects using the FORCE_GENERIC ON schema.

The S-57 ENC format supports "list" attributes. FME represents list attributes as a comma-separated list for the attribute value. For example, COLOUR is a list type attribute and may have a value "2,6,2".

Soundings

Depth soundings are handled somewhat specially in the S-57 format to efficiently represent the many available data points. In S-57, one sounding feature can have many sounding points. The FME S-57 reader splits each of these out into its own feature type, `SOUNDG` feature, with an `s57_type` of `s57_point3d`. All soundings from a single feature record have the same `AGEN`, `FIDN`, `FIDS`, and `LNAM` values.

Feature Relationships (LNAM)

The S-57 format has a concept of features being related to one another by way of the `LNAM` subfield of the `FFPT` (Feature to Feature Object Pointer) field. These relationships are encoded in the `LNAM_REFS{}` list attribute of FME features when such relationships exist.

In the S-57 format, these relationships are marked as being *master*, *slave*, or *peer-to-peer*. In practice, though, the only values that exist are *master-to-slave* pointers, so the explicit relationship is not preserved.

Each feature is also tagged with an `LNAM` value, which is the unique identifier for the feature within a single file. The FME `ReferenceFactory` can be used to associate the geometry of slave features with their master as shown in this example.

```
#=====
# Collect geometries for C_AGGR objects.

FACTORY_DEF * TeeFactory \
FACTORY_NAME AggrGeomDuplicate \
INPUT FEATURE_TYPE * s57_type s57_point \
INPUT FEATURE_TYPE * s57_type s57_line \
INPUT FEATURE_TYPE * s57_type s57_polygon \
OUTPUT FEATURE_TYPE * \
OUTPUT FEATURE_TYPE GeomSource \
@KeepAttributes(LNAM,DSNM)

FACTORY_DEF * ReferenceFactory \
FACTORY_NAME AggrCollector \
INPUT REFERENCEE FEATURE_TYPE GeomSource \
INPUT REFERENCER FEATURE_TYPE C_AGGR \
REFERENCEE_FIELDS LNAM \
REFERENCER_FIELDS LNAM_REFS{ } \
REFERENCE_INFO GEOMETRY \
GROUP_BY DSNM \
AGGREGATE_ONLY \
OUTPUT COMPLETE FEATURE_TYPE * \
OUTPUT INCOMPLETE FEATURE_TYPE *
```

Scalable Vector Graphics (SVG) Writer

The Scalable Vector Graphics (SVG) Writer enables FME to write documents that conform to the World Wide Web Consortium's (W3C) SVG 1.1 specification. This chapter assumes familiarity with the specification.

Overview

FME's SVG output is optimal for scripting and spatial information display. Specific features include:

- coordinate preservation;
- layered spatial geometry with adjustable paint order;
- template processing for incorporation of predefined scripts, style sheets, images and other SVG entities;
- single SVG element output for each FME feature (including features with donut and aggregate geometry)
- international character support.
- gzip compression support.

SVG Quick Facts

Format Type Identifier	SVG
Reader/Writer	Writer
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	group ID attribute name
Typical File Extensions	.svg, .svgz
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	Yes
Spatial Index	Not applicable
Schema Required	Optional
Transaction Support	No
Geometry Type	svg_type
Encoding Support	No

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	yes	point	yes
circles	yes	polygon	yes
circular arc	yes	raster	no
donut polygon	yes	solid	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
elliptical arc	yes		surface	no
ellipses	yes		text	yes
line	yes		z values	no
none	yes			

Writer Overview

The SVG writer converts a set of FME features into geometric SVG elements. These elements are output in a document with three sections: the template section, the layer section and the geometric section.

- The template section is composed of a single non-extended SVG document.
- The layer section is embedded in the template section, and is composed of zero to more SVG group elements
- The geometric section is embedded in the layer section, and is composed of zero to more geometric SVG elements.

In the layer section, one SVG group element is produced for each unique feature type in a writer's feature set. The **id** attribute on this group element is set equal to the feature type name. The layer is inserted immediately before the template document's closing root element tag.

The document's geometric section contains a single SVG element for each FME feature that is sent to the writer. These elements are grouped according to their feature type and embedded under the group element with a matching ID in the layer section.

The SVG writer does not check FME attribute name characters that are invalid XML attribute name characters. FME attribute names are transcoded from the operating system's local code page to UTF-8 and written directly to the SVG document; the writer does not escape any attribute name characters into character entities. It is the responsibility of the user to ensure the FME attribute names are valid XML attribute names.

Writer Directives

The directives listed below are processed by the SVG writer. The suffixes shown are prefixed by the current **<writerKeyword>** in a mapping file. By default, the **<writerKeyword>** for the SVG writer is **SVG**.

DATASET

Required/Optional: *Required*

The value for this keyword is the pathname for the output SVG file. If a file with this pathname already exists, then it will be overwritten. A typical mapping file fragment specifying an output SVG dataset looks like:

```
SVG_DATASET /tmp/outputFile.svg
```

If the output filename's extension is **svgz** then the output document will be compressed using gzip compression.

Note that all SVG documents are written using UTF-8 encoding.

Workbench Parameter: *Destination SVG File*

TEMPLATE

Required/Optional: *Optional*

This optional parameter directs the writer to the location of the SVG document to use as the outline of the output dataset.

The syntax for this keyword's value is:

```
<writerKeyword>_TEMPLATE <value>
```

(where **<value>** is location of the template path)

If this keyword is not provided in the mapping file, then the file named `defaultTemplate.svg` under the `svg` directory in the FME home directory is used.

The template document has several uses including: the insertion of predefined geometric elements, the inclusion of Cascading Style Sheets, and the embedding of scripting information. There are a few issues that must be considered to ensure proper template processing. The template must conform to the non-extended SVG language defined by the SVG 1.1 specification. The encoding of the template must be one of the following: ASCII, UTF-8, UTF-16, UCS4, ISO-8859-1 or Windows-1252. Note that the encoding of the output SVG document is always UTF-8. Any document type declaration provided in the template will be overridden in the output document.

Two placeholder macros have been defined for use in the SVG template in order to retrieve information specified in other keywords:

- `$(FME_SVG_ATTR_NS_PREFIX)` will be replaced with the value of the `ATTR_NAMESPACE_PREFIX` keyword, and
- `$(FME_SVG_ATTR_NS_URI)` will be replaced with the value of the `ATTR_NAMESPACE_URI` keyword.

These macros will only work inside CDATA sections (`<![CDATA[...]]>`) of the SVG template. If they are found outside a CDATA section, they will remain unchanged.

Workbench Parameter: *Template File*

COORDINATE PRECISION

Required/Optional: *Optional*

This optional parameter specifies the number of decimal digits to use when writing an SVG element coordinate's value. The default is 6. Specifying a larger value increases coordinate precision and may increase rendering precision.

Workbench Parameter: *Precision*

NORMALIZE

Required/Optional: *Optional*

This optional parameter will normalize the lower coordinate bounds of the writer's feature set to (0,0). Normalization can reduce rendering inaccuracies by SVG viewers with small coordinate precision capability. A normalized document's file size is typically smaller than a non-normalized version.

Workbench Parameter: *Normalize*

DEF

Required/Optional: *Optional*

The syntax for **DEF** is:

```
<WriterKeyword>_DEF <FeatureType>
  SVG_PAINT_ORDER [0-9]+
  SVG_LAYER_STYLE string
  <UserAttributeName0> char([0-9]+)
  ...
  <UserAttributeNameN> char([0-9]+)
```

The `SVG_PAINT_ORDER` parameter on a **DEF** line is used to determine the order of feature output. Features in layers that have a higher value for this parameter will be output last. Following SVG's "painter" algorithm, features that are in layers with higher values will be painted last when the SVG document is rendered.

The `SVG_LAYER_STYLE` parameter on a **DEF** line is used to specify the value to set the layer's **STYLE** attribute in the output layer group.

The user attribute keywords specify which FME attributes to extract from an incoming FME feature. The extracted FME attributes are embedded in the geometric element's attribute list.

If there are no user attribute **DEF** line parameters specified, then no FME user attributes will be inserted in any SVG element's attribute list, and no SVG DTD extension is produced.

ABSOLUTE_COORDINATES

Required/Optional: *Optional*

Allows absolute instead of relative coordinates to be used for lines and polygons that are written out as `<path>` elements. The valid values for this keyword are **Yes** and **No**; its default value is **No**.

Workbench Parameter: *Use absolute Coordinate*

WHITE_STROKES_TO_BLACK

Required/Optional: *Optional*

Determines whether the SVG writer should automatically switch white `fme_color` specifications into black. This directive does not affect the `svg_color` (that is, the `svg_color` attribute takes precedence over the `fme_color`). The valid values for this keyword are **Yes** and **No**; its default value is **Yes**.

Workbench Parameter: *Automatically turn white strokes into black*

DOCTYPE_EXTERNAL

Required/Optional: *Optional*

Determines if the SVG file depends on an external SVG DTD. The valid values for this keyword are **Yes** (default value) and **No**. When set to **Yes** the document type declaration's public and system identifier for SVG 1.1 are used by default, but these default identifiers can also be overwritten with the `DOCTYPE_PUBLIC_ID` and `DOCTYPE_SYSTEM_ID` keywords.

Workbench Parameter: *Reference external SVG DTD*

DOCTYPE_PUBLIC_ID

Required/Optional: *Optional*

This keyword only applies when the `DOCTYPE_EXTERNAL` keyword is set to **Yes**. It specifies the public identifier for the document type declaration. This keyword must be used in conjunction with the `DOCTYPE_SYSTEM_ID` keyword, that is, a system identifier must also be simultaneously specified; otherwise, this keyword has no effect.

Workbench Parameter: *DOCTYPE public identifier*

DOCTYPE_SYSTEM_ID

Required/Optional: *Optional*

This keyword only applies when the `DOCTYPE_EXTERNAL` keyword is set to **Yes**. It specifies the system identifier for the document type declaration. This keyword can be used alone or in conjunction with the `DOCTYPE_SYSTEM_ID` keyword.

Workbench Parameter: *DOCTYPE system identifier*

ATTR_NAMESPACE_PREFIX

Required/Optional: *Optional*

This directive specifies the prefix which will be used to identify the namespace of all user attributes in the SVG document.

Each user attribute written to the SVG file will be written as an XML attribute with the format `namespace_prefix:user_attr = "value"`.

The default namespace prefix is "fme".

Workbench Parameter: *Attributes Namespace Prefix*

ATTR_NAMESPACE_URI

Required/Optional: *Optional*

This directive specifies the URI with which the namespace prefix (specified by `ATTR_NAMESPACE_PREFIX`) will be associated. This will be the namespace URI for all user attributes in the SVG document. (Note that the writer does not check if this is a valid URI that complies with XML standards.)

The namespace will be defined as follows, where `namespace_prefix` is the value defined by `ATTR_NAMESPACE_PREFIX` and `namespace_uri` is defined by `xmlns:namespace_prefix="namespace_uri"`

The default value is `"http://www.safe.com/fme"`

Workbench Parameter: *Attributes Namespace URI*

VIEWBOX_MINX, VIEWBOX_MINY, VIEWBOX_WIDTH, VIEWBOX_HEIGHT

Required/Optional: *Optional*

These directives allow the user to set viewBox size when writing to SVG format. By default, the viewBox is not available. Only when the user has set all four parameters (listed below), the viewBox will be displayed.

Syntax for the values is:

```
<WriterKeyword>_VIEWBOX_MINX<value>  
<WriterKeyword>_VIEWBOX_MINY<value>  
<WriterKeyword>_VIEWBOX_WIDTH<value>  
<WriterKeyword>_VIEWBOX_HEIGHT<value>
```

All values must be specified in decimal, integer or scientific notation.

Workbench Parameter: *Viewbox - Min x, Viewbox - Min y, Viewbox - Width, Viewbox - Height*

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Any feature that is sent to the SVG writer has several attributes that the writer uses to determine that feature's geometric representation in SVG. The start of this processing occurs when the writer examines the feature's `svg_type` attribute. Once the writer determines this attribute's value, it can process the other attribute information required to complete the geometric conversion process. If, for example, the feature has an `svg_type` of `svg_arc`, then the writer will retrieve the `svg_primary_axis`, `svg_secondary_axis`, `svg_start_angle` and `svg_rotation` attributes to determine what geometric attribute values will exist in the feature's path element representation.

All format-specific attributes begin with `svg_` and are predefined in this section's tables. In addition to format attributes, the SVG writer can process user-defined attributes. The writer extracts these user attributes from the incoming features and inserts them to the output element's attribute list. To determine which attributes to extract, the writer examines its mapping file's `DEF` lines. The feature's feature type must match the `DEF` lines type for this extraction to occur. The user-defined attributes are defined in the SVG element's attribute list under the qualified name prefix `fme`. An extension to the SVG DTD is produced to accommodate these user-defined attributes. This extension is defined in the document's internal document type declaration.

When producing an element's attribute list, the writer will examine the contents of the attribute values to determine if there are any `<` or `"` characters. If these values are present, they are output using `<` and `"`, respectively. Attribute values are embedded in an element's attribute list using the quote (`"`) delimiter.

The following table lists the format attributes that are common to all features sent to the SVG writer. Other than `svg_color` and `svg_fill_color`, all attributes in this table have a direct mapping to the attribute names that can be set on individual SVG elements. Selected SVG elements attribute names have been prepended with a `svg_` string to produce the FME attribute names.

Attribute Name	Contents
svg_color	<p>The color used to stroke an element. The string is formatted with three comma-separated values representing the ordered intensities red, green, and blue. The individual intensity values are character decimal character strings that can range in value from 0 to 1 with 1 being the highest. See note on color below.</p> <p>Range: string 0.0..1.0, 0.0..1.0, 0.0..1.0</p> <p>Default: None</p>
svg_fill_color	<p>The color used to fill an element. The string is formatted with three comma-separated values representing the ordered intensities red, green, and blue. The individual intensity values represent decimals that can range in value from 0 to 1 with 1 being the highest. This value is not applicable to <code>svg_line</code> or <code>svg_arc</code> features. See note on color below.</p> <p>Range: string 0.0..1.0, 0.0..1.0, 0.0..1.0</p> <p>Default: None</p>
svg_id	<p>An element's unique identifier. Directly maps to an element's <code>id</code> attribute. Refer to the XML 1.0 specification for applicable values. It is strongly recommended that users not create IDs that begin with "FME_".</p> <p>Range: string</p> <p>Default: None</p>
svg_class	<p>Assigns a class name or set of class names to an element.</p> <p>Directly maps to an element's <code>class</code> attribute. Refer to the SVG 1.1 specification for applicable values.</p> <p>Range: string</p> <p>Default: None</p>
svg_style	<p>Specifies style information for an element. Directly maps to an element's <code>style</code> attribute. Refer to the SVG 1.1 specification for applicable values.</p> <p>Range: string</p> <p>Default: None</p>
svg_onfocusin	<p>Identifies the script method to call when an element</p>

Attribute Name	Contents
	<p>receives focus. Directly maps to an element's onfocus attribute.</p> <p>Range: string (must match an available script method ID)</p> <p>Default: None</p>
svg_onfocusout	<p>Identifies the script method to call when an element loses focus. Directly maps to an element's onfocusout attribute.</p> <p>Range: string (must match an available script method ID)</p> <p>Default: None</p>
svg_onclick	<p>Identifies the script method to call when a pointing device button is clicked over an element. Directly maps to an element's onclick attribute.</p> <p>Range: string (must match an available script method ID)</p> <p>Default: None</p>
svg_onmousedown	<p>Identifies the script method to call when a pointing device button is pressed over an element. Directly maps to an element's onmousedown attribute.</p> <p>Range: string (must match an available script method ID)</p> <p>Default: None</p>
svg_onmouseup	<p>Identifies the script method to call when a pointing device button is release over an element. Directly maps to an element's onmouseup attribute.</p> <p>Range: string (must match an available script method ID)</p> <p>Default: None</p>
svg_onmouseover	<p>Identifies the script method to call when a pointing device button is moved on to an element. Directly maps to an element's onmouseover attribute.</p> <p>Range: string (must match an available script method ID)</p> <p>Default: None</p>
svg_onmousemove	<p>Identifies the script method to call when a pointing device button is moved while it is over an element. Directly maps to an element's onmousemove attribute.</p> <p>Range: string (must match an available script</p>

Attribute Name	Contents
	method ID) Default: None
svg_onmouseout	Identifies the script method to call when a pointing device button is moved away from an element. Directly maps to an element's <code>onmouseout</code> attribute. Range: string (must match an available script method ID) Default: None

Note:

The attributes `fme_color` and `svg_color` can both be used to set the value on an element's `stroke` attribute. `fme_color` and `svg_color` are translated to SVG's RGB function syntax. For the case where more than one color attribute is specified on a feature, order of precedence is `svg_color`, and then `fme_color`.

The same processing occurs for the attribute `fme_fill_color`, `svg_fill_color`, except both values can be used to set the element's 'fill' attribute.

In addition, if the values for the `svg_color` or `svg_fill_color` do not match the FME color specification, i.e., "r,g,b" where r,g,b are in [0..1], then the writer will plainly transfer the value specified into the SVG stroke and fill attributes respectively. This is useful if the user needs to by pass the FME "r,g,b" syntax, for example, to use the SVG's predefined color names, "red", "black", etc..., or if the user wants to use gradient fill. A user-defined SVG template (see the writer's TEMPLATE keyword) could define several gradients to be referenced by the FME feature's `svg_fill_color` attribute.

Consider the following TEMPLATE for the SVG writer. It defines the MyGradient `linearGradient` that can be referenced by FME features by setting their `svg_fill_color` attribute to the value, `url(#MyGradient)`:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg xmlns="http://www.w3.org/2000/svg">
  <defs>
    <linearGradient id="MyGradient">
      <stop offset="5%" stop-color="#F60" />
      <stop offset="95%" stop-color="#FF6" />
    </linearGradient>
  </defs>
</svg>
```

Points

svg_type: `svg_point`

Point features can have one or more coordinates. All point features are usually output as SVG path elements. Single, non-aggregate, point features may also be output by an SVG use element that references another element whose graphical content is to be drawn at the position of the single point:

Attribute Name	Contents
svg_use	This format attribute only applies to single point features. If it is present and non-empty in a single point feature then the point feature is written out with a SVG use rather than a SVG path element. The value of the <code>svg_use</code> attribute must be a valid href location because this value is directly copied into

Attribute Name	Contents
	<p>the use element's xlink:href attribute. The single coordinate of the point is also transferred onto the use element's x and y attributes. The element referenced by the the SVG use element may be predefined in an FME SVG writer template file, see the writer's TEMPLATE keyword.</p> <p>Range: string Default: Empty String</p>

Lines

svg_type: [svg_line](#)

Polyline features must have at least two coordinates. Line features are output as SVG path elements

Polygons

svg_type: [svg_polygon](#)

Polygon features must have at least two coordinates. Polygon feature are output as SVG path elements, and are automatically closed if the first and last coordinate of a polygon segment do not match.

Text

svg_type: [svg_text](#)

Text features must have exactly one coordinate. Text features are output as SVG text elements, and have the following additional attributes:

Attribute Name	Contents
svg_text_string	<p>The text string may contain blanks and there is no limit on its length. This attribute must be present for all svg_text features.</p> <p>Range: string Default: Empty String</p>
svg_text_size	<p>The size of the text in ground units.</p> <p>Range: Any real number > 0 Default: 0</p>
svg_rotation	<p>The rotation of the text, as measured in degrees counterclockwise from the horizontal.</p> <p>Range: -360.0...360.0 Default: 0</p>

Ellipse

svg_type: [svg_ellipse](#)

Ellipse features must have exactly one coordinate. Ellipse features are output as SVG ellipse elements and have the following additional attributes:

Attribute Name	Contents
svg_primary_axis	The length of the semi-major axis in ground units. Range: Any real number > 0 Default: 0
svg_secondary_axis	The length of the semi-minor axis in ground units. Range: Any real number > 0 Default: 0
svg_rotation	The rotation of the ellipse, as measured in degrees counterclockwise from the horizontal. Range: -360.0...360.0 Default: 0

Arc

svg_type: [svg_arc](#)

Arc features must have exactly one coordinate. Arc features are output as SVG path elements, and have the following additional attributes:

Attribute Name	Contents
svg_primary_axis	The length of the semi-major axis in ground units. Range: Any real number > 0 Default: 0
svg_secondary_axis	The length of the semi-minor axis in ground units. Range: Any real number > 0 Default: 0
svg_start_angle	The start angle defines the counterclockwise distance from the primary axis to the starting point of the arc. It is measured in degrees. Range: 0.0...360.0 Default: 0
svg_rotation	The rotation of the major axis. The rotation is measured in degrees counterclockwise up from the horizon. Range: -360.0...360.0 Default: 0

Rectangle

svg_type: [svg_rectangle](#)

The extends of this feature are calculated using its bounding box. Rectangle features are output as SVG rect elements.

Rounded Rectangle

svg_type: [svg_rectangle](#)

The extents of this feature are calculated using its bounding box. Rounded Rectangle features are output as SVG rect elements.

Attribute Name	Contents
svg_rounding	Contains the diameter, in ground units, of the circle used to produce the rounded corners. Range: Any real number > 0 Default: 0

Spatial Archive and Interchange Format (SAIF) Reader/Writer

The Spatial Archive and Interchange Format (SAIF) features a powerful object-oriented data model described in an easy-to-use data definition language called Class Syntax Notation (CSN).

SAIF is the standard archive and interchange format for geographic data in the province of British Columbia. SAIF was developed to address both data interchange and data archival issues.¹ As a result, SAIF is an excellent format for storing geographic data in a vendor-neutral manner. FME enables data stored in SAIF to be easily translated to any of the popular vendor formats.

Overview

SAIF uses the latest paradigm in data modeling. It employs an object-oriented data model supporting multiple inheritance. SAIF was designed to be user-extensible allowing users to easily create new class definitions. While designed with spatial data in mind, SAIF can be used just as effectively to model any type of data.

SAIF also supports other advanced data modeling concepts not found in any of the other formats.

- **Object Referencing:** SAIF enables objects within a single dataset to reference component objects. For example, if the geometry of a linear feature defines both a river bank and a lot boundary, then SAIF enables both the river and the lot boundary to reference the same linear feature.
- **Direct Support for Multimedia Datatypes:** SAIF enables multimedia datatypes such as JPEG, Graphic Interchange Format (GIF), Sound Files, or any other type of file to be stored directly within a dataset. Attributes which describe the embedded information are also stored in the file.
- **Object Linking:** SAIF enables objects within a SAIF dataset to refer to other objects and to associate attributes with these links.

SAIF datasets have the following structure.



SAIF Dataset Structure

¹SAIF datasets are self-contained. A single SAIF dataset contains both the data and the data model which describes the data.

SAIF Directory

SAIF datasets are composed of a collection of addressable objects. Each addressable object is identified with a unique identifier stored in the SAIF directory, along with the object's class information and the object's location within the dataset.

Unlike other file-based data storage formats, SAIF uses the directory to support random retrieval of data. For example, if a SAIF dataset contains **Roads**, **Railroads**, **Rivers**, and so on, you can quickly retrieve the **Roads** objects from the dataset without having to read features of any other type. Each addressable object in SAIF is generally used to hold a collection of features of the same type. For example, one addressable object may hold all of the **Roads** while another addressable object holds the **Railroads**, and a third addressable object contains the **Rivers**. This organization of data fits well with that used by most Geographical Information Systems (GIS) products.

Tip: If a user wishes to read every feature in a SAIF dataset, then the IDs keyword can be omitted.

The <ReaderKeyword>_IDs statement within an FME mapping file is used to identify the objects to be retrieved from a SAIF dataset.

Upon opening a SAIF dataset, the SAIF reader logs the contents of the SAIF dataset to the FME log file.

SAIF Schema

The second major component of a SAIF dataset is the SAIF Schema. The SAIF Schema contains the class definitions for all objects stored within the SAIF dataset. Every SAIF feature within the dataset is defined by the data model stored in this portion of the dataset. The class definitions are specified the Class Syntax Notation (CSN). CSN is an easy to read notation, used specifically for defining classes in SAIF. See the *Spatial Archive and Interchange Format: Formal Specification Release 3.2* for a complete description of SAIF and CSN.

SAIF Object Definitions

The third, and final, component of a SAIF dataset contains the feature data. The feature data within SAIF is stored in Object Syntax Notation (OSN). OSN is used specifically for defining objects in SAIF. See the *Spatial Archive and Interchange Format: Formal Specification Release 3.2* for a complete description of SAIF and OSN.

The object definitions are broken down into smaller units called *object sets*. Each object set contains a collection of objects. For discussion purposes, it is assumed that there is a one-to-one correspondence between *addressable objects* and *object sets*, and you can use them interchangeably. The distinction between these two concepts is beyond the scope of this document.

For a more detailed description of the organizations of a SAIF dataset, see the *SAIF Toolkit API Programmer's Reference Manual Release 1.1*.

SAIF Quick Facts

Format Type Identifier	SAIF
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	File
Feature Type	Class name
Typical File Extensions	.saf, .zip
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	as per SAIF class definition
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	yes		z values	yes
none	yes			

Reader Overview

The SAIF reader module produces FME features from the features held in a SAIF dataset. The SAIF reader first opens the SAIF dataset, retrieving the coordinate system and directory information. Then it determines the objects to be read from the dataset by comparing the objects held in the dataset with those specified on the **IDs** statement of the FME mapping file. If no **IDs** are specified, then the SAIF reader module returns all objects in the SAIF dataset. The SAIF reader then extracts features from the SAIF dataset, one at a time, and passes them on to the rest of the FME. Each feature has its coordinate system tagged with the coordinate system read from the SAIF file.

Reader Directives

The directives processed by the SDL reader are listed below. The suffixes shown are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the SDL reader is **SDL**.

DATASET

Required/Optional: *Required*

The value for this keyword is the name of the SAIF dataset file. A typical mapping file fragment specifying an input SAIF dataset looks like:

```
SAIF_DATASET /usr/data/SAIF/92i080.zip
```

Workbench Parameter: *Source SAIF File(s)*

IDs

Required/Optional: *Optional*

This optional specification is used to limit which of the available and defined SAIF addressable objects are read. If there are no **IDs** specified, then all defined and available addressable objects are read.

Tip: The SAIF Utilities package can be used to list the IDs present in a SAIF dataset.

The syntax of the **IDs** keyword is:

```
<ReaderKeyword>_IDs <SAIF ID1> \  
    <SAIF ID2> ... \  
    <SAIF ID3>
```

The list of **IDs** can also be specified across multiple **<ReaderKeyword>_IDs** statements, in which case the union of all IDs statements are used.

The example below selects only the **roads** for input during a translation:

```
SAIF_IDS roads
```

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the **mitab.dll** in the FME home directory to **mapinfo.dll**.

The syntax of the **MAPINFO_SEARCH_ENVELOPE** directive is:

```
MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

```
MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The **COORDINATE_SYSTEM** directive, which specifies the coordinate system associated with the data to be read, must always be set if the **SEARCH_ENVELOPE_COORDINATE_SYSTEM** directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

Writer Overview

The SAIF writer creates and writes feature data to the SAIF archive identified by **SAIF_DATASET**. If the output SAIF dataset existed before the writer was run, then it is overwritten with the new data. The SAIF writer is able to have many different SAIF object sets open at a single time. As features are routed to the SAIF writer by the FME, it determines the object set into which the feature is destined and writes the feature out to that object set. The writer will also output the coordinate system of the features to the output SAIF dataset.

Writer Directives

The directives that are processed by the SAIF writer are listed below. The suffixes shown are prefixed by the current **<WriterKeyword>_** in a mapping file. By default, the **<WriterKeyword>** for the SAIF writer is **SAIF**.

Tip: The SAIF Utilities package can be used to check the syntax and validate the definitions held in the CSN before they are used by the FME.

DATASET

Required/Optional: *Required*

The value for this keyword is the name of the SAIF dataset file. A typical mapping file fragment specifying an output SAIF dataset looks like:

```
SAIF_DATASET /usr/data/SAIF/92i080.zip
```

DEF

Required/Optional: *Required*

Before any SAIF data can be written, the SAIF addressable object, in which the features are contained, must be specified. The syntax of the SAIF **DEF** line is:

```
SAIF_DEF <membertype> \  
    [SAIF_COMPOSITE_CLASS <composite type>] \  
    [SAIF_IDENTIFIER <identifier>] \  
    [SAIF_OBJECTSET <object set id>] \  
    [SAIF_AGGREGATE <aggregate spec>] \  
    [SAIF_COMPONENT_PATH <componentPath>] \  
    [<geoComponents spec>]* \  
    [<attr path> <attr value>]*
```

Each of the components of the **SAIF_DEF** statement are described below. The DEF line attempts to make SAIF as easy to define as possible by generating defaults that can be used in the majority of cases. See the points below for the values of these defaults and a discussion of what they mean and when to override them.

<membertype>: This is the feature type of the objects stored within the SAIF object being defined. In SAIF it is strongly suggested that all class names consist of two parts; a <class name>, and <domain name> separated by a double colon (::).

<class name>: A unique class name within the current domain.

<domain name>: Each CSN definition set, except the base set, must use a domain name. The domain name choice is left up to you.

The example below defines the member type of objects to be **Roads::TRIM**. Roads is the <class name> component and **TRIM** is the <domain name> component.

```
SAIF_DEF Roads::TRIM
```

SAIF_COMPOSITE_CLASS

This is the name of the composite class into which the SAIF objects of type <membertype> are stored. This is the actual SAIF class which is addressable. Each **DEF** line defines a single addressable object of the type specified here. If this is not specified, then a default value is generated which inserts the word Composite immediately before the double :: specified in <membertype>.

Using the above example, the default value of **SAIF_COMPOSITE_CLASS** is **RoadsComposite::TRIM**. This value can be overridden simply by specifying the **SAIF_COMPOSITE_CLASS** parameter. The example below overrides the default value instead specifying **RoadsCollection::TRIM**.

```
SAIF_COMPOSITE_CLASS RoadsCollection::TRIM
```

SAIF_IDENTIFIER

This is the identifier used to identify the SAIF addressable object being defined by this **SAIF_DEF** line. If not specified, the default value for the identifier is the same as the <membertype> with the double colon and the domain name removed.

Using the example above, the default value of **SAIF_IDENTIFIER** is **Roads**. This value can be overridden by specifying the **SAIF_IDENTIFIER** parameter. The example below overrides the default value specifying **TRIMRoads**.

```
SAIF_IDENTIFIER TRIMRoads
```

SAIF_OBJECTSET

This defines the object set into which the addressable object being defined is to be stored. If not specified, the default value for the identifier is generated using the first 4 characters of the <membertype> followed by the last two characters before the double colon (::). If the <class name> portion of the <membertype> is less than 6 characters, the object set name is taken to be equal to <class name>.

Using the example above, the default value of `SAIF_OBJECTSET` is `roads`. This value can be overridden by specifying the `SAIF_OBJECTSET` parameter. The example below overrides the default value specifying `troads`.

```
SAIF_OBJECTSET troads
```

SAIF_AGGREGATE

This defines the aggregate into which the objects are to be placed. This parameter is only used when the features being defined are stored within a SAIF aggregate, which is itself stored within another aggregate. An example of when this occurs is the SAIF DEM into which DEMpoints, Breaklines, and other aggregates are stored. See the *SAIF Formal Specification* for a description of SAIF aggregates. The example below shows how to specify the aggregate for storing DEMpoints.

```
SAIF_AGGREGATE geoComponents{0}.position.geometry.masspoints
```

The statement above gives the full path to where the features belonging to this SAIF_DEF line are placed. They are placed within the aggregate identified by `position.geometry.masspoints` which itself is stored as the first element within the aggregate `geoComponents`.

Usually, this line follows immediately after one or more `<geoComponents spec>` lines, which are described below.

SAIF_COMPONENT_PATH

The SAIF component path defines the path to the aggregate into which the features belonging to this SAIF_DEF are placed. By default, the value of this parameter is `geoComponents` as this is the value used in the majority of cases. An example of when the value needs to be overridden is when the feature represents a SAIF text object. In this case, the `SAIF_COMPONENT_PATH` should be specified as `annotationComponents`. The example below overrides the default value and specifies a component path of `annotationComponents`.

```
SAIF_COMPONENT_PATH annotationComponents
```

<geoComponents spec>

This portion of the SAIF_DEF line is required only when the SAIF_AGGREGATE line is used. These specifications define the `geoComponent` aggregate classes before the first feature arrives. This statement configures the aggregates so that the features associated with this SAIF_DEF line can be inserted into SAIF. The example below continues the example given for the SAIF_AGGREGATE line above, and defines the two aggregates required before DEMpoints can be stored within the SAIF dataset.

```
geoComponents{0}.Class PointsAndBreaklines::TRIM  
geoComponents{0}.position.geometry.Class MeasuredSurface
```

The first line defines the type of aggregate for the first element of the geoComponents aggregate. It is defined to be of the type `PointsAndBreaklines::TRIM`. The first part of this statement `geoComponents{0}` defines the path name of the aggregate. The `.Class` suffix instructs the underlying SAIF Toolkit, which the SAIF writer uses, that this is the name of the class for the object that has a path of `geoComponents`. See the *SAIF Toolkit* documentation for a full discussion of the `.Class` notation and the meaning of path names in SAIF.

The second line defines the type of aggregate for the object with the path name `position.geometry` within the aggregate `geoComponents{0}`. It is defined to be of the type `MeasuredSurface`.

Once this is defined, the SAIF_AGGREGATE line follows to define the aggregate where the DEMpoints will be stored.

<attr path> <attr value>

These lines are used to simply specify attribute path and attribute value pairs. The first part of the line identifies the attribute path to be set and the second part of the line specifies the value to be assigned to the attribute. These attribute values are used to set any attribute values at the aggregate level. The SAIF_DEF line cannot be used to set any attribute values for the features actually stored within the SAIF dataset.

CSN

Required/Optional: *Required*

The FME mapping file will have one or more SAIF_CSN file lines which define the CSN files that contain the SAIF class definitions for the objects to be stored within the SAIF dataset. If an attempt is made to define any object not of a class specified in the CSN files, then an error results and the FME session is stopped.

The example below defines two CSN files. The first file is the SAIF base set of classes and must always be the first CSN file specified. The second CSN file is a file that contains a set of domain-specific definitions.

```
SAIF_CSN saif32.csn
SAIF_CSN forest32.csn
```

XCOORD_TYPE

Required/Optional: *Required*

This is the numeric domain of the x coordinate. All coordinates stored within the SAIF dataset will have their x value in the domain specified on this line. The example below instructs the SAIF writer module that all x coordinates are of the type integer.

```
SAIF_XCOORD_TYPE STK_INT32
```

YCOORD_TYPE

Required/Optional: *Required*

This is the numeric domain of the y coordinate. All coordinates stored within the SAIF dataset will have their y value in the domain specified on this line. The example below instructs the SAIF writer module that all y coordinates are single precision floats.

```
SAIF_YCOORD_TYPE STK_REAL32
```

ZCOORD_TYPE

Required/Optional: *Required*

This is the numeric domain of the z coordinate. All coordinates stored within the SAIF dataset will have their z value in the domain specified on this line. The example below instructs the SAIF writer module that all z coordinates are double precision floats.

```
SAIF_ZCOORD_TYPE STK_REAL64
```

Feature Representation

SAIF features consist of a feature type, a geometry or text class, attribute path and attribute value pairs, and coordinates. A typical FME correlation line for SAIF has two forms: geometric entity form and text entity form.

Geometric Entity Form

This form of FME correlation line is used for all SAIF geometric entities. The line first specified is the **<member type>**. The value specified for **<member type>** must match a value specified in a **SAIF_DEF** line. The line then stipulates the type of geometry that the feature contains. The FME supports all SAIF geometries permitted by SAIF-Lite. Finally, the **<attribute path>** **<attribute value>** pairs are specified, as they are for any other formats. The only difference with SAIF is that the **<attribute path>** values may be of arbitrary depth. See the *SAIF Toolkit Application Programming Interface (API)* document for a discussion of attribute paths.¹

¹The SAIF-Lite specification and *SAIF Toolkit API* document describe the allowed geometry types and the attribute path syntax used by SAIF.

```
SAIF <member type> \  
    position.geometry.class <geometry class> \  
    [<attribute path> <attribute value>]*
```

Text Entity Form

This form of the FME correlation line is used for all SAIF text entities. The line is almost the same as the geometric entity above, except that instead of specifying a <geometry class>, a <text class> is specified. See the *SAIF Formal Specification* for a list of all the different SAIF text classes that can be specified.

```
SAIF <member type> \  
    textOrSymbol.class <text class> \  
    [<attribute path> <attribute value>]*
```

Spatial Data Transfer Standard (SDTS) Reader

This chapter provides the Feature Manipulation Engine (FME) with access to Spatial Data Transfer Standard (SDTS) formatted file sets.

Overview

While the SDTS reader should be able to import at least some data from any SDTS data source, it has been tested with USGS DLG dataset which adheres to the SDTS Topological Vector Profile (TVP) and USGS Digital Elevation Model (DEM) datasets that adhere to the SDTS Raster Profile.

More information on the SDTS Format can be found at:

<http://mcmcweb.er.usgs.gov/sdts/>

Some aspects of an SDTS transfer, such as data quality information, is not accessible via the SDTS reader, but can be extracted using the ISO 8211 reader by a user well versed in the SDTS format.

SDTS Quick Facts

Format Type Identifier	SDTS
Reader/Writer	Reader
Licensing Level	Base
Dependencies	None
Dataset Type	File
Feature Type	Feature role
Typical File Extensions	.ddf
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Never
Schema Required	Not applicable
Transaction Support	No
Geometry Type	sdts_type
Encoding Support	No

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	point	yes
circles	no	polygon	yes
circular arc	no	raster	no
donut polygon	yes	solid	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
elliptical arc	no		surface	no
ellipses	no		text	no
line	yes		z values	yes
none	yes			

Reader Overview

The SDTS Reader module produces FME features for all data within one SDTS transfer, a group of **.DDF** files identified by the ***CATD.DDF** file. Each vector object (point, line or polygon), attribute record of an attribute module or pixel within a raster image is translated into an FME feature.

Reader Directives

The directives processed by the SDTS reader are listed below. The suffixes shown are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the SDTS reader is **SDTS**.

DATASET

Required/Optional: *Required*

The file name of the catalog file relating the files of a single SDTS transfer. This will normally end in **CATD.DDF**, and binds together all the files with a common prefix. For instance, a USGS hypsography transfer might be accessed as shown. All files starting with **HP01** would be part of the same transfer.

SDTS_DATASET PALO_ALTO\HP01CATD.DDF

Workbench Parameter: *Source SDTS CATD File(s)*

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

All features read from SDTS transfers are assigned a feature type based on the SDTS module of the file they came from. Typically this module name is also part of the file name, so for instance, a line feature read from **HP01LE01.DDF** would have a feature type of **LE01** indicating it came from the 1st line module.

The types of SDTS modules supported are:

- **Point-Node:** A node in a topology, polygon label or freestanding point. Will have an **sdts_type** of **sdts_point**, a **PNTS** attribute, and possibly an **ARID** attribute. May have zero or more **ATID{n}** attributes.
- **Line:** A line, possibly part of a topology. It will have an **sdts_type** of **sdts_line**, and a unique identifier in the **LINE** attribute. If it is part of a topology, it will have a **PIDR**, **PIDL**, **SNID** and **ENID**. It may have one or more **ATID{n}** attributes.
- **Polygon:** A polygon. It will have an **sdts_type** of **sdts_polygon**, and a unique identifier in the **POLY** attribute. It may have one or more **ATID{n}** attributes. The polygon does not have any geometry as it emerges from the reader; however, the standard processing pipeline will assemble this from the related lines.
- **Attribute Primary:** A geometries attribute record with an **sdts_type** of **sdts_attr_primary**. It will contain additional attributes specific to the module based on the data product it is a part of, for instance the **ELEVATION** and **ENTITY_LABEL** specifically mentioned below. It will also have an **ATPR** attribute containing its unique identifier. Most primary attribute records are consumed by the default SDTS pipeline, **sdts_read.fmi**, as they are appended to their referencing point, line and polygon features.
- **Attribute Secondary:** A geometryless attribute record with an **sdts_type** of **sdts_attr_secondary**. It will contain additional attributes specific to the module based on the data product it is a part of, some of which will be keys into attributes on primary attribute records such as **COUNTY** and **STATE**. It will also have an **ATSC** attribute containing its unique identifier.
- **Cell or Raster:** Each pixel of data in a raster cell is translated into a point with an **sdts_type** of **sdts_point_dem**, but no **PNTS** attribute. The elevation of the point is in an attribute called **ELEVATION**.

Attribute Name	Description	Defined On
sdts_type	The type of this geometry. One of sdts_point , sdts_line , sdts_polygon or sdts_attr .	All features
PNTS	A unique identifier for the point in the current transfer.	sdts_point
ARID	A unique identifier for the area that this point labels.	sdts_point (optional)
LINE	A unique identifier for the line in the current transfer.	sdts_line
PIDL	Identifier for the left polygon from the current line. Relates to the POLY attribute.	sdts_line (optional)
PIDR	Identifier for the right polygon from the current line. Relates to the POLY attribute.	sdts_line (optional)
SNID	Identifier for the start node of the current line. Relates to the PNTS attribute.	sdts_line (optional)
ENID	Identifier for the end node of the current line. Relates to the PNTS attribute.	sdts_line (optional)

Attribute Name	Description	Defined On
POLY	A unique identifier for the polygon in the transfer.	sdts_polygon
ATID{n}	Identifier for an attribute record that applies to the current object. Relates to the ATPR attribute of the attribute records.	sdts_point (optional) sdts_line (optional) sdts_polygon (optional)
ATPR	A unique identifier for the primary attribute record within a transfer.	sdts_attr_primary
ATSC	A unique identifier for the secondary attribute record within a transfer. Not normally useful for any purpose.	sdts_attr_secondary
ELEVATION	The elevation of the feature. Will be found on points from DEM raster transfers. It is also found in a primary attribute record related to hypsography features such as contour lines and is attached to the features by the default pipeline.	sdts_point_dem sdts_attr_primary (optional)
ENTITY_LEVEL	USGS DLG transfers have this primary attribute for most features. A detailed listing of meaningful values and other DLG-3 specific attributes is contained in the USGS document <i>DLG-3 SDTS Transfer Description</i> which may be found at the following website address: ftp://sdts.er.usgs.gov/pub/sdts/data/sets/tvp/dlg3/dlg3sdts.ps	sdts_attr_primary (optional)

The following is an example of the attributes on a polygon feature from a vegetation surface cover DLG transfer, after edge geometry has been merged and primary attribute records have been attached by the standard processing pipeline (`sdts_read.fmi`).

```

Feature Type: PC01'
ATID' is ASCF_86'
ATPR' is ASCF_86'
BEST_ESTIMATE' is '
ENTITY_LABEL' is 0700101'
POLY' is PC01_123'
POLY_OBRP' is PC'
fme_geometry' is fme_polygon'
sdts_type' is sdts_polygon'
Geometry Type: Polygon (4)
Number of Coordinates: 38 -- Coordinate Dimension: 2 -- Coordinate System: 0'

```

Secondary Attributes

Some SDTS transfers include secondary attribute modules. These are essentially tables related to data fields in one or more primary attribute tables. One example of this is DLG-3 1:2000000 boundary datasets which keep the county names in secondary tables along with the state, and county numbers.

The default SDTS pipeline, `sdts_read.fmi`, does not include factories to append these secondary records to their target features because the names of the key fields vary depending on the data product.

By default, secondary records are passed through and are available as output features to write to an output file. For instance, the following definition is produced for county names.

```
SDrobert-findlerBFPC \  
  sdts_type          sdts_attr_secondary \  
  ATSC               %ATSC \  
  COUNTY             %COUNTY \  
  STATE              %STATE \  
  AREA_NAME          %AREA_NAME
```

The `ATSC` field is a record identifier but isn't generally useful for anything, since it doesn't relate to fields in any other records. The `COUNTY` and `STATE` values are numeric identifiers and can be related to the `COUNTY` and `STATE` fields that get attached to political polygons from a primary attribute module using a factory like that shown in the following example. It could be placed in a mapping file in order to append county names to polygon features after processing by the default pipeline.

```
FACTORY_DEF * ReferenceFactory \  
  FACTORY_NAME CountyNameAppender \  
  INPUT REFEREEC FEATURE_TYPE * \  
    sdts_type sdts_attr_secondary \  
  INPUT REFERENCER FEATURE_TYPE * \  
    sdts_type sdts_polygon \  
  REFEREEC_FIELDS STATE COUNTY \  
  REFERENCER_FIELDS STATE COUNTY \  
  REFERENCE_INFO ATTRIBUTES \  
  OUTPUT COMPLETE FEATURE_TYPE * sdts_type sdts_polygon \  
  OUTPUT NO_REFERENCES FEATURE_TYPE * \  
  OUTPUT INCOMPLETE FEATURE_TYPE * \  
  OUTPUT UNREFERENCED FEATURE_TYPE *
```

SQLite Reader/Writer

Overview

The SQLite reader and writer modules provide FME with access to attribute data held in sqlite3 database tables. This data may not necessarily have a spatial component to it. FME provides read and write access to sqlite3 databases.

Tip: See the @SQL function in the FME Functions and Factories manual. This function allows arbitrary Structured Query Language (SQL) statements to be executed against any database.

SQLite Quick Facts

Format Type Identifier	SQLITE3
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	Database
Feature Type	Table name
Typical File Extensions	.db .sl3
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	Yes
Encoding Support	Yes
Geometry Type	db_none

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	no
circles	no		polygon	no
circular arc	no		raster	no
donut polygon	no		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	no		z values	n/a
none	yes			

Reader Overview

FME considers a database data set to be a collection of relational tables. The tables must be defined in the mapping file before they can be read. Arbitrary WHERE clauses and joins are fully supported.

Reader Directives

The suffixes listed are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the SQLite3 reader is **SQLITE3**.

DATASET

Required/Optional: *Required*

This is the file name of the SQLite3 Database.

For example,

```
SQLITE3_DATASET c:/data/citySource.db
```

Workbench Parameter: *Source SQLite3 Database File(s)*

DEF

Required/Optional: *Required*

The syntax of the definition is:

```
SQLITE3_DEF <tableName> \  
[sqlite3_sql_statement <sqlQuery>] \  
[sqlite3_where_clause <whereClause>] \  
[<fieldName> <fieldType>] +
```

OR

```
SQLITE3_DEF <queryName> \  
[sqlite3_sql_statement <sqlQuery>] \  
]
```

The **<tableName>** must match the name of an existing SQLite3 table in the database. This will be used as the feature type of all the features read from the table. The exception to this rule is when using the `sqlite3_sql_statement` keyword. In this case, the **DEF** name may be any valid alphabetic identifier; it does not have to be an existing table name – rather, it is an identifier for the custom SQL query. The feature type of all the features returned from the SQL query are given the query name.

The **<fieldType>** of each field must be given, but it is not verified against the database definition for the field. In effect, it is ignored.

The definition allows specification of separate search parameters for each table. If any of the per table configuration parameters are given, they will override, for that table, whatever global values have been specified by the reader keywords such as the **WHERE_CLAUSE**. If any of these parameters is not specified, the global values will be used.

The following table summarizes the definition line configuration parameters:

Parameter	Contents
sqlite3_where_clause	This specifies the SQL WHERE clause applied to the attributes of the layer's features to limit the set of features returned. If this is not specified, then all the rows are returned. This keyword will be ignored if the <code>sql3_sql_statement</code> is present.
sqlite3_sql_statement	This specifies an SQL SELECT query to be used as the

Parameter	Contents
	source for the results. If this is specified, the SQLite3 reader will execute the query, and use the resulting rows as the features instead of reading from the table <queryName>. All returned features will have a feature type of <queryName>, and attributes for all columns selected by the query. The sqlite3_where_clause is ignored if sqlite3_sql_statement is supplied. This form allows the results of complex joins to be returned to FME.

If no <whereClause> is specified, all rows in the table will be read and returned as individual features. If a <whereClause> is specified, only those rows that are selected by the clause will be read. Note that the <whereClause> does not include the word **WHERE**.

The SQLite3 reader allows one to use the sqlite3_sql_statement parameter to specify an arbitrary SQL **SELECT** query on the DEF line. If this is specified, FME will execute the query, and use each row of data returned from the query to define at least one feature. Each of these features will be given the feature type named in the DEF line, and will contain attributes for every column returned by the **SELECT**. In this case, all DEF line parameters regarding a **WHERE** clause or spatial querying are ignored, as it is possible to embed this information directly in the text of the <sqlQuery>.

In the following example, all the records whose ID is less than 5 will be read from the supplier table:

```
SQLITE3_DEF supplier \
  sqlite3_where_clause "id < 5" \
  ID integer \
  NAME text \
  CITY text
```

In this example, the results of joining the **employee** and **city** tables are returned. All attributes from the two tables will be present on each returned feature. The feature type will be set to **complex**.

```
SQLITE3_DEF complex \
  sqlite3_sql_statement \
  "SELECT * FROM EMPLOYEE, CITY WHERE EMPLOYEE.CITY = CITY.NAME"
```

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined database tables that will be read. If no **IDs** are specified, then all tables are read. The syntax of the **IDs** keyword is:

```
SQLITE3_IDS <featureType1> \
  <featureType2> ... \
  <featureTypeN>
```

The feature types must match those used in DEF lines.

The example below selects only the **HISTORY** table for input during a translation:

```
SQLITE3_IDS HISTORY
```

RETRIEVE_ALL_SCHEMAS

Required/Optional: *Optional*

This specification is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

When set to "Yes", indicates to the reader to return all the schemas of the tables in the database.

If this value is not specified, it is assumed to be "No".

Range: YES | NO

Default: NO

RETRIEVE_ALL_TABLE_NAMES

Required/Optional: *Optional*

This specification is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

Similar to **RETRIEVE_ALL_SCHEMAS**; this optional directive is used to tell the reader to only retrieve the table names of all the tables in the source database. If **RETRIEVE_ALL_SCHEMAS** is also set to "Yes", then **RETRIEVE_ALL_SCHEMAS** will take precedence. If this value is not specified, it is assumed to be "No".

Range: YES | NO

Default: NO

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the **mitab.dll** in the FME home directory to **mapinfo.dll**.

The syntax of the **MAPINFO_SEARCH_ENVELOPE** directive is:

```
MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

```
MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The **COORDINATE_SYSTEM** directive, which specifies the coordinate system associated with the data to be read, must always be set if the **SEARCH_ENVELOPE_COORDINATE_SYSTEM** directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the **SEARCH_ENVELOPE_COORDINATE_SYSTEM** to the reader **COORDINATE_SYSTEM** prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The SQLite3 writer module stores attribute records into a live relational database. The SQLite3 writer provides the following capabilities:

- **Transaction Support:** The SQLite3 writer provides transaction support that eases the data loading process. Occasionally, a data load operation terminates prematurely due to data difficulties. The transaction support provides a mechanism for reloading corrected data without data loss or duplication.
- **Table Creation:** The SQLite3 writer uses the information within the FME mapping file to automatically create database tables as needed.
- **Writer Mode Specification:** The SQLite3 writer allows the user to specify what database command should be issued for each feature received. Valid writer modes are INSERT, UPDATE and DELETE. The writer mode can be specified at three unique levels: at the writer level, on the feature type, or on individual features.

Writer Directives

The directives listed below are processed by the SQLite3 writer. The suffixes shown are prefixed by the current **<WriterKeyword>** in a mapping file. By default, the **<WriterKeyword>** for the SQLite3 writer is **SQLITE3**.

DATASET

Required/Optional: *Required*

The **DATASET** directive operates in the same manner as it does for the SQLite3 reader.

Workbench Parameter: *Destination SQLite3 Database File*

DEF

Required/Optional: *Required*

Each SQLite3 table must be defined before it can be written. The general form of a SQLite3 definition statement is:

```
SQLITE3_DEF <tableName> \
    [sqlite3_update_key_columns <keyColumns>] \
    [sqlite3_drop_table (yes|no)] \
    [sqlite3_truncate_table (yes|no)] \
    [sqlite3_table_writer_mode (inherit_from_writer|insert|
        update|delete)] \
    [<fieldName> <fieldType>[,<indexType>]]+
```

The table definition allows control of the table that will be created. If the fields and types are listed, the types must match those in the database. Fields which can contain NULL values do not need to be listed - these fields will be filled with NULL values.

If the table does not exist, then the field names and types are used to first create the table. In any case, if a **<field-
Type>** is given, it may be any field type supported by the target database.

The configuration parameters present on the definition line are described in the following table:

Parameter	Contents
tableName	The name of the table to be written. If a table with the specified name exists, it will be overwritten if the <code>sqlite3_drop_table</code> DEF line parameter is set to YES , or it will be truncated if the <code>sqlite3_truncate_table</code> DEF line parameter is set to YES . Otherwise the table will be appended. Valid values for table names include any character string devoid of SQL-offensive characters (the " is the only SQL-offensive character in SQLite) and less than 255 characters in length.
sqlite3_table_writer_mode	The default operation mode of the feature type in terms of the types of SQL statements sent to the database. Valid values are INSERT , UPDATE , DELETE and INHERIT_FROM_WRITER . Note that INSERT mode allows for only INSERT operations where as UPDATE and DELETE can be overwritten at the feature levels. INHERIT_FROM_WRITER simply indicates to take this value from the writer level and not to override it at the feature type level. Default: INHERIT_FROM_WRITER
sqlite3_update_key_columns	This is a comma-separated list of the columns which are matched against the corresponding FME attributes' values to

Parameter	Contents
	<p>specify which rows are to be updated or deleted when the writer mode is either UPDATE or INSERT.</p> <p>For example: sqlite3_update_key_columns ID would instruct the writer to ensure that the ID attribute is always matched against the column with the same name. Also, the target table is always the feature type specified in the DEF line.</p> <p>Each column listed with the <code>sqlite3_update_key_columns</code> keyword must be defined with a type on the DEF line, in addition to the columns whose values will be updated by the operation.</p>
sqlite3_drop_table	<p>This specifies that if the table exists by this name, it should be dropped and replaced with a table specified by this definition.</p> <p>Default: NO</p>
sqlite3_truncate_table	<p>This specifies that if the table exists by this name, it should be cleared prior to writing.</p> <p>Default: NO</p>
fieldName	<p>The name of the field to be written. Valid values for field name include any character string devoid of SQL-offensive characters (the " is the only SQL-offensive character in SQLite) and less than 255 characters in length.</p>
fieldType	<p>The type of a column in a table. The valid values for the field type are listed below:</p> <p>blob float integer real(width, decimal) text varchar(width)</p>
indexType	<p>The type of index to create for the column.</p> <p>If the table does not previously exist, then upon table creation, a database index of the specified type is created. The database index contains only the one column.</p> <p>The valid values for the column type are listed below:</p> <p>indexed: An index without constraints. unique: An index with a unique constraint.</p>

START_TRANSACTION

Required/Optional: *Optional*

This statement tells the SQLite3 writer module when to start actually writing features into the database. The SQLite3 writer does not write any features until the feature is reached that belongs to **<last successful transaction> + 1**. Specifying a value of zero causes every feature to be output. Normally, the value specified is zero – a non-zero value is only specified when a data load operation is being resumed after failing partway through.

Parameter	Contents
<last successful transaction>	The transaction number of the last successful transaction. When loading data for the first time, set this value to 0 . Default: 0

Example:

`SQLITE3_START_TRANSACTION 0`

Workbench Parameter: *Start transaction at*

TRANSACTION_INTERVAL

Required/Optional: *Optional*

This statement informs the FME about the number of features to be placed in each transaction before a transaction is committed to the database.

If the `SQLITE3_TRANSACTION_INTERVAL` statement is not specified, then a value of 500 is used as the transaction interval.

Parameter	Contents
<transaction_interval>	The number of features in a single transaction. Default: 500

If the `SQLITE3_TRANSACTION_INTERVAL` is set to zero, then feature based transactions are used. As each feature is processed by the writer, they are checked for an attribute called `fme_db_transaction`. The value of this attribute specifies whether the writer should commit or rollback the current transaction. The value of the attribute can be one of `COMMIT_BEFORE`, `COMMIT_AFTER`, `ROLLBACK_AFTER` or `IGNORE`. If the `fme_db_transaction` attribute is not set in any features, then the entire write operation occurs in a single transaction.

Example:

`SQLITE3_TRANSACTION_INTERVAL 5000`

Workbench Parameter: *Transaction interval*

WRITER_MODE

Required/Optional: *Optional*

Note: For more information on this directive, see the chapter *Database Writer Mode*.

This directive informs the SQLite3 writer which SQL operations will be performed by default by this writer. This operation can be set to `INSERT`, `UPDATE` or `DELETE`. The default writer level value for this operation can be overwritten at the feature type or table level. The corresponding feature type DEF parameter name is called `sqlite3_table_writer_mode`. It has the same valid options as the writer level mode and additionally the value `INHERIT_FROM_WRITER` which causes the writer level mode to be inherited by the feature type as the default for features contained in that table.

The operation can be set specifically for individual features as well. Note that when the writer mode is set to INSERT this prevents the mode from being interpreted from individual features and all features are inserted unless otherwise marked as UPDATE or DELETE features. These are skipped.

If the SQLITE3_WRITER_MODE statement is not specified, then a value of INSERT is given.

Parameter	Contents
<writer_mode>	The type of SQL operation that should be performed by the writer. The valid list of values are below: INSERT UPDATE DELETE Default: INSERT

Example:

```
SQLITE3_WRITER_MODE INSERT
```

Workbench Parameter: *Writer Mode*

BEGIN_SQL{n}

Occasionally you must execute some ad-hoc SQL prior to opening a table. For example, it may be necessary to ensure that a view exists prior to attempting to read from it.

Upon opening a connection to read from a database, the reader looks for the directive <ReaderKeyword>_BEGIN_SQL{n} (for n=0,1,2,...), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the FME_SQL_DELIMITER keyword, embedded at the beginning of the SQL block. The single character following this keyword will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;
DELETE FROM instructors;
DELETE FROM people
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

*** Workbench Parameter**

SQL Statement to Execute Before Translation

END_SQL{n}

Occasionally you must execute some ad-hoc SQL after closing a set of tables. For example, it may be necessary to clean up a temporary view after writing to the database.

Just before closing a connection on a database, the reader looks for the directive `<ReaderKeyword>_END_SQL{n}` (for $n=0,1,2,\dots$), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the `FME_SQL_DELIMITER` directive, embedded at the beginning of the SQL block. The single character following this directive will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
FME_SQL_DELIMITER ;
DELETE FROM instructors;
DELETE FROM people
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

* Workbench Parameter

SQL Statement to Execute After Translation

INIT_TABLES

Required/Optional: *Optional*

This directive informs the SQLite3 writer when each table should be initialized. Initialization encompasses the actions of dropping or truncating existing tables, and creating new tables as necessary.

When `INIT_TABLES` is set to `IMMEDIATELY`, the SQLite3 writer will initialize all tables immediately after parsing the `DEF` lines and opening the database file. In this mode, all tables will be initialized, even if the SQLite3 writer receives no features for a given table.

When `INIT_TABLES` is set to `FIRSTFEATURE`, the SQLite3 writer will only initialize a table once the first feature destined for that table is received. In this mode, if the SQLite3 writer does not receive any features for a given table, the table will never be initialized.

Workbench Parameter: *Initialize Tables*

Writer Mode Specification

The SQLite3 writer allows the user to specify a writer mode, which determines what database command should be issued for each feature received. Valid writer modes are `INSERT`, `UPDATE` and `DELETE`.

Writer Modes

In `INSERT` mode, the attribute values of each received feature are written as a new database record.

In `UPDATE` mode, the attribute values of each received feature are used to update existing records in the database. The records which are updated are determined via the `sqlite3_update_key_columns` DEF line parameter, or via the `fme_where` attribute on the feature.

In `DELETE` mode, existing database records are deleted according to the information specified in the received feature. Records are selected for deletion using the same technique as records are selected for updating in `UPDATE` mode.

Writer Mode Constraints

In **UPDATE** and **DELETE** mode, the **fme_where** attribute always takes precedence over the **sqlite3_update_key_columns** DEF line parameter. If both the **fme_where** attribute and the **sqlite3_update_key_columns** DEF line parameter are not present, then **UPDATE** or **DELETE** mode will generate an error.

When the **fme_where** attribute is present, it is used verbatim as the WHERE clause on the generated **UPDATE** or **DELETE** command. For example, if **fme_where** were set to `'id<5'`, then all database records with field ID less than 5 will be affected by the command.

When the **fme_where** attribute is not present, the writer looks for the **sqlite3_update_key_columns** DEF line parameter and uses it to determine which records should be affected by the command. Please refer to DEF for more information about the **sqlite3_update_key_columns** DEF line parameter.

Writer Mode Selection

The writer mode can be specified at three unique levels. It may be specified on the writer level, on the feature type or on individual features.

At the writer level, the writer mode is specified by the **WRITER_MODE** keyword. This keyword can be superseded by the feature type writer mode specification. **Note:** For more information on this directive, see the chapter *Database Writer Mode*.

At the feature type level, the writer mode is specified by the **sqlite3_writer_mode** DEF line parameter. This parameter supersedes the **WRITER_MODE** keyword. Unless this parameter is set to **INSERT**, it may be superseded on individual features by the **fme_db_operation** attribute. Please refer to the DEF line documentation for more information about this parameter.

At the feature level, the writer mode is specified by the **fme_db_operation** attribute. Unless the parameter at the feature type level is set to **INSERT**, the writer mode specified by this attribute always supersedes all other values. Accepted values for the **fme_db_operation** attribute are **INSERT**, **UPDATE** or **DELETE**.

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Features read from a database consist of a series of attribute values. They have no geometry. The attribute names are as defined in the **DEF** line if the first form of the **DEF** line was used. If the second form of the **DEF** line was used, then the attribute names are as they are returned by the query, and as such may have their original table names as qualifiers. The feature type of each SQLite3 feature is as defined on its **DEF** line.

Features written to the database have the destination table as their feature type, and attributes as defined on the **DEF** line.

Standard Linear Format (SLF) Reader

The Standard Linear Format (SLF) Reader module provides the Feature Manipulation Engine (FME) with the capability to read SLF files. This chapter assumes familiarity with the SLF format.

Overview

SLF data sets are ASCII format files that use the chain-node—also referred to as link-node or segment-node—data structure. This means that regardless of the number of features a segment might belong to, that segment is stored only once in the data set. By not storing repeated segments, the SLF format simplifies updates and corrections. It also avoids overlap and gap problems, as well as being responsive to thinning and generalization algorithms.

An SLF file logically consists of these four sequential records:

- the Data Set Identifier (**DSI**) record
- the Segment (**SEG**) record
- the Feature (**FEA**) record
- the Text (**TXT**) record

The **DSI**, **SEG**, and **FEA** records are required for each SLF data set, whereas the **TXT** record is optional. The contents of the **TXT** record are product-specific and the only type of **TXT** record recognized by the SLF reader is when SLF is used to implement Interim Terrain Data (ITD) in two-dimensional (2D) format.

The FME accepts any valid SLF file as input, regardless of file name or extension.

This section outlines the features and attributes produced directly by the SLF reader. The **slf_point**, **slf_linear**, and **slf_area** features produced by the SLF reader only contain references to the segments that make up the feature. By themselves, these features do not contain any coordinates. Through using a sequence of FME factories, specially designed to process the SLF features, the FME can re-assemble the sequence of coordinates that make up each feature from the segments. This sequence of factories is described at the end of this chapter, under the heading Features Created by the FME Factories.

SLF Quick Facts

Format Type Identifier	SLF
Reader/Writer	Reader
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	Geometry based name
Typical File Extensions	.slf
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Not applicable
Transaction Support	No
Geometry Type	slf_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	yes		z values	no
none	yes			

Reader Overview

The SLF reader opens the input file and returns a feature to represent the data from each SLF logical record it encounters. The reader does not have any requirement for definition statements.

Each feature returned by the SLF reader has its FME feature type set to one of the following: `slf_segment`, `slf_point`, `slf_linear`, `slf_areal`, `slf_txt` or `slf_dsi`.

Reader Directives

The directives processed by the SLF reader are listed below. The suffixes shown are prefixed by the current `<ReaderKeyword>` in a mapping file. By default, the `<ReaderKeyword>` for the SLF reader is `SLF`.

DATASET

Required/Optional: *Required*

The value for this keyword is the file containing the SLF data set to be read. A typical mapping file fragment specifying an input SLF file looks like:

```
SLF_DATASET /usr/slfdata/slffile
```

Workbench Parameter: *Source Standard Linear Format (SLF) File(s)*

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the `mitab.dll` in the FME home directory to `mapinfo.dll`.

The syntax of the `MAPINFO_SEARCH_ENVELOPE` directive is:

```
MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

```
MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The `COORDINATE_SYSTEM` directive, which specifies the coordinate system associated with the data to be read, must always be set if the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` to the reader `COORDINATE_SYSTEM` prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the `SEARCH_ENVELOPE` directive.

Values

YES | NO (default)

Mapping File Syntax

<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]

* Workbench Parameter

Clip To Envelope

Feature Representation

All SLF features contain the `slf_type` attribute. The value for this attribute is set identically to the value of the feature's feature type.

Attribute Name	Contents
<code>slf_type</code>	Specifies what type of data the feature represents. Range: <code>slf_segment</code> <code>slf_point</code> <code>slf_linear</code> <code>slf_areal</code> <code>slf_txt</code> <code>slf_dsi</code> Default: No default

Segments

slf_type: `slf_segment`

The SLF segments contain the coordinate sequence that make up the SLF point, linear, and areal features.

These attributes are specific to the segment features.

Attribute Name	Contents
<code>slf_seg_segment</code>	This is the unique identification number given to the segment.
<code>slf_seg_feature_count</code>	A segment can be referenced by various features. This number keeps a count of the number of features referencing the segment.
<code>slf_seg_point_count</code>	The number of coordinates contained in this segment.

Points

slf_type: `slf_point`

The `slf_point` features do not have any coordinate. An `slf_point` feature is allowed to reference more than one segment. That is, an `slf_point` may be a collection of points having all attributes in common except for their geographical location.

The following table lists the attributes common to all `slf_point`, `slf_linear`, and `slf_areal` features.

Attribute Name	Contents
<code>slf_fea_feature_id</code>	This is the unique identification number given to the feature.

Attribute Name	Contents
slf_fea_feature_header_block_count	The slf_feature_header is made up of a sequence of 40 byte blocks. This attribute counts the number of 40 byte blocks that make up the slf_feature_header . For ITD data, the value of this attribute is 7.
slf_feature_header	Contains descriptive information for the feature. In the case of ITD data, the length for the value of this attribute is 280 bytes.
slf_fea_segment_count	Counts the number of segments that make up the feature.
slf_segmentID{#}.id	A list of unique segment IDs that make up the feature.

Linear

slf_type: slf_linear

Linear features output by the SLF reader do not have any coordinates. The attributes for linear features are identical to the attributes found for slf_point features.

Areal

slf_type: slf_areal

Areal¹ features output by the SLF reader do not have any coordinates. The attributes for areal features are identical to the attributes found for slf_point features.

Text

slf_type: slf_txt

At this time, only the **TXT** records for ITD data are recognized and processed. If the SLF data set is not ITD, then the **TXT** record is ignored.

The SLF reader outputs two types of slf_txt features when reading ITD data, distinguishable by the value of their slf_txt_type attribute. The slf_txt_type attribute may contain the following values:

- misc_or_not_eval—these are features output when processing the Miscellaneous/Not Evaluated **TXT** record
- surface_material—these are features output when processing the Surface Roughness Information **TXT** record

The following table lists the attributes for **txt** features having misc_or_not_eval as the value of their slf_txt_type attribute.

Attribute Name	Contents
slf_txt_feature_type	Indicates whether the feature is a miscellaneous feature (9D010) or a not evaluated feature (9D020). There is no default.

¹An areal feature is just a polygon. The SLF's specification always refers to these features as areal features therefore, to assist those readers familiar with the SLF, it is referred to as areal throughout this section.

Attribute Name	Contents
slf_txt_feature_id	This is the feature slf_point , slf_linear , or slf_areal to which this slf_txt feature is referred.
slf_txt_feature_description	This is where you find descriptive information about this feature.

The following table lists the attributes for **txt** features having **surface_material** as the value of their **slf_txt_type** attribute.

Attribute Name	Contents
slf_srq_value	Surface roughness quality value
slf_srq_description	Description of surface roughness type

DSI

slf_type: slf_dsi

There is exactly one feature of this type for every SLF file processed. The **slf_dsi** feature contains no geometry. It is a place holder for all descriptive information common to all data. The data is contained in the **DSI** record of the SLF file.

The following table lists the attributes that the **slf_dsi** feature contains. Values for some attributes listed may be blank. This is dependent on the actual information contained in the DSI record. Only relevant **DSI** attributes, as described in *Appendix M, Implementing Interim Terrain Data (ITD) in 2-D SLF*, of the *MIL-STD-2413* document, are listed here.

*Tip: Any DSI fields not listed in the following table are still attributes in the **slf_dsi** feature. A log of the feature shows all attributes that the feature contains.*

Attribute Name	Contents	# of Bytes
slf_dsig_product_type	Specifies the product type. Possible values are: Digital Interim Terrain Data/Tactical (DITDT) or Digital Interim Terrain Data/Planning (DITDP).	5
slf_dsig_data_set_id	This is the data set identification. Range: String	20
slf_dsig_edition	Edition number of the data set. For a new data set, the edition number is 1. Range: 1..999	3
slf_dsig_compilation_date	Date the data set was compiled. Range: YYYY	4
slf_dsig_maintenance_date	Date the data set was updated. Set to 0000 for new data. Range: YYYY	4
slf_dsig_SLF_version_date	Date of the SLF version that applies to the data set.	6

Attribute Name	Contents	# of Bytes
	Range: YYMMDD	
slf_dsig_DMAFF_version_date	The Defense Mapping Agency Feature Format (DMAFF) version date that applies to the data set. Range: YYMMDD	6
slf_dssg_security_classification	This is the security classification code. Range: T (Top Secret) S (Secret) C (Confidential) F (For Official Use Only) R (Restricted) U (Unclassified)	1
slf_dssg_security_release	Security control and release code based on source and ancillary materials.	2
slf_dssg_downgrading_declass_date	Downgrading and declassification date. Range: DDMMYY OADR (originating agency's determination required) blank spaces	6
slf_dssg_security_handling	Security handling description.	21
slf_dspg_data_type	The data type is set to GEO for ITD. Range: GEO (geographic coordinate data)	3
slf_dspg_horizontal_units_of_measure	Measuring system for the horizontal coordinates. Range: SEC (geographic seconds)	3
slf_dspg_horizontal_resolution_units	Number of units of measure that constitute the least count of the horizontal coordinate system. Range: 0.010 (for 1:50,000 scale ITD) 0.020 (for 1:250,000 scale ITD)	5
slf_dspg_geodetic_datum	Reference system code for horizontal positions. Range: World Geodetic System 1984 (WGE) or	3

Attribute Name	Contents	# of Bytes
	a local datum from Appendix B, Datums, of MIL-STD_2413 when no conversion to WGE exists.	
slf_dspg_ellipsoid	Ellipsoid to which the horizontal datum is referenced. Range: WGE Ellipsoid codes from <i>Appendix E, Ellipsoids of MIL-STD-2413</i>	3
slf_dspg_vertical_resolution_units	Number of units of measure that constitute the least count of the vertical coordinate system. Range: b.bbb (where b is a blank)	5
slf_dspg_vertical_reference_system	Set to Mean Sea Level (MSL) for ITD. Range: MSL	4
slf_dspg_latitude_of_origin	Latitude to which all geographic coordinates in the data set are referenced. Range: DDMMSSSH	9
slf_dspg_longitude_of_origin	Longitude to which all geographic coordinates in the data set are referenced. Range: DDDMMSSSH	10
slf_dspg_latitude_of_SW_corner	Southernmost latitude of the data set. Range: DDMMSSSH	9
slf_dspg_longitude_of_SW_corner	Westernmost longitude of the data set. Range: DDDMMSSSH	10
slf_dspg_latitude_of_NE_corner	Northernmost latitude of the data set. Range: DDMMSSSH	9
slf_dspg_longitude_of_NE_corner	Easternmost longitude of the data set. Range: DDMMSSSH	10
slf_dspg_total_number_of_features	The total number of features in the data set. Range: Integer >= 0	6
slf_dspg_number_of_point_features	The total number of point features in the data set. Range: Integer >= 0	6

Attribute Name	Contents	# of Bytes
slf_dspg_number_of_linear_features	The total number of linear features in the data set. Range: Integer >= 0	6
slf_dspg_number_of_areal_features	The total number of areal features in the data set. Range: Integer >= 0	6
slf_dspg_total_number_of_segments	The total number of segments in the data set. Range: Integer >= 0	6
slf_dshg_edition_code	The first two digits of the edition code is the recompilation code. The third digit is the revision count. The edition code is set to 000 for first-time coverage of a geographic area.	3
slf_dshg_project	Project Specification. Range: String	15
slf_dshg_spec_date	Product specification date. Range: YYYY	4
slf_dshg_spec_amendment_number	Product specification amendment number. Range: 1..999	3
slf_dshg_producer	Producer of the data. Range: String	8
slf_dshg_digitizing_system	The system used to digitize the data set. Range: String	10
slf_dshg_processing_system	The system used to process the data set. Range: String	10
slf_dshg_data_generalization	The data generalization code is set to 0 for ITD.	1
slf_dshg_north_match_merge_number	Number of times that this data set has merged with the adjacent data to the north side. The value is zero-filled for new data. Range: Integer	1
slf_dshg_east_match_merge_number	Number of times that this data set has merged with the adjacent data to the east side. The value is zero-filled for new data. Range: Integer	1

Attribute Name	Contents	# of Bytes
slf_dshg_south_match_merge_number	Number of times that this data set has merged with the adjacent data to the south side. The value is zero-filled for new data. Range: Integer	1
slf_dshg_west_match_merge_number	Number of times that this data set has merged with the adjacent data to the west side. The value is zero-filled for new data. Range: Integer	1
slf_dshg_north_match_merge_date	North match merge date. The value is zero-filled for new data. Range: YYYYMM	4
slf_dshg_east_match_merge_date	East match merge date. The value is zero-filled for new data. Range: YYYYMM	4
slf_dshg_south_match_merge_date	South match merge date. The value is zero-filled for new data. Range: YYYYMM	4
slf_dshg_west_match_merge_date	West match merge date. The value is zero-filled for new data. Range: YYYYMM	4
slf_dshg_yy_mm_of_earliest_source	Year and month of the earliest source material. Range: YYYYMM	4
slf_dshg_yy_mm_of_latest_source	Year and month of the latest source material. Range: YYYYMM	4
slf_dshg_data_conversion_code	Set to 001 for ITD (1:50,000 scale). Otherwise, it is NULL .	3

Features Created by the FME Factories

The geometric information for the slf_point, slf_linear, and slf_areal features are indirectly coded into the features by their **slf_segmentID{#}.id** list attribute. This section describes the sequence of factories used in the mapping file to pull in the geometric information from the slf_segment features for these features.

SLF features returned by the SLF factory pipeline have their FME feature type set to one of the following: slf_point, **slf_linear**, **slf_areal**, **slf_txt** or **slf_dsi**. Features of the slf_segment type are not present any more because after factory processing, the point, linear, and areal features contain their own coordinates. The special attribute slf_type still has its value identically set to the feature's feature type.

Each `slf_point` feature that references more than one `slf_segment` feature is split into several `slf_point` features. Each of the resulting `slf_point` features has a different coordinate taken from each segment referenced by the original point. All resulting points share the same attributes as the original point.

Some SLF point, linear or areal features may be classified as "miscellaneous" or "not evaluated". More information about these features can be found in the attributes of the `slf_txt` features having a `slf_text_type` of `misc_or_not_eval`. After exiting the sequence of factories, the SLF miscellaneous or not evaluated features contain the attributes of the `slf_txt` features that referenced them.

The sequence of factories that show the factory pipeline that some features from the SLF reader must enter can be found in the `slf/sl_factories.fmi` file in the FME installation directory.

Using the Multi-Reader and the SLF Reader for ITD 2D Data

The sequence of factories that show the FME factories needed when ITD 2D data is read with the Multi-Reader and the SLF Reader can be found in the `slf/sl_multi_factories.fmi` file in the FME installation directory. The Comma-Separated Value (CSV) table used in the *CorrelationFactory* can also be found in the FME installation directory in the `slf/itd_FN_attr.csv` file.

STAR-APIC Mercator MCF Reader/Writer

Format Notes: MERCATOR, formerly developed by Esko-Graphics, is now part of STAR-APIC (<http://www.star-apic.com>).

The Mercator MCF Reader/Writer module provides FME with access to the Mercator MCF format.

Overview

The Mercator MCF format is a collection of different types of data which include lines, areas, points, text, externals, and jobs. Each type of data is stored in a separate file, often with all the files of a particular dataset residing in the same directory. The files are stored in DTO format, have the extension `.dto` and are named `<dataset name>_<file type identifier>.dto`. The file type identifier is "l" for lines, "a" for areas, "p" for points, "e" for externals, "t" for text, and "j" for jobs.

The FME considers a Mercator MCF dataset to be a set of files in the same directory with the same dataset name. The FME is able to determine the type of data by the `<file type identifier>`.

Mercator MCF Quick Facts

Format Type Identifier	MCF
Reader/Writer	Both
Licensing Level	Professional
Dependencies	None
Dataset Type	Directory
Feature Type	File name
Typical File Extensions	.dto
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	No
Spatial Index	No
Schema Required	Yes
Transaction Support	No
Geometry Type	mcf_type
Encoding Support	No

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	yes	point	yes
circles	no	polygon	yes
circular arc	no	raster	no
donut polygon	yes	solid	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
elliptical arc	no		surface	no
ellipses	no		text	yes
line	yes		z values	no
none	yes			

Reader Overview

The MCF reader produces FME features for all feature data held in files that reside in a particular directory. For each file, the reader checks to see if that file is requested by looking at the list of **IDS** specified in the mapping file. If a match is made or no **IDS** were specified in the mapping file, then that file is opened to be read. The MCF reader extracts features one at a time and passes them on to the rest of the FME for further processing. When the file is exhausted, the MCF reader starts on the next file. When all MCF files in the directory have been read, then the reader is closed.

MCF uses an upper-left origin for its coordinates and FME uses a bottom-left origin. To compensate for this, all y values are multiplied by -1 to "flip" them over the x-axis.

Reader Directives

The directives processed by the MCF reader are listed below. The suffixes shown are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the MCF reader is **MCF**.

DATASET

Required/Optional: *Required*

The value for this keyword is the directory containing the DTO files to be read. A typical mapping file fragment specifying an input MCF dataset looks like:

```
MCF_DATASET /usr/data/mexico/
```

Workbench Parameter: *Source STAR-APIC Mercator MCF File(s)*

DEF

Required/Optional: *Optional*

The definition specifies only the base name of the file, the type of geometry it contains, and names and types of all attributes. The syntax of an MCF **DEF** line is:

```
<ReaderKeyword>_DEF <baseName> \
    mcf_type mcf_line| \
        mcf_area| \
        mcf_point| \
        mcf_text| \
        mcf_external| \
        mcf_job
    [<attrName> <attrType>]+
```

All MCF data is two-dimensional.

The attribute types created by the MCF format are listed below.

Field Type	Description
char(<width>)	Character fields store fixed-length strings. The width parameter controls the maximum characters

Field Type	Description
	that can be stored by the field. When a character field is written, it is right-padded with blanks, or truncated, to fit the width. When a character field is retrieved, any padding blank characters are stripped away.
decimal(<width>,<decimals>)	Decimal fields store single and double precision floating point values. The width parameter is the total number of characters allocated to the field, including the decimal point. The decimals parameter controls the precision of the data and is the number of digits to the right of the decimal.

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined MCF files to be read. If no **IDs** are specified, then all defined and available files are read. The syntax of the **IDs** keyword is:

```
<ReaderKeyword>_IDs <baseName1> \  
    <baseName2> ... \  
    <baseNameN>
```

The base names must match those used in **DEF** lines.

The example below selects only the **area** and **line DTO** file for input during a translation:

```
MCF_IDS mexico_a \  
    mexico_l
```

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

✳ Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

*** Workbench Parameter**

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

*** Workbench Parameter**

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

*** Workbench Parameter**

Additional Attributes to Expose

Writer Overview

Writer Directives

The directives that are processed by the MCF writer are listed below. The suffixes shown are prefixed by the current `<writerKeyword>_` in a mapping file. By default, the `<writerKeyword>` for the MCF writer is `MCF`.

DATASET

Required/Optional: *Required*

The value for this keyword is the directory where the DTO files are to be written. A typical mapping file fragment specifying an output MCF directory looks like:

```
<writerKeyword>_DATASET /usr/data/mexico/output
```

Workbench Parameter: *Destination STAR-APIC Mercator MCF Directory*

DEF

Required/Optional: *Optional*

The definition specifies only the base name of the file, the type of geometry it contains, and names and types of all attributes. The syntax of an MCF `DEF` line is:

```
<writerKeyword>_DEF <baseName> \
    mcf_type mcf_line| \
        mcf_area| \
        mcf_point| \
        mcf_text| \
        mcf_external| \
        mcf_job
    [<attrName> <attrType>]+
```

The attribute types created by the MCF format are listed below.

Field Type	Description
char(<width>)	Character fields store fixed-length strings. The width parameter controls the maximum characters that can be stored by the field. When a character field is written, it is right-padded with blanks, or truncated, to fit the width. When a character field is retrieved, any padding blank characters are stripped away.
decimal(<width>, <decimals>)	Decimal fields store single and double precision floating point values. The width parameter is the total number of characters allocated to the field, including the decimal point. The decimals parameter controls the precision of the data and is the number of digits to the right of the decimal.

HORIZ_SIZE

Required/Optional: *Optional*

The value for this keyword is the horizontal size of the dataset to be output into the job file. This value is in Mercator units. One Mercator unit is 1/256000 of an inch. If a feature of type `mcf_job` comes into the writer and this keyword is not set, then the job file will have the largest horizontal size from all of the job features. If this keyword is not

specified and there are no input job features, the job file output will have the value shown below (A4 width in Mercator units). The syntax of an MCF HORIZ_SIZE line is:

```
<WriterKeyword>_HORIZ_SIZE 2956651
```

Workbench Parameter: *Horizontal Size [optional]*

VERT_SIZE

Required/Optional: *Optional*

The value for this keyword is the vertical size of the dataset to be output into the job file. This value is in Mercator units. One Mercator unit is 1/256000 of an inch. If a feature of type **mcf_job** comes into the writer and this keyword is not set, then the job file will have the largest vertical size from all of the job features. If this keyword is not specified and there are no input job features, the job file output will have the value shown below (A4 width in Mercator units). The syntax of an MCF VERT_SIZE line is:

```
<WriterKeyword>_VERT_SIZE 2118144
```

Workbench Parameter: *Vertical Size [optional]*

OFFSET_X

Required/Optional: *Optional*

The value for this keyword is the value by which all x-coordinates will be offset. This is a floating point value. For MCF-to-MCF translations, this value is set to 0. If this value is not set, then the writer will calculate its own offset to bring the coordinates to the origin of (0,0). The syntax of an MCF **OFFSET_X** line is:

```
<WriterKeyword>_OFFSET_X 0
```

All MCF data is two-dimensional.

Workbench Parameter: *Offset(X)*

OFFSET_Y

Required/Optional: *Optional*

The value for this keyword is the value by which all y-coordinates will be offset. This is a floating point value. For MCF-to-MCF translations, this value is set to 0. If this value is not set, then the writer will calculate its own offset to bring the coordinates to the origin of (0,0). The syntax of an MCF **OFFSET_Y** line is:

```
<WriterKeyword>_OFFSET_Y 0
```

Workbench Parameter: *Offset(Y)*

SCALE_X

Required/Optional: *Optional*

The value for this keyword is the value by which all x-coordinates will be scaled. This is a floating point value. A value of 1 will leave the x-coordinates of the incoming features as is. For MCF-to-MCF translations this value defaults to 1 but can be changed in the mapping file. If this value is not specified, then the writer calculates its own scale factor that is as large as possible to minimize the loss of precision when decimal coordinates are stored as integers in MCF when writing Version 6.4. The syntax of an MCF **SCALE_X** line is:

```
<WriterKeyword>_SCALE_X 1
```

Workbench Parameter: *Scale(X)*

SCALE_Y

Required/Optional: *Optional*

The value for this keyword is the value by which all y-coordinates will be scaled. This is a floating point value. For MCF to MCF translations this is default to 1 but can be changed in the mapping file. The writer flips the y-coordinates over the x-axis because of MCF's origin. If this value is not specified, then the writer calculates its own scale factor that is as large as possible to minimize the loss of precision when decimal coordinates are stored as integers in MCF when writing version 6.4. The syntax of an MCF **SCALE_Y** line is:

```
<writerKeyword>_SCALE_Y 1
```

Workbench Parameter: *Scale(Y)*

VERSION

Required/Optional: *Optional*

The value for this keyword determines the version of Mercator MCF that is to be output. Version 7.0 allows decimal values, unlike Version 6.4 which allows only integer values. When 7.0 is specified, the default values for **SCALE_X**, **SCALE_Y**, **OFFSET_X**, **OFFSET_Y** are **1, 1, 0, 0** respectively.

Note: It is possible that the version number may be called something other than 7.0.

The syntax of a version line is:

```
<writerKeyword>_VERSION
```

Workbench Parameter: *MCF Version Out*

Feature Representation

MCF features consist of geometry and attribute information. All MCF FME features contain the **mcf_type** attribute that identifies the geometric type.

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Lines

mcf_type: **mcf_line**

MCF line features are two dimensional and represent linear features. These features have the following special attributes associated with them. If there are multiple vectors in a particular group, these vectors will be returned as an aggregate of vectors.

Attribute Name	Contents	Required/Optional
mcf_key	The key value for the feature in the DTO dictionary. If this is not specified, then a default value of <DEFAULT> is used.	Optional
mcf_data_file	The data file where the feature is to be output.	Optional
mcf_group_num	The group number of the feature within the key specified by mcf_key .	Optional

Polygons

mcf_type: **mcf_area**

MCF area features represent closed polygonal features that are two dimensional. These features have the following special attributes associated with them. If there are multiple polygons in a particular group, these polygons will be returned as an aggregate of polygons.

Attribute Name	Contents	Required/Optional
mcf_key	The key value for the feature in the DTO dictionary. If this is not specified, then a default value of <DEFAULT> is used.	Optional
mcf_data_file	The data file where the feature is to be output.	Optional
mcf_group_num	The group number of the feature within the key specified by mcf_key .	Optional

Text

mcf_type: mcf_text

MCF text features hold text information. A single two-dimensional position is associated with the text block. Text features may have the following special attributes associated with them.

Attribute Name	Contents	Required/Optional
mcf_key	The key value for the feature in the DTO dictionary. If this is not specified, then a default value of <DEFAULT> is used.	Optional
mcf_text_string	The text string to be represented.	Required
mcf_rotation	The rotation of the text string in degrees clockwise from the horizontal axis. Rotation in MCF is clockwise from the x-axis, whereas in FME rotation is counter-clockwise from the x-axis. The reader and writer take care of these conversions.	Optional
mcf_scaleY	The size of the text string in the y direction in GRA file units.	Optional
mcf_scaleX	The size of the text string in the x direction in GRA file units.	Optional
mcf_data_file	The data file where the feature is to be output.	Optional
mcf_group_num	The group number of the feature within the key specified by mcf_key .	Optional
mcf_slant	A number defining a slant angle in degrees.	Optional
mcf_kern	The kerning expressed as a percentage of the normal spacing.	Optional
mcf_quad	The quadding mode referring to the posi-	Optional

Attribute Name	Contents	Required/Optional
	tion of the text box that is at the point specified by they x,y point	
mcf_bgfont	The Barco sequence number of the font or string descriptor of other formats.	Optional
mcf_ltext	A Boolean flag to signify if it is a live text object.	Optional
mcf_offset	The offset of the first character from the start of the curve.	Optional
mcf_distance	The orthogonal distance of characters from the curve.	Optional

Points

mcf_type: mcf_extern

MCF external features represent external entity features that are two-dimensional. These features have the following special attributes associated with them.

Attribute Name	Contents	Required/Optional
mcf_key	The key value for the feature in the DTO dictionary.	Optional
mcf_external_file_name	The filename containing the external entity to go at this point.	Required (external only)
mcf_scale1	A scale factor. If this is not specified, a default of 1 is used.	Optional
mcf_rotation	The rotation angle of the external symbol. If this is not specified, a default value of 0 is used. Rotation in MCF is clockwise from the x-axis, whereas in FME, rotation is counterclockwise from the x-axis. The reader and writer take care of these conversions.	Optional
mcf_distortion	The distortion factor. A value of 0.0 means no distortion.	Optional
mcf_scale2	The ratio between the vertical and horizontal scale.	Optional
mcf_data_file	The data file where the feature is to be output.	Optional
mcf_group_num	The group number of the feature within the key specified by mcf_key .	Optional

mcf_type: mcf_point

MCF point features represent point features that are 2D. These features have the following special attributes associated with them.

Attribute Name	Contents	Required/Optional
mcf_key	The key value for the feature in the DTO dictionary.	Optional
mcf_scale1	A scale factor. If this is not specified, a default of 1 is used.	Optional
mcf_rotation	The rotation angle of the point symbol. If this is not specified, a default value of 0 is used. Rotation in MCF is clockwise from the x-axis, whereas in FME rotation is counter-clockwise from the x-axis. The reader and writer take care of these conversions.	Optional
mcf_distortion	The distortion factor. A value of 0.0 means no distortion.	Optional
mcf_scale2	The ratio between the vertical and horizontal scale.	Optional
mcf_data_file	The data file where the feature is to be output.	Optional
mcf_group_num	The group number of the feature within the key specified by mcf_key .	Optional

No_Geometry

mcf_type: mcf_job

MCF job features represent job features that have no geometry. These features have the following special attributes associated with them.

Attribute Name	Contents	Required/Optional
mcf_key	The key value for the feature in the DTO dictionary.	Optional
mcf_vertical_size	The vertical size for the job file in Mercator units.	Required
mcf_horizontal_size	The horizontal size for the job file in Mercator units.	Required
mcf_data_file	The data file where the feature is to be output.	Optional

Swedish KF85 Reader/Writer

The Swedish KF85 Reader and Writer modules allow the Feature Manipulation Engine (FME) to read and write Swedish KF85 files. KF85 is a published ASCII format used by the KF85 product. The *KF85 Reference Manual* describes the KF85 format and all constants it uses.

Overview

KF85 is a two-dimensional (2D) or three-dimensional (3D) system.

KF85 files store both feature geometry and attributes. Both standard attributes and user-defined attributes are supported. All user-defined attributes require integer attribute names. A logical KF85 file consists of one physical file that can have any file extension. The extension **.k85** is added to the base name of the KF85 file when writing files.

The KF85 reader and writer support the storage of point, line, arc, and text geometric data in KF85 files. The KF85 format can also store features with no geometry. Features that have no geometry are referred to as having a geometry of *none*.

The FME automatically recognizes standard KF85 coordinate systems from the header when reading and can convert these projections automatically. When writing KF85, the correct coordinate system string is written to the header and may be overridden with the **COORDINATE_SYSTEM_STRING** directive.

The FME considers a KF85 dataset to be a single KF85 file.

KF85 Quick Facts

Format Type Identifier	KF85
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	File
Feature Type	Geometry type
Typical File Extensions	.k85 (.klf, .kfp, .apt and others)
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	No
Spatial Index	Never
Schema Required	No
Transaction Support	No
Enhanced Geometry	Yes
Geometry Type	kf85_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	yes		polygon	no
circular arc	yes		raster	no
donut polygon	no		solid	no
elliptical arc	yes		surface	no
ellipses	no		text	yes
line	yes		z values	yes
none	yes			

Reader Overview

The KF85 reader opens the input file and immediately starts reading features, returning them to the rest of the FME for processing. The reader does not have any requirement for definition statements as there are no user-defined attributes.

Reader Directives

The directives that are processed by the KF85 reader are listed below. The suffix shown is prefixed by the current <ReaderKeyword> in a mapping file. By default, the <ReaderKeyword> for the KF85 reader is KF85.

DATASET

Required/Optional: *Required*

The value for this directive is the file name of the KF85 file to be read. A typical mapping file fragment specifying an input KF85 dataset looks like:

```
KF85_DATASET /usr/data/KF85/roads.k85
```

Workbench Parameter: *Source Swedish KF85 File(s)*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

Writer Overview

The KF85 writer creates and writes feature data to a KF85 file specified by the DATASET directive. Unlike the reader, the file may or may not exist before the translation occurs. An existing file with the same path and name is overwritten with new feature data. Only one KF85 file may be written to in a single FME session (one translation).

Writer Directives

The directives that are processed by the KF85 writer are listed below. The suffixes shown are prefixed by the current <WriterKeyword> in a mapping file. By default, the <WriterKeyword> for the KF85 writer is KF85.

DATASET

Required/Optional: *Required*

The value for this directive is the file name of the KF85 file to be written. A typical mapping file fragment specifying an output KF85 dataset looks like:

```
KF85_DATASET /usr/data/KF85/roads.k85
```

Workbench Parameter: *Destination Swedish KF85 File*

X_ORIGIN

Required/Optional: *Optional*

The value for this directive is the mathematical X coordinate¹ of the false origin in the destination KF85 file. If this directive is not present in the mapping file, a default value of 0 will be used. A typical mapping file fragment specifying an output KF85 X origin coordinate looks like:

```
KF85_X_ORIGIN 100
```

Y_ORIGIN

Required/Optional: *Optional*

The value for this directive is the mathematical Y coordinate² of the false origin in the destination KF85 file. If this directive is not present in the mapping file, a default value of 0 will be used. A typical mapping file fragment specifying an output KF85 Y origin coordinate looks like:

```
KF85_Y_ORIGIN 300
```

SCALE

Required/Optional: *Optional*

The directive value is the scale of the destination KF85 file. If this directive is not present in the mapping file, a default value of 1 will be used. A typical mapping file fragment specifying an output KF85 scale looks like:

```
KF85_SCALE 500
```

TEXT_LENGTH

Required/Optional: *Optional*

The value for this directive is the maximum length of the text strings appended to the lines and points output to KF85. The maximum value for this directive is 20. If this directive is not present, the default of 20 is used. This does not set the length of text on a **text** feature. A typical mapping file fragment specifying an output KF85 text length looks like:

```
KF85_TEXT_LENGTH 15
```

MANAGER

Required/Optional: *Optional*

The value for this directive sets the **MANAGER** field in the **J** header of the output KF85 file. If this directive is not present, the field is left blank. A typical mapping file fragment specifying an output KF85 manager header field looks like:

```
KF85_MANAGER "BJÖRN"
```

COORDINATE_SYSTEM_STRING

Required/Optional: *Optional*

The value for this directive sets the **coordinate system** field in the **G** header of the output KF85 file, even when using the FME reprojection capabilities. This only overrides the output field and does not affect the data projection. If this directive is not present, the coordinate system string is automatically set to the correct value based on the recognized coordinate system. A typical mapping file fragment specifying an output KF85 coordinate system string looks like:

```
KF85_COORDINATE_SYSTEM_STRING "RT 90 2.5 gon V RH 70"
```

¹This is the **horizontal** component.

²This is the **vertical** component.

Feature Representation

KF85 features consist of geometry and attributes. When reading KF85 files, several predefined attributes hold the data from the file. When writing KF85 files, the values in these predefined attributes are written out to the file. If the feature does not have these predefined attributes, appropriate default values will be used.

Free Attributes or User-Defined Attributes

KF85 files also support free attributes, also known as user-defined attributes. These attributes must have an integer as an attribute name. When a KF85 file is read, each of these attributes is added to the individual features it is associated with. As well, a special attribute `kf85_attribute_list` is attached, whose value is a comma-separated list of all the user-defined attribute names and values present.

When writing KF85 files, all attributes with integer names are interpreted as user-defined attributes and are written out—associated with the appropriate feature. As well, if any feature contains the special attribute `kf85_attribute_list`, this attribute is parsed, looking for attribute name and value pairs. Each of these attributes is written to the KF85 output file as user-defined attributes, associated with the appropriate feature.

Predefined Attributes

All KF85 FME features contain the `kf85_type` attribute to identify the geometric type. All features also have a special KF85 code contained in the `kf85_code` attribute. Depending on the geometric type, the feature contains additional attributes specific to the geometric type. These are described in subsequent sections.

Attribute Name	Contents
<code>kf85_type</code>	The KF85 geometric type of this entity. Required: <code>kf85_point</code> <code>kf85_line</code> <code>kf85_arc</code> <code>kf85_text</code> <code>kf85_common_info</code> <code>kf85_comment</code> <code>kf85_symbol</code> <code>kf85_none</code> Default: NULL
<code>kf85_code</code>	The feature's special code. Required: any 9 digit integer Default: 0
<code>kf85_attribute_list</code>	An alphanumeric comma-separated list of all user-defined attribute names and values present on the feature. Range: maximum 255 character string Default: NULL string

Points

kf85_type: `kf85_point`

A KF85 point feature specifies a single 2D or 3D coordinate and the mean error in both x and y. The point feature may also have text (a label) and/or a symbol associated with the point.

Attribute Name	Contents
kf85_internal_name	An alphanumeric code for the feature. Range: maximum 9 character string Default: NULL string
kf85_mean_error_plane	The mean error in the X coordinate of the point measured in millimetres (mm). Range: 0 ... 32767 Default: 0.0
kf85_mean_error_height	The mean error in the Y coordinate of the point measured in mm. Range: 0 ... 32767 Default: 0.0
kf85_text_string	The optional text string associated with the point. Range: maximum 20 character string Default: NULL string
kf85_rotation	The rotation of the text, measured in degrees counterclockwise from horizontal. (Note that the value actually stored in the file is converted from this into GON.) Range: any real number Default: 0.0
kf85_text_height	The height of the text measured in mm. Range: any real number Default: 1
kf85_text_width	The width of the text measured in mm. Range: any real number Default: 1.0
kf85_text_pos_x	The X coordinate of the point's text. Range: any real number Default: 0.0
kf85_text_pos_y	The Y coordinate of the point's text. Range: any real number Default: 0.0

Attribute Name	Contents
kf85_text_position	The justification of the point's text. Specifically, the point on the text's bounding box where the position is given, as shown below. 0 is an unspecified position. If the value is 0, then the other values relating to the point's text must also be 0. Range: 0 ... 9 Default: 7
kf85_symbol_rotation	The rotation of the symbol, measured in degrees counterclockwise from horizontal. (Note that the value actually stored in the file is converted from this into GON.) Range: any real number Default: 0.0
kf85_symbol_height	The height of the symbol measured in mm. Range: any real number Default: 1.0
kf85_symbol_width	The width of the symbol measured in mm. Range: any real number Default: 1.0

Lines

kf85_type: kf85_line

KF85 line features contains 2D or 3D coordinates, as well as an internal name and a text string. This text also has a position, justification, rotation, width, and height.

When reading a KF85 line; its vertices can be defined in two ways. They can be defined by their x,y (and z) co-ordinates. Or they can be defined by specifying the internal_name of another point in the same file. The KF85 Reader supports both methods. The resulting line is the same regardless of which way it is defined.

Attribute Name	Contents
kf85_internal_name	An alphanumeric code for the feature. Range: maximum 9 character string Default: NULL string
kf85_text_string	The optional text string associated with the line. Range: maximum 20 character string Default: NULL string
kf85_rotation	The rotation of the text, measured in degrees counterclockwise from horizontal. (Note that the value actually stored in the file is converted from this into GON.) Range: any real number Default: 0.0

Attribute Name	Contents
kf85_text_height	The height of the text measured in mm. Range: any real number Default: 1.0
kf85_text_width	The width of the text measured in mm. Range: any real number Default: 1.0
kf85_text_pos_x	The X coordinate of the line's text, relative to the midpoint of the line. Range: any real number Default: 0.0
kf85_text_pos_y	The Y coordinate of the line's text, relative to the midpoint of the line. Range: any real number Default: 0.0
kf85_text_position	The justification of the line's text. Specifically, the point on the text's bounding box where the position is given, as shown below. 0 is an unspecified position. If the value is 0, then the other values relating to the line's text must also be 0. Range: 0 ... 9 Default: 7

Arcs

kf85_type: kf85_arc

Arc features are partial ellipse boundaries with two additional angles that control the portion of the ellipse boundary that is drawn.

Note: The function @Arc() can be used to convert an arc to a linestring. This is useful for storing arcs in systems that do not support them directly.

Attribute Name	Contents
kf85_primary_axis	The length of the semi-major axis in ground units. Range: Any real number > 0 Default: No default
kf85_secondary_axis	The length of the semi-minor axis in ground units. Range: Any real number > 0 Default: No default
kf85_rotation	The rotation of the major axis. The rotation is measured in degrees counterclockwise up from horizontal. Range: -360.0..360.0

Attribute Name	Contents
	Default: 0
kf85_start_angle	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of start_angle. Range: 0.0..360.0 Default: No default
kf85_sweep_angle	Refer to the @Arc (function) in the <i>FME Functions and Factories manual</i> for a detailed definition of sweep_angle. Range: Any real number > 0 Default: No default
kf85_orientation	The orientation of the arc. As the sweep angle is always returned as positive, this field can be used to determine the original orientation of the arc. Range: clockwise counterclockwise Default: none

Text

kf85_type: kf85_text

KF85 text features contain 2D coordinates and a text string, along with the justification, rotation, height, width, font, and distance between lines of the text.

Attribute Name	Contents
kf85_internal_name	An alphanumeric code for the feature. Range: maximum 9 character string Default: NULL string
kf85_text_string	The text string. Range: maximum 62 character string Default: NULL string
kf85_rotation	The rotation of the text, measured in degrees counter-clockwise from horizontal. (Note that the value actually stored in the file is converted from this into GON.) Range: any real number Default: 0.0
kf85_text_height	The height of the text measured in mm. Range: any real number Default: 1.0
kf85_text_width	The width of the text measured in mm. Range: any real number Default: 1.0
kf85_text_position	This is the justification of the line's text. Specifically,

Attribute Name	Contents
	the point on the text's bounding box where the position is given, as shown below. 0 is an unspecified position. If the value is 0, then the other values relating to the line's text must also be 0. Range: 0 ... 9 Default: 7
kf85_text_spacing	The spacing between the text lines. Range: any real number Default: 1.0
kf85_font	The font type for the text. There is no documented standard for this field and, therefore, it must be interpreted by the user. Range: maximum 2 character string Default: NULL string

Common Info

kf85_type: kf85_common_info

KF85 **common info** features result from **common info** records that can occur anywhere within a KF85 file. These features contain no geometry. They are used to store an ID code and string information not associated with any geographic location within the KF85 file.

Attribute Name	Contents
kf85_common_code	The ID code present on the common info . Range: maximum 3 digit integer Default: 0
kf85_common_string	The text string present on the common info . Range: maximum 62 character string Default: NULL string

Comment

kf85_type: kf85_comment

KF85 **comment** features result from **comment** records that can occur anywhere within a KF85 file. These features contain no geometry. They are used to store string information not associated with any geographic location within the KF85 file.

Attribute Name	Contents
kf85_comment_string	The text string present on the comment . Range: maximum 62 character string Default: NULL string

Symbol

kf85_type: kf85_symbol

KF85 symbol features contain 2D coordinates to indicate the location of a symbol. The symbol's height, width, and rotation are also indicated.

Attribute Name	Contents
kf85_rotation	The rotation of the symbol, measured in degrees counterclockwise from horizontal. (Note that the value actually stored in the file is converted from this into GON.) Range: any real number Default: 0.0
kf85_symbol_height	The height of the symbol measured in mm. Range: any real number Default: 1.0
kf85_symbol_width	The width of the symbol measured in mm. Range: any real number Default: 1.0

Swedish Masik Reader/Writer

The Masik reader and writer modules provide the FME with the ability to read and write Swedish Masik files. The Masik file format is an ASCII format used mainly in Sweden.

Overview

Masik data is strictly two-dimensional (2D).

Masik data is stored in a data set that consists of a number of files. A logical Masik data set consists of any number of files in the same directory. There are no conventions concerning file extensions.

The Masik reader and writer support the storage of symbol (point), line, text, and polygon data. Each Masik file can contain either text, symbols, or linework (polygons and lines), and cannot mix these types.

Masik Quick Facts

Format Type Identifier	MASIK
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	Directory
Feature Type	File base name
Typical File Extensions	N/A
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes
Transaction Support	No
Geometry Type	MASIK_GEOMETRY
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	yes

Geometry Support				
Geometry	Supported?		Geometry	Supported?
line	yes		z values	no
none	no			

Reader Overview

The Masik reader first scans the directory it is given for Masik files that have been defined in the mapping file. The Masik reader then extracts features from the files one at a time, and passes them on to the rest of the FME for further processing.

Reader Directives

The directives processed by the Masik reader are listed below. The suffixes shown are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the Masik reader is **MASIK**.

DATASET

Required/Optional: *Required*

The value for this directive is the directory containing the Masik files to be read. A typical mapping file fragment specifying an input Masik data set looks like:

```
MASIK_DATASET /usr/data/cityinfo
```

Workbench Parameter: *Source Swedish MASIK Directory*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

✳ Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

`<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>`

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

`<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]`

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The Masik writer outputs each feature type into a separate file to comply with Masik file regulations. Each feature has the appropriate geometry associated with it – symbols also have an optional symbol type and vectors (lines and polygons) have an optional text attribute.

Writer Directives

The directives that are processed by the Masik writer are listed below. The suffixes shown are prefixed by the current `<WriterKeyword>_` in a mapping file. By default, the `<WriterKeyword>` for the Masik writer is **MASIK**.

DATASET

Required/Optional: *Required*

The value for this keyword is the name of the created Masik directory. If a directory of this name exists, it is replaced by the new Masik data. A typical mapping file fragment specifying an output Masik dataset looks like:

```
MASIK_DATASET /tmp
```

Workbench Parameter: *Destination Swedish MASIK Directory*

DEF

Required/Optional: *Required*

To define files to write features to, the Masik writer uses MASIK_DEF lines. The feature type used in the MASIK_DEF line is used as the file name. The dot character '.' used to separate the basename from the extension must be replaced with an underscore. For example, to read or write using the file name `symbols.fyr`, the feature type on the MASIK_DEF line must be `symbols_fyr`. A typical mapping file fragment specifying an output Masik file in this type of example looks like:

```
MASIK_DEF symbols_fyr masak_symbol_type char(254)
```

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

A special FME feature attribute called MASIK_GEOMETRY directs the Masik writer on how to interpret the feature. The correct values for MASIK_GEOMETRY are `masik_symbol`, `masik_text`, `masik_line`, and `masik_polygon`. Any further parameters specified to each of these four types are described in the following subsections.

Symbols

masik_type: `masik_symbol`

The Masik writer outputs a symbol object containing the point as specified in the input file. Also, the Masik reader and writer associate the symbol type for the object with a specific attribute.

Attribute Name	Contents
<code>masik_symbol_type</code>	A text attribute that specifies the symbol type for the feature. Required: No Default: NULL

Text

masik_type: `masik_text`

The Masik writer outputs a text object containing the point as specified in the input file. The Masik reader and writer also associate the text string for the object with a specific attribute.

Attribute Name	Contents
<code>masik_text_string</code>	A text attribute that specifies a the text string for the feature. Required: No Default: NULL

Lines

masik_type: masak_line

The Masik writer outputs a line object containing the points as specified in the input file. The Masik reader and writer also associate the optional attribute for the object with a specific attribute called **namn**.

Attribute Name	Contents
namn	A text attribute that specifies value of the optional attribute for the feature. Required: No Default: NULL

Polygons

masik_type: masak_polygon

The Masik writer outputs a polygon object containing the points as specified in the input file. The Masik reader and writer also associate the optional attribute for the object with a specific attribute called **namn**.

Attribute Name	Contents
namn	A text attribute that specifies value of the optional attribute for the feature. Required: No Default: NULL

Text File Reader/Writer

Overview

The Text File Reader/Writer allows the Feature Manipulation Engine (FME) to read and write any arbitrary text file one line at a time.

Although it is possible to use this reader and writer in a generic translation, this usually will not produce a useful result by itself. Instead, this format is typically used as part of a customized format or custom workspace.

In such a case, the lines read are parsed using FME transformers into data elements which are further transformed or combined before output is produced. The reader can directly read from gzip file with .txt.gz extension and the writer can write a compressed gzip file if the extension of the output file is .gz.

Text File Quick Facts

Format Type Identifier	TEXTLINE
Reader/Writer	Both
Licensing Level	Base
Dependencies	None
Dataset Type	File
Feature Type	Fixed (text_line)
Typical File Extensions	*.txt, *.txt.gz
Automated Translation Support	Yes
User-Defined Attributes	N/A
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	No
Transaction Support	No
Encoding Support	Yes
Geometry Type	textline_type
End of line type preservation	Yes

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	point	no
circles	no	polygon	no
circular arc	no	raster	no
donut polygon	no	solid	no
elliptical arc	no	surface	no

Geometry Support				
Geometry	Supported?		Geometry	Supported?
ellipses	no		text	no
line	no		z values	n/a
none	yes			

Reader Overview

The Text File reader module produces an FME feature for each line of text in the text file except when READ_WHOLE_FILE_AT_ONCE is specified. In this case, only a single feature containing the `text_line_data` attribute is produced per input file.

Reader Directives

READ_BOTTOM_UP

This optional directive specifies whether the input file should be read backwards, from the end of the file to the top. This feature is useful when only a few features from the end of a large file are required to be read.

Required/Optional

Optional

✳ Workbench Parameter

Read Bottom up

Writer Overview

The Text File writer writes one text line per feature.

Writer Directives

The directives listed below are processed by the Text File writer. The suffixes shown are prefixed by the current `<writerKeyword>` in a mapping file. By default, the `<writerKeyword>` for the Text File writer is `TEXTLINE`.

DATASET

The value for this directive is a text file (*.txt), or a compressed text file (*.txt.gz).

Required/Optional

Required

✳ Workbench Parameter

Destination Text File

OVERWRITE_FILE

This optional directive specifies whether the output file should be overwritten or appended to. The default is for the destination file to be overwritten.

Values

YES | NO

Required/Optional

Optional

*** Workbench Parameter**

Overwrite Existing File

END_OF_LINE

Specifies the format of line terminators to be used for the output file.

Values

Windows | UNIX | Macintosh

Values

YES | NO

Required/Optional

Optional

*** Workbench Parameter**

Line Termination

ENCODING

This optional specification controls which character encoding is used when writing the output file.

If the value is not set, then the character encoding will be automatically detected from the system on which translation is being performed.

Values

UTF-8, UTF-16LE, UTF-16BE, ANSI, BIG5, SJIS, CP437, CP708, CP720, CP737, CP775, CP850, CP852, CP855, CP857, CP860, CP861, CP862, CP863, CP864, CP865, CP866, CP869, CP932, CP936, CP950, CP1250, CP1251, CP1252, CP1253, CP1254, CP1255, CP1256, CP1257, CP1258, ISO8859-1, ISO8859-2, ISO8859-3, ISO8859-4, ISO8859-5, ISO8859-6, ISO8859-7, ISO8859-8, ISO8859-9, ISO8859-13, ISO8859-15

*** Workbench Parameter**

Character Encoding

MIME_TYPE (FME Server use only)

This directive is only of interest to users who are authoring for FME Server's streaming service, and would like to be able to create HTML files on the fly for streaming into the browser (as opposed to streaming plain text back to the browser).

By adjusting this setting, the user is communicating to and storing in the workspace information on how the text file will be streamed back if it is used in the streaming service.

The valid values for this directive are text/plain and text/html.

Required/Optional

Optional

WRITE_UTF8_BOM

This directive specifies whether the byte order mark for UTF-8 encoded file should be written at the beginning. This option applies only when the encoding is set to UTF-8.

Required/Optional

Optional

Values

YES (default) | NO

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), a Text File feature consists of the following attributes:

Attribute Name	Contents
text_line_data	The line of text read from the file.
text_line_length	Total number of bytes in text_line_data attribute. (Reader only)
text_line_number	Line number of the text line within the file. (Reader only)

Trimble JobXML Reader

The Trimble JobXML Reader provides FME with access to data in the JobXML file format. This file format is produced by Trimble Equipment and used mostly for survey work.

Overview

A JobXML file can be divided into three components:

1. **The Environment:** This contains a variety of information about the environment in which the data was created. Some information is important only for the actual hardware (such as the display settings) and some is directly relevant for interpreting the data (such as the coordinate system definition). All portions of the environment are made available as attributes, so even information about the display or job settings is available to the user
2. **The Reductions:** This section contains all the point elements **after** various corrections have been done. Each of these points is constructed with all of its attributes, and a point geometry based on these attributes is created.

IMPORTANT: The coordinate system has not been defined, so the point geometries created here have no associated coordinate system. You can set the coordinate system manually either by selecting from one of FME's predefined coordinate systems, or by creating one in the MyCoordSysDefs.fme file in the FME_HOME/Reproject/ directory. Search www.fmepedia.com for advice on constructing coordinate systems.

3. **The Fieldbook:** This component contains all the recordings that were made during this job. While FME encapsulates each Fieldbook entry into a feature, not all will be of interest. Some track the users' changing of display settings, while others include the raw observation data (UnitRecords and PointRecords, respectively). Based on the XML schema, there are currently 43 possible FieldBook entries, but more may be created in the future. Each FieldBook entry type will be mapped to a distinct feature type.

JobXML Quick Facts

Format Type Identifier	JOBXML
Reader/Writer	Reader
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	n/a
Typical File Extensions	.jxl
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	No
Transaction Support	No
Geometry Type	xml_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	no
circular arc	no		raster	no
donut polygon	no		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	no		z values	n/a
none	yes			

Reader Overview

The JOBXML reader module produces an FME feature for each line in each the data file.

Reader Directives

The directives processed by the JOBXML reader are listed below. The suffixes shown are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the JOBXML reader is **JOJBXML**.

DATASET

Required/Optional: *Required*

An example of the **DATASET** keyword in use is:

```
JOJBXML_DATASET /usr/data/input.jxl
```

Workbench Parameter: *Source Trimble JobXML File(s)*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

✳ Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Feature Representation

JobXML is based on the XML reader, so all features have XML specific format attributes (there are no JobXML format specific attributes).

U.S. Environmental Protection Agency (EPA) Geospatial Data Reader

EPA Geospatial is an XML-based file format produced by the United States Environmental Protection Agency.

More information about this format is available at:

http://www.epa.gov/enviro/geo_data.html

The XML schema file for this format can be located at:

http://www.epa.gov/enviro/html/frs_demo/geospatial_data/EPA_GEODATA_v1.0.xsd

Overview

An EPA Geospatial XML file is divided into two components:

1. **Header:** This section contains some general information about the EPA and the purpose of the XML document, including the date it was produced.
2. **Facility List:** This section contains a list of facilities compiled by the EPA. The information that can be provided for each facility includes: name, address, (including street address, ZIP code, state), electronic address (typically a URL), latitude/longitude and CRS datum name. Additionally, each facility is accompanied by a list of EPA programs that are in effect at the facility.

EPA Geospatial Quick Facts

Format Type Identifier	EPA_GDXML
Reader/Writer	Reader
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	n/a
Typical File Extensions	.xml
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	No
Transaction Support	No
Geometry Type	xml_type

Geometry Support			
Geometry	Supported?	Geometry	Supported?
aggregate	no	point	yes

Geometry Support				
Geometry	Supported?		Geometry	Supported?
circles	no		polygon	no
circular arc	no		raster	no
donut polygon	no		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	no		z values	n/a
none	yes			

Reader Overview

The EPA Geospatial reader module produces an FME feature for each facility in the data file. The header information is ignored by the FME reader.

Reader Directives

The directives processed by the EPA Geospatial XML reader are listed below. The suffixes shown are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the EPA Geospatial XML reader is EPA_GDXML.

DATASET

Required/Optional: *Required*

An example of the **DATASET** keyword in use is:

```
EPA_GDXML_DATASET /usr/data/input.xml
```

Workbench Parameter: *Source EPA Geospatial XML Dataset*

SEARCH_ENVELOPE

This directive specifies a bounding box used to filter the input features. Only features that interact with the bounding box are returned.

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

If all four coordinates of the search envelope are specified as zero, the search envelope will be disabled

Required/Optional

Optional

* Workbench Parameter

Minimum X, Minimum Y, Maximum X, Maximum Y

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Vector Markup Language (VML) Writer

The Vector Markup Language (VML) Format Writer module enables the Feature Manipulation Engine (FME) to be used in conjunction with the world wide web to translate vector data on-the-fly for display in web browsers. VML is an Extensible Markup Language (XML) based exchange, editing, and delivery format for vector graphics on the web. This section assumes familiarity with the VML format and the XML standard.

Overview

VML, which is written using the XML syntax, is a text-based markup language used for describing vector graphics that can be viewed and edited by a wide variety of tools. It provides for the description of lines, polygons, curves, images, and text objects. Positioning and layout of the vector graphics are accomplished by using the Cascading Style Sheets, Level 2 (CSS2) visual rendering model. For more information on VML, XML, CSS1, and CSS2 see the World Wide Web Consortium Web site at <http://www.w3.org>.

The two primary objects that VML describes are the shape and group elements. The shape element is used to define a visible vector graphic element whereas the group element is used to group together several shapes so that they may be transformed together as one unit. These are top-level elements that may define their own local coordinate system. The shape and group coordinate spaces define a CSS2 block level box.

In addition, VML defines several auxiliary top-level elements to help make the editing and representation of graphical information more compact and convenient. These auxiliary elements are the **shapetype** element and the predefined shape elements **line**, **polyline**, **curve**, **rect**, **roundrect**, **oval**, **arc** and **image**. The usage of some of these VML elements currently supported by FME are explained in subsequent sections.

VML Quick Facts

Format Type Identifier	VML
Reader/Writer	Writer
Licensing Level	Base
Dependencies	None
Dataset Type	File
Feature Type	Layer
Typical File Extensions	.vml
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Not applicable
Schema Required	No
Transaction Support	No
Geometry Type	vml_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	yes
line	yes		z values	no
none	no			

Writer Overview

The VML Format writer creates a single VML top-level group element that is used as the containing block for all feature data. This top-level group element defines the coordinate space for all of its sub-elements. The coordinate space for the top-level group element may be specified with the FME **COORDSIZE** and **COORDORIGIN** mapping file directives. These directives are described under the *Writer Directives*. All features are drawn relative to its containing block's — the top-level group element—top left corner. The specified coordinate space has its positive x and y axis in the lower right quadrant.

The VML writer creates a single HyperText Markup Language (HTML) file with the VML data contained in the <body> region of the HTML. At the time of this writing, the only browser supporting VML is Internet Explorer 5 (IE5) and, as a result, the header of the HTML file produced contains information specific to this browser. This information is needed in the header to let other browsers know that the embedded VML data is to be handed off to the browser's VML-specific processor.

Writer Directives

The directives listed below are processed by the VML writer. The suffixes shown are prefixed by the current <WriterKeyword> in a mapping file. By default, the <WriterKeyword> for the VML writer is **VML**.

DATASET

Required/Optional: *Required*

The value for this directive is the name of the VML file to be created. You may want to add the **.html** extension to the file name since the file produced is an HTML file. If a file with this name already exists, then the file will be overwritten. A typical mapping file fragment specifying an output VML data set looks like:

```
VML_DATASET /tmp/outputFile
```

Workbench Parameter: *Destination Vector Markup Language (VML) File*

LEFT

Required/Optional: *Optional*

This directive specifies the left position on the web page for the top-level group element. The top-level group element is the container for all VML features that are drawn.

The syntax for VML **LEFT** is:

```
<WriterKeyword>_LEFT <value> (where <value> is in CSS length units)
```

The default value for this directive is **100pt**.

Note: CSS length values are formed by an optional + or -, followed by a number, followed by a two-letter abbreviation that indicates the unit. There are two types of length units—relative and absolute. Relative length units give a length relative to another length property. The following relative units are available: em—ems, the height of the element's font; ex—x-height, the height of the letter 'x'; and px—pixels, relative to the canvas resolution. Absolute length units are highly dependent on the output medium. The following absolute units are available: in—inches; cm—centimetres; mm—millimetres; pt—points; and pc—picas. For more information on CSS units, please see the CSS1 or CSS2 specifications on the <http://www.w3.org> website.

TOP

Required/Optional: *Optional*

This directive specifies the top position on the web page for the top-level group element.

The syntax for VML **TOP** is:

```
<writerKeyword>_TOP <top> (where <top> is in CSS length units)
```

The default value for this directive is **100pt**.

Note: The values for the **LEFT** and **TOP** directive are written out to be the values for the CSS "left" and "top" style attributes of the top-level group element, respectively. You can modify these two CSS style attributes, with any text editor, to reposition the drawing on the web page. Note that the CSS "left" and "top" style attributes are ignored by web browsers if the CSS *position* style for a shape is not set to *absolute*. By default, the VML writer sets the top-level group CSS *position* style to *absolute*. If this is not suitable, you may use a text editor to change the *position* style attribute to be either *static* or *relative*.

WIDTH

Required/Optional: *Optional*

This directive sets the width, in CSS units, of the containing block for the top-level group element.

The syntax for VML **WIDTH** is:

```
<writerKeyword>_WIDTH <width> (where <width> is in CSS length units)
```

The default value for this directive is **512pt**.

HEIGHT

Required/Optional: *Optional*

This directive sets the height, in CSS units, of the containing block for the top-level group element.

The syntax for VML **HEIGHT** is:

```
<writerKeyword>_HEIGHT <height> (where <height> is in CSS length units)
```

The default value for this directive is **512pt**.

Note: The values for the **WIDTH** and **HEIGHT** directives are written out to be the values for the CSS *width* and *height* style attributes of the top-level group element, respectively. These CSS style attributes for the group may be modified in the translated output file to change the size of the vector drawing on the web page. Changing these does not affect the local coordinate space set by the top-level group element.

COORDSIZE

Required/Optional: *Optional*

This directive defines the number of units along the width and height of the containing block for the top-level group element.

The syntax for VML **COORDSIZE** is:

```
<writerKeyword>_COORDSIZE <width> <height>
```

Note: The value for this directive becomes the value for the VML **coordsize** attribute in the top-level group element. This value should not be modified after the VML output file has been written. Modifying this in the VML output file produces a malformed vector drawing because the position of all elements within the group were calculated based on the original values given by the **COORDSIZE** and **COORDORIGIN** directives.

COORDORIGIN

Required/Optional: *Optional*

This directive defines the coordinate at the top left corner of the containing block for the top-level group element. The positive y-axis is downwards.

The syntax for VML **COORDORIGIN** is:

```
<writerKeyword>_COORDORIGIN <left> <top>
```

Note: The value for this directive becomes the value for the VML **coordorigin** attribute in the top-level group element. The y-axis is inverted; positive is downwards.

SPATIAL_EXTENT

Required/Optional: *Optional*

This directive fixes the spatial extent that the VML output covers, in ground units. The specified spatial extent must be greater than or equal to the minimum bounding rectangle of the feature data. This directive when used in conjunction with the **COORDSIZE** and **COORDORIGIN** directives is useful for maintaining the same VML coordinate space for different output VML files that were translated at different times. Assuming that the **SPATIAL_EXTENT**, **COORDSIZE** and **COORDORIGIN** were kept invariant for all translations, the contents of the output VML files may then be combined into one file by copying and pasting the shapes from the different groups into a single group. If this directive is not specified, then the spatial extent will be set equal to the minimum bounding rectangle of the feature data.

The syntax for VML **SPATIAL_EXTENT** is:

```
<writerKeyword>_SPATIAL_EXTENT <min-x> <min-y> <max-x> <max-y>
```

KEEP_ASPECT_RATIO

Required/Optional: *Optional*

This directive directs the VML writer to maintain the original aspect ratio—determined by spatial extent in ground units—of the input feature data.

The syntax for VML **KEEP_ASPECT_RATIO** is:

```
<writerKeyword>_KEEP_ASPECT_RATIO (YES|NO)
```

The default value for this directive is YES.

PRETTY_PRINT

Required/Optional: *Optional*

This directive gives the option for the VML writer to print the output file in a more attractive format.

The syntax for VML **PRETTY_PRINT** is:

```
<writerKeyword>_PRETTY_PRINT (YES|NO)
```

The default value for this directive is NO.

Note: Enabling this option produces a considerably larger VML output file due to extra blank spaces.

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Special attributes direct the VML writer as it writes the features into the VML file. The most important of these is the **vml_type** attribute which controls the interpretation of the feature. Acceptable values for **vml_type** are **vml_text**, **vml_polyline**, **vml_polygon**, and **vml_point**.

User-defined attributes are ignored by the VML writer. Limited user attribution can be output in the VML file by using the **vml_title** attribute or the **vml_url** attribute allowing a feature to have a URL link which may point to some external data source.

Some VML feature attributes have data type **VML boolean**, **VML number**, or **VML color**. The value for these attributes are copied directly from the FME mapping file to the VML output file, and therefore must conform to the VML specification. The following table lists the VML data type specification for these attributes.

Data Type	Description
VML boolean	An attribute which can take true and false values. The following directives are defined for VML. Value for true: true yes on t 1 Value for false: false no off f 0
VML number	Numeric data used for values that are integer or fractional numbers and for values that specify lengths. Lengths and numbers follow the lexical form defined for CSS with a suffix indicating a scale factor.
VML color	The full set of values are taken from HTML, CSS1, and VML specification. Example: HTML defines the following 16 colours. black silver gray white maroon red purple fuchsia green lime olive yellow navy blue teal aqua Example: Using the CSS1 rgb form "rgb(red,green,blue)" where red, green, and blue are values in the range of 0..255.

The following table lists the attributes that are common to all VML features:

Attribute Name	Contents
vml_color	The color of the brush used to stroke the feature. Range: VML color Default: black

Attribute Name	Contents
vml_weight	<p>The width of the brush used to stroke the feature.</p> <p>Range: VML number</p> <p>Default: 0.75pt</p>
vml_title	<p>The title of the feature that may be displayed by the VML viewer.</p> <p>Range: string</p> <p>Default: None</p>
vml_url	<p>The URL to jump to if this feature is clicked on.</p> <p>Range: string</p> <p>Default: None</p>
vml_target	<p>The target frame in an URL.</p> <p>Range: string</p> <p>Default: None</p>
vml_z_index	<p>The z-index of the feature in the output VML file. Positive numbers are in front of the screen. Negative numbers are behind the screen. Features having a higher z-index obscure features with lower z-index.</p> <p>Range: integer</p> <p>Default:</p> <p>0 (for vml_polygon features)</p> <p>10 (for vml_polyline features)</p> <p>11 (for vml_point features)</p> <p>12 (for vml_text features)</p>
vml_fill_color	<p>The color of the brush used to fill the feature. This attribute is not applicable for vml_polyline features.</p> <p>Range: VML color</p> <p>Default:</p> <p>black for points and text</p> <p>No Default for polygons</p>
vml_fill_attr{#}	<p>This list attribute allows the feature to be filled with customized effects. If a feature has this list attribute the VML element that represents the feature in the output file will contain a VML fill sub-element. The contents of the vml_fill_attr{#} list attribute must be of the following form: <attribute-name>=<attribute-value></p> <p>Where <attribute-name> is a name of an attribute for the VML fill sub-element, and <attribute-value> is one of the possible values for that attribute name. Please refer to the VML specification for all the possible attributes that the VML fill sub-element may contain. The VML specification may be found at http://www.w3.org.</p>

Attribute Name	Contents
	<p>For example, to specify that a feature is filled with a gradient and blue color add the following vml_fill_attr{#} list attribute with the following values to the feature:</p> <pre>vml_fill_attr{0} "type=gradient" vml_fill_attr{1} "color=blue"</pre> <p>NOTE: If the above string values contain spaces then they must be enclosed between double quotes.</p> <p>Also note that the index for the vml_fill_attr{0} must start from 0. The order in which the attributes are listed in the list attribute is of no importance.</p> <p>Range: <attribute-name>=<attribute-value> as described above</p> <p>Default: No Default</p>
vml_stroke_attr{#}	<p>This list attribute allows the feature to be render with a customized outline. The values for this list attribute is similar to the vml_fill_attr{#} described above with the values for the <attribute-name> and <attribute-value> taken from the VML stroke sub-element instead.</p> <p>Example, to specify that a feature should be render with a dotted blue line add the following vml_stroke_attr{#} list attribute with the following values to the feature:</p> <pre>vml_stroke_attr{0} "color=blue" vml_stroke_attr{1} "dashstyle=dot"</pre> <p>Range: <attribute-name>=<attribute-value></p> <p>Default: No Default</p>
vml_shadow_attr{#}	<p>This list attribute allows the feature to be render with a shadow effect. The values for this list attribute is similar to the vml_fill_attr{#} described above with the values for the <attribute-name> and <attribute-value> taken from the VML shadow sub-element instead.</p> <p>Example, to specify that a feature should be render with a dotted blue line add the following list attribute with the following values to the feature:</p> <pre>vml_shadow_attr{0} "on=true" vml_shadow_attr{1} "type=perspective"</pre> <p>Range: <attribute-name>=<attribute-value></p> <p>Default: No Default</p>
vml_imagedata_attr{#}	<p>This list attribute allows the feature to have a picture render on top of it. The values for this list attribute is similar to the vml_fill_attr{#} described above with the values for the <attribute-name> and <attribute-value> taken</p>

Attribute Name	Contents
	<p>from the VML imagedata sub-element instead.</p> <p>Example, to specify that a feature should be rendered with the sample 's.jpg' image on top add the vml_image_data_attr{#} list attribute with the following values to the feature:</p> <pre>vml_imagedata_attr{0} "src=c:\temp\s.jpg"</pre> <p>Range: <attribute-name>=<attribute-value></p> <p>Default: No Default</p>

Points

vml_type: vml_point

Point features must have exactly one coordinate. The VML writer uses the predefined VML oval element to generate round point features. Point features have their vml_fill_color set to black by default.

VML point features have the following additional attribute:

Attribute Name	Contents
vml_point_size	<p>The size of the point in ground units.</p> <p>Range: real>0</p> <p>Default: x, where $x=0.006*\text{deltaY}$. deltaY is the y-coordinate range of the spatial extent of the input data in ground units.</p>

Polylines

vml_type: vml_polyline

Polyline features must have at least two coordinates. The VML writer writes out a vml_polyline feature in the VML output file as a predefined VML polyline element. The writer also writes out the z-index attribute to 10 so that, by default, the polylines only obscure the polygons.

Polygon

vml_type: vml_polygon

Polygon features must have at least four coordinates, with the last coordinate equal to the first coordinate. The vml_polygon features may not contain holes. The VML writer writes out a vml_polygon feature in the output file as a predefined VML polyline element. By default, the VML writer does not set the z-index attribute. When it's not set, the web browser interprets the polygon—the VML filled polyline element—to have a z-index of 0. Polygons produced by the VML writer, therefore, by default do not obscure other objects with the exception of overlapping polygons with the same z-index value. Objects drawn later with equal z-indexes obscure earlier ones.

Text

vml_type: vml_text

Text features must have exactly one coordinate. The vml_text_string attribute must also be present in the vml_text feature. Text is drawn in the output file by placing a VML textpath sub-element inside of a VML shape element. By default, vml_text features have their vml_fill_color attribute set to black.

VML text features have the following additional attributes:

Attribute Name	Contents
vml_text_string	<p>The text string may contain blanks and there is no limit on its length. This attribute must be present for all vml_text features.</p> <p>Range: string Default: None</p>
vml_text_size	<p>The size of the text in ground units.</p> <p>Range: real>0 Default: x, where $x=0.008*\text{deltaY}$. deltaY is the y-coordinate range of the spatial extent of the input data in ground units.</p>
vml_text_justification	<p>The justification of the text.</p> <p>Range: left center right Default: left</p>
vml_rotation	<p>The rotation of the text, as measured in degrees counterclockwise from the horizontal.</p> <p>Range: -360.0...360.0 Default: 0</p>
vml_font_family	<p>The CSS1 font family name.</p> <p>Range: CSS1 font family name. Default: Times New Roman</p>
vml_font_style	<p>The style of the font.</p> <p>Range: normal italic oblique Default: normal</p>
vml_font_weight	<p>The weight of the font.</p> <p>Range: normal bold bolder lighter 100 200 300 400 500 600 700 800 900 Default: normal</p>
vml_rotate_letters	<p>Rotate the letters of the text by 90 degrees.</p> <p>Range: VML boolean Default: f</p>
vml_same_letter_heights	<p>Stretches lowercase letters to the height of uppercase letters.</p> <p>Range: VML boolean Default: f</p>

Vector Product Format (VPF) Coverage and Database Reader/Writer

Format Notes:

This format is not supported by FME Base Edition.

The VPF¹ Writer is an extra-cost format available from Safe Software.

Overview

The Vector Product Format (VPF™) is a standard format, structure, and organization for large geographic databases. VPF data is stored in a structure described in the *Military Standard, Vector Product Format, MIL-STD-2407*. The Standard specifies the structure for directories, tables, table columns, table join relationships, and media exchange conventions for all VPF data. While the Standard describes the structure, it does not describe the contents of a set of VPF data; this is the role of "VPF Product Specifications."

The VPF Reader module provides the Feature Manipulation engine (FME) with access to data in any of the number of formats that follow the VPF specification. This includes, but is not limited to, data that adheres to the Digital Chart of the World (DCW), Digital Nautical Chart (DNC), VMap Level 0, VMap Level 1, VMap Level 2, and UVMAP database standards.

The VPF Writer enables FME to write to VPF product database. Product database types include Vector Smart Map Level 0, 1 and 2 (VMap0, VMap1, VMap2); Digital Nautical Chart (DNC), Urban Vector Map (UVMAP); and Foundation Feature Data (FFD).

¹VPF is a trademark of the National Geospatial-Intelligence Agency.

VPF Reader Quick Facts

Format Type Identifier	VPF, VPF_DB (see note*)
Reader/Writer	Reader
Licensing Level	Professional
Dependencies	None
Dataset Type	Directory or File
Feature Type	Feature class name Metadata table name
Typical File Extensions	*.*ft, dht (see note*)
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	None
Schema Required	Not applicable
Transaction Support	No
Geometry Type	vpf_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	yes
line	yes		z values	no
none	no			

Note: The VPF reader is capable of reading a VPF dataset from the database level or the coverage level. The keyword VPF invokes the reader in the coverage mode; the keyword VPF_DB invokes the reader in database mode. The typical file extension for the reader in coverage mode is *.*ft, and the typical file extension for the reader in database mode is dht.

VPF Writer Quick Facts

Format Type Identifier	VPF_DB
Reader/Writer	Writer
Licensing Level	Professional
Dependencies	Extra-cost plug-in required
Dataset Type	Directory
Feature Type	N/A
Typical File Extensions	N/A
Automated Translation Support	No
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	No
Spatial Index	No
Schema Required	No
Transaction Support	No
Geometry Type	vpf_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	
donut polygon	no		solid	no
elliptical arc	no		surface	no
ellipses	no		text	yes
line	yes		z values	yes
none	yes			

Reader Overview

The VPF reader module produces FME features for metadata and feature class features in a VPF database.

VPF databases are implemented using a hierarchy of file system directories. The root of this hierarchy is the database directory, which contains a subdirectory for each library in the database. Each library contains a subdirectory for every coverage in that library. Coverages contain one or more feature classes, and these feature classes are implemented as feature table files (*.ft).

When the reader is invoked in database mode (VPF_DB), it examines feature classes in every coverage of every library in a database and produces an FME feature for each row in each feature table file. The reader also examines the metadata tables available at each level in the hierarchy and produces non-geometric metadata features that contain the table's data.

When the reader is invoked in coverage mode (**VPF**), the set of features produced is restricted to only those feature classes in the input coverage directory.

The feature set produced by the reader can be further restricted by specifying a subset of feature classes. This subset can be specified using either a list of **IDS** or multiple **DEF** line entries. The specification of **DEF** or **ID** parameters restricts the feature classes that the reader will open and produce FME features for. Note that if both the **DEF** and **ID** parameters are specified for an instance of a reader, then the intersection of these two sets will be used as the restriction set.

The tiles that the reader produces for its feature set can be restricted by specifying a tile subset value using the **TILES** parameter.

Reader Directives

This section describes the directives that are recognized by the VPF reader. Each directive is prefixed by the current **<ReaderKeyword>_** when placed in a mapping file. By default, the **<ReaderKeyword>** for the VPF reader is **VPF**.

DATASET

Required/Optional: *Required*

In database mode (**VPF_DB**), the value for this directive is the path to the DHT file. The directory where this file exists is the root directory of the VPF database. This directory directly and indirectly contains all the libraries, coverages, feature classes and related metadata for the database. A typical mapping file fragment that selects a VMap database from drive **e:** would be:

```
VPF_DATASET e:/vmap1v0/dht
```

In coverage mode (**VPF**), the value for this directive defines the coverage directory that the reader will read from. A typical mapping file fragment that selects a VMap database from drive 'e:' would be:

```
VPF_DATASET e:/vmap1v0/noamer/hydro
```

Workbench Parameter: *Source Vector Product Format Database File(s)*

READ_UPPER_CASE

Required/Optional: *Optional*

If set to **YES**, all the attributes are read in uppercase; otherwise, the attributes are read in lowercase.

TILES

Required/Optional: *Optional*

This optional specification is used to limit geometric features that are produced. Only feature class features that exist in the specified tiles will be output. If no value is specified, then all the tiles in the library will be used. If a tiled subset is specified for a nontiled library, then this specification is ignored.

The syntax for the **TILES** directive used in a database mode reader (**VPF_DB**) is:

```
<ReaderKeyword> _TILES \  
  <libName0>{ <tileId0>, <tileId1>,...}\  
  ...\  
  <libNameN>{ <tileId0>, <tileId1>,...}
```

Note that you must specify **libName**.

Example:

```
VPF _DB_TILES H1316010{30} A1316080{50, 30, 31, 32, 33, 34}  
VPF_DB_TILES EURNASIA{1-5}
```

An option to listing a sequence of tile IDs is to specify a range of tiles. The syntax for a tile range is:

```
<tileId0>-<tileIdN>
```

The following example, using a tile range, is semantically equivalent to the previous example:

```
VPF_DB _TILES H1316010{30} A1316080{50, 30-34}
```

Note: In Database Mode reader (VPF_DB), if no IDs or DEFs are specified then tiles specified will only apply to coverages that are tiled. If a coverage is not tiled, then all the features from that coverage will be returned.

The syntax for the **TILES** directive used in a Coverage Mode reader (VPF) is:

```
<ReaderKeyword> _TILES \  
  <tileId0>, <tileId1>, ...<tileIdN>
```

Note that this syntax is different from the one in DB Mode as it does not require that tile IDs be contained within curly braces. An example that selects tiles 30, 31, 32, 33, 34 and 50 :

```
VPF _TILES 50,30-34
```

Workbench Parameter: *Tile IDs*

TILE_EXTENTS

Required/Optional: *Optional*

This option specifies the regional extents of the tiled subsection. Note that this directive is ignored if the data is not tiled.

```
<ReaderKeyword>_TILE_EXTENTS [xmin, ymin, xmax, ymax]
```

Workbench Parameter: *Tile Extents*

DEF

Required/Optional: *Optional*

This optional specification is used to limit the available feature classes that are read. A feature class definition specifies the name of the feature class and the library and coverage where the class is located. Feature attribute information is ignored by the reader. The syntax for a **DEF** line used in a database mode reader (VPF_DB) is:

```
<ReaderKeyword>_DEF <libName>\<coverageName>\<featClass>
```

The syntax for a **DEF** line used in a coverage mode reader (VPF) is:

```
<ReaderKeyword>_DEF <featClass>
```

Any additional declarations in the **DEF** line parameter are ignored by the reader.

Note: If both DEF lines and IDs are used to specify feature class, then the intersect of these sets determines the actual feature classes to read.

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available feature classes that are read. If no **IDs** are specified, then all available feature classes are read. The syntax of the **IDs** directive in a database mode reader (VPF_DB) is:

```
<ReaderKeyword>_IDS <libName>\<coverageName>\<featClass> \  
  ...\  
  <libNameM>\<coverageNameN>\<featClassO>
```

The syntax for a **DEF** line used in a coverage mode reader (VPF) is:

```
<ReaderKeyword>_IDS <featClass1> \  
  ... \  
  <featClassN>
```

Note: If both DEF lines and IDs are used to specify feature class, then the intersect of these sets determines the actual feature classes to read.

FEAT_TYPE_SEP

Required/Optional: *Optional*

The value for this parameter is the character that is used in a feature's feature type to separate the library and coverage names from the metadata table and feature class names. If this attribute is not specified, then the '\' separator is used.

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the `mitab.dll` in the FME home directory to `mapinfo.dll`.

The syntax of the `MAPINFO_SEARCH_ENVELOPE` directive is:

```
MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

```
MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The `COORDINATE_SYSTEM` directive, which specifies the coordinate system associated with the data to be read, must always be set if the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` to the reader `COORDINATE_SYSTEM` prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the `SEARCH_ENVELOPE` directive.

Values

YES | NO (default)

Mapping File Syntax

<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Feature Representation (VPF Reader)

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

The VPF reader produces FME features for VPF feature class features as well as VPF metadata table entries when a reader is invoked in database mode (**VPF_DB**).

The VPF reader produces FME features for VPF feature class features as well as VPF metadata table entries when a reader is invoked in database mode (**VPF_DB**).

There are many types of FME features produced for metadata at every level of a VPF database.

At the root level of the database, FME features are produced for the Database Header Table (DHT); the Library Attribute Table (LAT); and, if exists, the Data Quality Table (DQT).

At the library level, FME features are produced for the Coverage Attribute Table (CAT); the Library Header Table (LHT); the Geographic Reference Table (GRT) and, if they exist, the Data Quality(DQT) and Lineage Document (LINEAGE.DOC) Tables.

At the coverage level, FME features are produced for: the Character and Integer Value Description Tables (CHAR.VDT) and (INT.VDT); the Symbol and Notes Related Attribute Tables (SYMBOL.RAT) and (NOTES.RAT); the Feature Class Attribute and Schema Tables (FCA) and (FCS); the Data Quality Table (DQT); the coverage document table (<coverage name>.DOC); and feature class document tables (<feature class name>.DOC).

With the exception of the feature class schema table (FCS), all coverage metadata tables are optional and FME features are only produced for metatables that exist.

A FME feature is produced for each row in each metadata table that is processed. For each column in a metadata table, there is one attribute in an FME feature with the same name and type as the column. The value of each attribute will be the same as the row from which the feature was produced. No geometric information is attached to features that are produced from metadata tables.

The other type of FME features that this reader produces result from VPF feature class features. This type of feature can have zero or more geometries attached to it. For this reason, all of the FME features produced for these VPF features contain an aggregate of geometries. For most feature class geometry types, all geometries in the resulting features are the same type—the type of the table. For complex feature classes, a single feature can contain any mixture of text, point, line, and area features.

The attributes for each geometry are given by the attribute `component{<n>}.<attrName>`, where `<n>` is the position of the geometry in question with the first one being at position 0, and `<attrName>` is the geometry-specific attribute in question.

The attributes defined for each geometry are totaled in the following table.

Attribute Name	Description	Defined On
<code>vpf_type</code>	The type of this specific geometry. This has one of the following values: <code>vpf_area</code> , <code>vpf_line</code> , <code>vpf_point</code> , or <code>vpf_text</code> .	All geometries
<code>vpf_database_name</code>	The name of the database the feature belongs to.	
<code>vpf_library_name</code>	The name of the library the feature belongs to.	
<code>vpf_coverage_name</code>	The name of the coverage the feature belongs to.	
<code>vpf_feature_class</code>	The name of the feature class the feature belongs to.	
<code>vpf_tile_name</code>	The name of the tile the feature lies in. The <code>tile_name</code> also depicts relative tile path.	
<code>vpf_containing_face</code>	This attribute is present for entity nodes for level 3 topology. It contains the <code>face_id</code> of the area containing the node.	
<code>vpf_first_edge</code>	This attribute contains the <code>edge_id</code> of the <code>first_edge</code> of a connected node. This attribute will be present for level 1 and higher topology	
<code>component{#}.attributeName</code> e.g., <code>component{#}.vpf_type</code>	List attributes of the parts of the geometry and defines the geometry-specific attributes on the feature. For example, <code>component{0}.vpf_type</code> means the <code>vpf_type</code> of the first component of the feature.	
<code>vpf_text_string</code>	The string to be displayed for a <code>vpf_text</code> geometry.	<code>vpf_text</code>

Attribute Name	Description	Defined On
vpf_text_height	The height of a vpf_text geometry. This is automatically extracted from the SYMBOL_RAT{0}.SIZE attribute. Although the VPF standard does not explicitly specify this attribute it is used to preserve the point size of the text feature when specified. It is used in combination with vpf_text_scale to calculate the text height in ground units stored in fme_text_size . This attribute will be automatically deleted in case the fme_text_size is modified. (See the information under the heading <i>Special Attribute Handling</i> .)	vpf_text
vpf_text_font	The font used to display vpf_text geometry. This is automatically extracted from the SYMBOL_RAT{0}.FONT or SYMBOL_RAT{0}.FON attribute. (See the information under the heading <i>Special Attribute Handling</i> .)	vpf_text
vpf_text_color	The color used to display vpf_text geometry. This is automatically extracted from the SYMBOL_RAT{0}.COLOR or SYMBOL_RAT{0}.COL attribute. (See the information under the heading <i>Special Attribute Handling</i> .)	vpf_text
vpf_text_style	The style used to display vpf_text geometry. This is automatically extracted from the SYMBOL_RAT{0}.STYLE or SYMBOL_RAT{0}.STY attribute. (See the information under the heading <i>Special Attribute Handling</i> .)	vpf_text
vpf_rotation	The rotation at which the text is to be displayed. This is calculated from the lower left and lower right coordinates of the text line, and is expressed in degrees counterclockwise from due east. It defaults to 0.0 if the text geometry has only one coordinate in the VPF data.	vpf_text
vpf_text_scale	The scale read from the Library Header Table (LHT) in the library containing the feature. A default scale of 0, 1000000 will be used if no scale is specified. Scale	vpf_text

Attribute Name	Description	Defined On
	is used in combination with <code>vpf_text_height</code> to calculate the <code>fme_text_size</code> .	
<code>vpf_original_geometry</code>	<p>Optional. The original geometry as read by the VPF Reader in the Well Known Text (WKT) format for when text geometries are not a simple single point. Used to preserve the original text geometry. This attribute will be deleted if <code>fme_rotation</code> is modified.</p> <p>This attribute is only present if the shape line contains more than one coordinate pair.</p> <p>The order of coordinate pairs follows the specs found in VPF: the first coordinate pair represents the lower left coordinate, and the second coordinate pair defines the lower right of the string. Third and subsequent coordinate pairs define control points in a shape line, i.e the points in between the first and second pairs.</p>	<code>vpf_text</code>
<code>vpf_sequenced_geometry</code>	<p>Optional. It is similar to <code>vpf_original_geometry</code> except for the order of the coordinate pairs. The list of coordinate pairs in this attribute is sequenced, i.e., the second coordinate pair in <code>vpf_original_geometry</code> is moved to the end of the list in <code>vpf_sequenced_geometry</code>. This attribute is only present if the shape line contains more than one coordinate pair.</p>	<code>vpf_text</code>

The following example lists the attributes that appear on a feature within a complex feature class, containing both a `line` and a `text` attribute:

```

ID 234
NOAMER\TRANS\ROADL MR001
component{0}.vpf_type vpf_line
component{1}.vpf_type vpf_text
component{1}.vpf_text_string "23rd Street"
component{1}.vpf_rotation 14.012
component{1}.vpf_text_font 12
component{1}.vpf_text_color 3
component{1}.vpf_text_style 5
component{1}.vpf_text_height 14.12

```

The features read from the feature class are normally pushed through a `DeaggregateFactory` to extract the individual geometries. This is defined as:

```

FACTORY_DEF VPF DeaggregateFactory \
    INPUT FEATURE_TYPE * fme_geometry fme_aggregate \
    LIST_NAME component{} \
    OUTPUT LINE FEATURE_TYPE * \
    OUTPUT DONUT FEATURE_TYPE * \
    OUTPUT POINT FEATURE_TYPE * \
    OUTPUT POLYGON FEATURE_TYPE *

```

FME Text Feature Creation

FME and VPF represent text differently. FME classic geometry allows for only single point geometries on text features and additionally can supply a text string, rotation and text size (height in ground units). VPF represents text primitive geometries as lines (one or more points, first being lower left corner and last being lower right corner) and an optional text size (height in typographical point size, as defined in the SYMBOL.RAT table).

The FME point geometry is simply extracted from the first point of the VPF line geometry, both of them being located at the lower left bound of the string.

The FME rotation, in degrees, is calculated from the first and the last coordinate of the VPF line geometry. If only one point is present, it defaults to zero.

The FME text size, in ground units, is calculated from the VPF text size, in typographical point size. In the special case of a VPF text primitive with only one point and no VPF text size, the conversion will be done using a default VPF text size of 2. Otherwise, in case the VPF text size is not present, the FME text size is approximated by using the width of the string in ground units and the approximation that the height of a character in ground units is half its size.

To avoid inconsistencies between VPF specific attributes and FME generic ones, the FME text size is entangled with the VPF text height and the FME rotation is entangled with the VPF original geometry. Attempts to modify the FME generic attribute will delete the VPF specific attribute. However, modification of the VPF specific attributes will not delete the FME generic attribute in order to maintain a valid FME feature while allowing the possibility to force the reader to use certain VPF specific attribute values.

Feature Type Values

When a reader is invoked in database mode (**VPF_DB**), all FME feature types are scoped to reflect their level in the database hierarchy. As an example, an FME feature produced from a row in a Library Header Table (LHT) will have a feature type that contains both the local name **LHT** and the library name where the table is found. FME features produced from a **LHT** metadata file in the **NOAMER** library will have a feature type of **NOAMER\LHT**.

Special Attribute Handling

The structuring of VPF data allows for a very expressive schema definition, which is somewhat difficult to capture using traditional typing information. The following variations, from a **flat** table structure, are of particular interest.

- VPF may contain attributes that are arrays of two-dimensional (2D) and three-dimensional (3D) coordinates.
- Any integer or text attribute in a VPF table may be associated with a value descriptor table, giving a more verbose textual description of the attribute.
- Feature classes in VPF are defined by joining various tables together, leading to a hierarchy of attribute values.
- Text primitives do not themselves contain any colour, style, size, or font information, but the features often define these attributes by relating in a symbology table, such as **SYMBOL.RAT**.

Value Descriptor Tables

VPF coverages typically contain two value descriptor tables, **INT.VDT** and **CHAR.VDT**. It is possible for any number of ***.VDT** tables, with any names, but these are typical. The purpose of the Value Descriptor Table (VDT) is to define, for each feature class-attribute name pair using the VDT, a mapping of an integer or short text string with limited number of values, and a longer text description of the attribute value.

For example, VDTs are used to assign an English description to a feature code. An airport might have a feature code attribute of **AF001** meaning an International Air Field. The feature table for the feature would contain a code of **AF001** and a reference to the value descriptor table. The value descriptor table then provides the mapping from **AF001** to International Air Field.

The lookups into value descriptor tables are handled automatically by the VPF reader. An attribute named **<attrName>** corresponding to an entry in a VDT results in two attributes being defined on the FME feature: **<attrName>** and **<attrName>desc**.

In the above example, the feature code would be defined in an attribute such as **FCOD** in the feature table. The resulting FME feature would contain two attributes for this:

```
FCOD    "AF001"
FCODdesc "International Air Field"
```

Table Relations

Every VPF feature class contains a feature table to define the attributes appearing on features of that class. It is also possible for a VPF feature class to include related attributes from another table, by specifying that a particular column of the feature class table relates to a primary column of another table.

Consider, for example, a mythical database which includes a list of the families of a street and if they have a fire hydrant on their lawn. This database also allows for many families to reside at one address. The feature class is defined as follows:

- The feature table contains a special column, **STRADDR.RAT_ID**, used to join to the identifier column of the **STRADDR.RAT** table.
- The **STRADDR.RAT** table defines the street addresses and contains a column, **OCCUPANT.RAT_ID**, that links into the **OCCUPANT.RAT** table.

The schema for the street feature class is something like:

```
Class name: STREETL
Geometry type: vpf_line
Feature table: STREETL.LFT
```

Table **STREETL.LFT** attributes:

```
ID int (Row identifier)
STRADDR.RAT_ID int (Link to STRADDR.RAT table.)
EDG_ID int (Link to edge primitive table.)
STRCODE int (Code for street name; refers to a value lookup in the
              INT.VDT table.)
```

Table **STRADDR.RAT** attributes:

```
ID int (Row identifier)
ADDRESS int (Street number)
HYDRANT char(3) (Does it have a hydrant on its lawn; 'Yes' or 'No ').
OCCUPANT int (Link to "occupants" table.)
```

Table **OCCUPANT.RAT** attributes:

```
ID int (Row identifier)
FAM_NAME text(40) (Family name)
NUM_RES int (Number of residents)
CLUST_ID int (Identifies "cluster" of occupants.)
```

The feature class table relates the tables according to the following structure:

Table	Foreign Attribute	Related Table	Join Attribute
STREET.LFT	EDG_ID	EDG	ID
STREETL.LFT	STRADDR.RAT_ID	STRADDR.RAT	ID
STRADDR.RAT	OCCUPANT	OCCUPANT.RAT	CLUST_ID

A small cul-de-sac containing two properties, one of which has two families living in it, might be represented in the **STREETL** feature class with the following feature. Note the use of nested lists and enhanced values from VDTs. The **HYDRANT** attribute would normally come from a value attribute table as well.

```

Feature class: STREETL
Geometry: Aggregate containing one line
Attributes on feature:
ID 12
STRADDR.RAT_ID 5
EDG_ID 19
STRCODE 5
STRCODEdesc "29th Avenue"
STRADDR_RAT{0}.ID 5
STRADDR_RAT{0}.ADDRESS 1234
STRADDR_RAT{0}.HYDRANT "Yes"
STRADDR_RAT{0}.OCCUPANT 23
STRADDR_RAT{0}.OCCUPANT{0}.ID 1
STRADDR_RAT{0}.OCCUPANT{0}.FAM_NAME "Smith"
STRADDR_RAT{0}.OCCUPANT{0}.NUM_RES 4
STRADDR_RAT{0}.OCCUPANT{0}.CLUST_ID 23
STRADDR_RAT{0}.OCCUPANT{1}.ID 2
STRADDR_RAT{0}.OCCUPANT{1}.FAM_NAME "Jones"
STRADDR_RAT{0}.OCCUPANT{1}.NUM_RES 2
STRADDR_RAT{0}.OCCUPANT{1}.CLUST_ID 23
STRADDR_RAT{0}.ID 6
STRADDR_RAT{0}.ADDRESS 2345
STRADDR_RAT{0}.HYDRANT "No"
STRADDR_RAT{0}.OCCUPANT 14
STRADDR_RAT{1}.OCCUPANT{0}.ID 3
STRADDR_RAT{1}.OCCUPANT{0}.FAM_NAME "Murray"
STRADDR_RAT{1}.OCCUPANT{0}.NUM_RES 2
STRADDR_RAT{1}.OCCUPANT{0}.CLUST_ID 14

```

Note: It is important to note that the author is not certain that the VPF standard allows databases to be structured in this way. However, the VPF reader interprets such structures and it provides a reasonable example to explain how related attribute tables are expressed in FME features. If the VPF standard calls for a flatter structure, this attribute naming scheme still applies.

Text Primitive Attributes

The display attributes one expects to find for text strings – colour, size, style and font – are not actually defined anywhere in the VPF specification.

They are usually handled by assigning a **symbology identifier** attribute to the text feature classes' feature table, and relating this **ID** to a symbology attribute table, typically **SYMBOL.RAT**. The VPF reader currently defines these text properties according to the values in the **SYMBOL.RAT** table. The following attributes are currently being used to define these values and are searched in the specified order, and the first value found is taken as the value.

Text Attribute	Attributed Symbology Used to Define It
text_color	SYMBOL_RAT{0}.COLOR, SYMBOL_RAT{0}.COL
text_height	SYMBOL_RAT{0}.SIZE, SYMBOL_RAT{0}.SIZ
text_font	SYMBOL_RAT{0}.FONT, SYMBOL_RAT{0}.FON
text_style	SYMBOL_RAT{0}.STYLE, SYMBOL_RAT{0}.STY

Writer Overview

The VPF writer converts a set of FME features into a VPF database. **DEF** lines defined in a writer's mapping file are ignored since all feature class user attributes are statically defined by its product specification.

Writer Directives

This section describes the directives processed by the VPF writer module. Each of the directives is prefixed by the current **<writerkeyword>**_ when they are placed in a mapping file. By default, the **<writerkeyword>** for the VPF writer is **VPF_DB**.

DATASET

Required/Optional: *Required*

The value for this parameter is the path to where the output databases's root directory is created. A typical mapping file fragment is:

```
VPF_DATASET /vpf_output/my_vmap0/
```

Workbench Parameter: *Destination Vector Product Format Database (VPF_DB) Directory*

FEAT_TYPE_SEP

Required/Optional: *Optional*

The value for this parameter is the character that is used in a feature's feature type to separate the library and coverage names from the metadata table and feature class names. If this attribute is not specified, then the '\' separator is assumed.

Workbench Parameter: *VPF Feature Class Separator*

LOG_ALL_MESSAGES

Required/Optional: *Optional*

The VPF writer module uses FME factory pipeline to write features. Factories in the pipeline generate many information messages which may or may not be desirable to the user. The value for this directive will determine whether or not to log messages generated by FME factory pipeline used within the writer. If **YES**, then all the messages including messages from factory pipeline and the writer itself will be logged. If **NO**, then only messages coming from the writer module will be logged. Any messages generated by factory pipeline within the writer will not be logged.

Workbench Parameter: *Log all messages*

PRODUCT

Required/Optional: *Optional*

The value for this parameter is the type of VPF product database to produce. The writer provides 6 possible choices: **VMAPO**, **VMAP1**, **VMAP2**, **DNC**, **UVMAP** and **FFD**. It is possible to add your own product type. To do so, copy the schema template for your product into **FME_HOME/vpf**. See one of the other schema templates in **FME_HOME/vpf** for an idea of what is expected. Secondly, open the file **FME_HOME/metafile/vpf_db.fmf** and add your product type on the line:

```
GUI CHOICE PRODUCT DNC%FFD%VMAPO%VMAP1%VMAP2%UVMAP VPF Product Name:
```

by adding %<product_type> after UVMAP. An example of this is:

```
GUI CHOICE PRODUCT DNC%FFD%VMAPO%VMAP1%VMAP2%UVMAP%NEW_PRODUCT VPF Product Name:
```

The name of the product type in the metafile must be the same as the folder containing the schema template within **FME_HOME/vpf**, although the case of the letters can be different. Contact Safe Software support if you experience difficulties trying to use a specific product.

Workbench Parameter: *VPF Product Name*

SKIP_TILE_CLIPPING

Required/Optional: *Optional*

If this directive is given, the data is assumed to be already clipped to the tiles given; this can be used for a performance increase when the source dataset is also VPF and the data has not been modified in such a way as to violate the integrity of the tiling.

Workbench Parameter: *Skip tile clipping*

WRITER_MODE

Required/Optional: *Optional*

Note: For more information on this directive, see the chapter *Database Writer Mode*.

In **UPDATE** mode, the destination dataset may contain previously written coverages; existing entries in the database-level Coverage Attribute Table (CAT) will be left intact. Note that each coverage being written by the current translation will still be overwritten in this mode; the point of this mode is to allow separate coverages to be written by distinct translations and for the destination database to be valid as a whole.

In **OVERWRITE** mode, the destination directory tree is cleared at the beginning of the translation. Use caution with this mode, since any existing data in the destination directory will be lost.

Valid values: **UPDATE** and **OVERWRITE**

Default: **UPDATE**

Workbench Parameter: *Writer MODE*

Feature Representation (VPF Writer)

The features in a VPF writer's feature set can be classified as either metadata or feature class features.

Metadata features are used to supply values for entries in a VPF database's metadata tables. There is a simple mapping from a FME metadata feature to VPF metadata table: each metadata feature represents one row in the VPF metadata table.

Feature class features are geometric entities that carry all the information necessary to populate the VPF tables related to a feature class. A typical feature class is composed of a feature table, a set of primitive tables, a primitive join table and a set of related attribute tables. When writing a geometric feature, the VPF writer breaks an incoming feature down into primitives, populates its primitive tables, feature table and related attribute tables, and then calculates the indices related to these tables.

Attributes in VPF products are case-sensitive. All the output products that the writer supports use lowercase attributes and all FME feature attributes must also be lowercase. The following table lists the format attributes that are used by the VPF writer to write geometric information.

Valid attribute values for feature attributes can be found in a product specification. When a feature is written to a table, a default value is often used.

Attribute Name	Description	Defined On
vpf_type	A feature class feature has one of the following geometric types: <code>vpf_area</code> , <code>vpf_line</code> , <code>vpf_point</code> , or <code>vpf_text</code> . A metadata feature has the geometric type <code>vpf_none</code> .	All features
vpf_text_string	The character string to be used for a feature with type <code>vpf_text</code> .	<code>vpf_text</code>
vpf_text_height	The height of a <code>vpf_text</code> feature. See Note below.	<code>vpf_text</code>
vpf_text_font	The font used to display <code>vpf_text</code> feature. See Note below.	<code>vpf_text</code>
vpf_text_color	The colour used to display <code>vpf_text</code> feature. See Note below.	<code>vpf_text</code>
vpf_text_style	The style used to display <code>vpf_text</code> geometry. See Note below.	<code>vpf_text</code>
vpf_rotation	The rotation at which the text is to be displayed. This is calculated from the lower left and lower right coordinates of the text line, and is expressed in degrees counterclockwise from due East. It defaults to 0.0 if the text geometry has only one coordinate in the VPF data.	<code>vpf_text</code>
vpf_text_scale	The scale read from the Library Header Table (LHT) in the library containing the feature. Used to calculate a new height from the <code>fme_text_size</code> if the <code>vpf_text_height</code> is not present. For all other purposes, this attribute is ignored by the writer.	<code>vpf_text</code>
vpf_original_geometry	Optional. The original geometry as read by the VPF Reader in the Well Known Text (WKT) format. It is only present if the original vpf text primitive had more than 1 coordinate. If the text feature has the same geometry as when it was read, then this attribute will provide the geometry to the writer.	<code>vpf_text</code>

Note: Only VMap0, VMap1, VMap2 and UVMAP product types support the storage of this symbolic information in symbol.rat table. DNC does not support symbol.rat tables. Text features must have the same symbolic information for each unique symbol identifier `symbol_id` supplied on a feature. The writer will ignore the symbolic information from features with no `symbol_id` attribute.

VPF Text Feature Creation

In addition to the Note given above, the VPF Writer will try to honor as much as possible the original VPF specific attributes.

If a `vpf_text_height` is present, it will be used to write the SYMBOL.RAT size attribute. If none is present but a `vpf_text_scale` is present, then a `vpf_text_height` will be guessed from the `fme_text_size`. If no `vpf_text_height` and no `vpf_text_scale` are present, then a default value of 2 will be used, as long as the text feature has a `symbol_id` attribute.

If a `vpf_original_geometry` attribute is present and the point geometry of the FME text feature is still identical to the first coordinate of the original geometry, then the original geometry will be written. If the `vpf_original_geometry` attribute is not present or the point geometry of the FME text feature is not identical to the first coordinate of the original geometry, then the current point coordinate will be written with a second point to express the current rotation, provided it is not zero. That second point, representing the lower right corner of the string, will be guessed using the `fme_text_size` (height in ground units) and the approximation that the width of a character in ground units is half its size.

Feature Type Values

A feature's feature type plays an important role in writing a VPF database. The feature type on a feature class feature tells the writer what feature class a feature belongs to; what coverage that feature class exists in; and what library the coverage belongs to. Since it is possible to have same table name under the same coverage belonging to different libraries but with different schemas, the only way a feature type can be defined as unique is by using a combination of library name, coverage name and feature class name. The syntax for a feature class feature's feature type is:

```
<library name>\<coverage name>\<feature class>
```

For example:

```
DNC      : H001\CUL\TRANSL  
VMAPO   : SASAUS\TRANS\ROADL
```

Note that the backslash separator can be changed to another character by specifying a value for the `FEAT_TYPE_SEP` parameter in the writer's mapping file.

Recognized library, coverage and feature class types will vary according to the writer's product type. For example, if a writer's product type is DNC and the writer finds a feature with the feature type `browse\libref\libref`, it will know that all the incoming features belong to the `libref` feature class in the `libref` coverage for the `browse` library in the DNC database.

The feature type for a metadata feature tells where in the database the metatable exists. A metadata feature's feature type may have one or no separators depending on where the underlying metadata table exists in the database. Features destined for a metadata table at the root level of a database will have no separators and the following syntax.:

```
<metatable name>
```

For example, if an incoming feature is destined for the database header table named `dht` the feature type for these features will simply be `dht`.

Features destined for a metadata table that exists at the library level of a database will have the syntax:

```
<library name>\<metatable name>
```

For example, if an incoming feature is destined for the library header table named `lht` in the `browse` library of a DNC database, the feature type for these features will be `browse\lht`.

VPF Topology

All VPF products have four recognized levels of topology ranging from level 0 to level 3. (See the *Military Standard, Vector Product Format, MIL-STD-2407*, for details.) Text features do not participate in topology building. Since topology is defined at the coverage level, the VPF writer accumulates all the features belonging to one coverage and then throws them in the topology pipeline. Based on the topology level the coverage belonged to, the topology pipeline

splits features into primitives and builds all the topological relationships. After the topology has been built, primitives are written to respective primitive tables and the features to the feature tables.

Note that the writer expects that the data given to it is topologically clean (that is, it is in conformance with the VPF spec on topology). The writer is not responsible for doing any topological cleaning. Any topologically unclean data may result in incorrect output data.

Tiling

VPF data can be tiled or untiled. When data is tiled, the features are first clipped for every tile and then thrown into the pipeline for topology building. This process is repeated for every tile, since topology for every tile is supposed to be independent from those of other tiles. It is possible to skip tile clipping using the directive `SKIP_TILE_CLIPPING` to improve performance. However, this directive is recommended only in certain specified conditions. (See the paragraph on `SKIP_TILE_CLIPPING` above.)

For the features that cross tile boundaries, the information of the current primitive ID, the external tile and the primitive ID in that tile are stored as triplet IDs on the primitives. By default, VPF writer does not calculate the triplet IDs due to immense overhead involved in the process.

To activate clipping in the writer, `SKIP_TILE_CLIPPING` must be set to 'no' and the `TILEREF` feature type for each of the libraries must be written.

Using the VPF Writer

In order to use the VPF writer, users should be familiar with not only the *Military Standard, Vector Product Format, MIL-STD-2407*, but also the specific product specifications (DNC, VMAP0, etc.). Each VPF product is defined by a different product specification. Each specification describes the specific implementation of the general VPF structures that are defined in the VPF Military Standard.

The VPF writer uses the schema templates installed in the `FME_HOME` directory to create and populate default metadata tables and their attributes (i.e., `dht`, `lat`, etc.)

VPF writer uses the product name as the default `database_name` if one is not specified (e.g., `vmap0`, `dnc`). To specify the database name, set the `database_name` attribute on the `dht` feature type. For the `tileref` metadata table, set the `tile_name` attribute.

When writing attributes, the required attribute values or codes must match the relevant product specification. For example, for the Culture Transportation line feature type (`<libname>\CUL\TRANSL`) the `f_code` attribute values must match the allowed FCC codes defined in the product specification.

In general, VPF writer follows these rules:

1. Allows writing multiple libraries at a time.
2. Allows adding a library to an existing database (Note that `WRITER_MODE` should be `UPDATE` to achieve this)
3. Allows adding coverage to an existing library (Note that `WRITER_MODE` should be `UPDATE` to achieve this).

The VPF writer does not, however, allow adding a feature class to an existing coverage. Once a coverage gets written, it cannot be updated. This is because the topology is built at the coverage level and adding a feature class to a coverage would require rebuilding of the topology, thus making all the existing primitive tables invalid.

Note that following metadata tables are created internally by the writer and the user has no control over them. An attempt to modify the following metadata tables will be ignored.

1. `lat` (Library Attribute Table) – since the library name is dictated by the feature type
2. `lht` (Library Header Table) – since the library name is dictated by the feature type
3. `cat` (Coverage Attribute Table)
4. `fcs` (Feature Class Schema) – independent of `WRITER_MODE` since feature class updating is not supported
5. `fca` (Feature Class Attribute) – independent of `WRITER_MODE` since feature class updating is not supported

The `dht` table, however, can be updated. For instance, `dht` feature types can be used to set the database name, as described earlier. Note that the VPF specification limits the length of database name to 8 characters.

Value Description Tables

Both of the value description tables `int.vdt` and `char.vdt` are created internally by the writer and the user has no control over them. The scope of both the tables is limited to coverage level, therefore the tables are independent of `WRITER_MODE` since feature class updating is not supported (that is, once they are written, they can never be updated).

Note that the contents of `int.vdt` and `char.vdt` for each coverage are defined by the VPF product specification. The value description tables as provided with other schema templates contain an exhaustive list of values as allowed by their respective product specifications. When a coverage is written, `int.vdt` and `char.vdt` are created with the values actually used by the feature classes being written to that particular coverage. However, these values have to be one of those provided in the list for that coverage.

Each coverage is provided with an exhaustive list of all the coded values and descriptions for each feature class as allowed by the VPF product specification. The specification does not allow creating new values (codes) in general with the exception of those attributes which have "null", "unk" or "unknown" values in the vdt's. Each feature class should get coded values as allowed by the coverage's vdt. For example:

The character vdt of `vmap1\bnd` coverage allows only following values for `f_code` for `polbnd1.1ft`

- FA000 Administrative Boundary
- FA020 Armistice Line
- FA030 Cease-Fire Line
- FA050 Convention Line/Mandate Line
- FA060 De Facto Boundary
- FA110 International Date Line

If an attempt is made to assign, say AL070, to f-code to any feature belonging to `polbnd1.1ft`, though the value gets written but the writer warns the user that it could not find it in the vdt table. Also the `char.vdt` never gets updated with this value.

On the other hand, `polbnd1.1ft` has the attributes "nm3" and "nm4" containing "UNK" values. These attributes can have any values, say "Vancouver". Note these values will show only in the feature class table and vdt's will neither have these values nor its description.

Since the description can appear in the vdt only and the users are not allowed to create vdt's directly, any attempt of writing a description of any code (value) gets ignored by the writer. Note that the vdt's are also provided along with the schema templates which conform to the respective product specifications. The writer actually uses these vdt templates as guidelines to create vdt for the coverage being written. The vdt which gets created with the coverage will contain only the values got used in the coverage.

For a detailed list of all possible contents in `char.vdt` in BND coverage, see the Boundaries Character Value Description Table in Appendix F of the *VMAP1 Specification*.

VPF Writing Tips

1. Copy the schemas of different VPF products under the `./install` directory as described earlier. All these schemas are in conformance with VPF spec. Therefore, when writing to VPF, make sure that the destination feature type bears the full path name starting from the library. For instance, for DNC writing, the feature type should look like `a11\cul\buildnga.aft`

Note that all the VPF products allow a predefined database schema with little flexibility. There is flexibility with the library only. For example, for a DNC database, we can pick library names starting with letters a (approach), c (coastal), g (general), h (harbour) or b (browse). You can append any suffix to these library names. For instance a101, abc or a6 are all valid names provided the length does not exceed 8 characters but names like k1 or t1 would be invalid for DNC. Also, lib1 can be a valid library name for product VMAP1 but not for DNC.

Make sure that destination feature class falls under the correct directory according to the VPF database structure. For example, `a11\cul\buildnga.aft` would be a valid destination feature type, but `a11\dqy\buildnga.aft`

would not, since dqy coverage is not supposed to have the table *buildnga.aft*. However, *a11\dqy\dqyarea.aft* would be acceptable.

2. When translating from a non-VPF format to VPF, make sure that the features end up in the correct destination feature type that geometrically makes sense. All area features should be directed to **.aft* tables, line features to **.lft* tables, and so on.. This mapping is automatically done from VPF to VPF.
3. Writing to VPF database may involve a huge number of feature classes and there is no automatic way of generating a mapping file or workspace which has all the destination feature types in the format *<lib_name>\<coverage_name>\<feature_class_name>*. Feature type names of all the destination feature classes have to be brought into this format manually. However, there is a work-around to this tedious exercise by using the Merge Feature Type option in Workbench.

Mapping File Example

Here is an example mapping file extract showing how to write a two feature types: *dht* and *SASAU\SND\B-ARRIERL*.

```
*****
READER_TYPE SHAPE
READER_KEYWORD SHAPE
WRITER_TYPE VPF_DB
WRITER_KEYWORD VPF_DB

SHAPE_DATASET "G:\SampleData\shape\LOTLINES.shp"

VPF_DB_DATASET "G:\vpf\out"
VPF_DB_PRODUCT VMAP0
VPF_DB_LOG_ALL_MESSAGES NO

LOG_FILENAME "G:\vpf\shape2vpf_db.log"
LOG_APPEND NO

# -----
SHAPE_DEF LOTLINES \
  SHAPE_GEOMETRY shape_polyline \
  LOTLINE_ID number(5,0)

# -----
FACTORY_DEF * TeeFactory \
  FACTORY_NAME "Source -> Generic" \
  INPUT FEATURE_TYPE * \
  OUTPUT FEATURE_TYPE * \
  @Transform(SHAPE,FME_GENERIC,PRESERVE_GEOMETRY)

# -----
FACTORY_DEF * CreationFactory \
  FACTORY_NAME NULLGEOMETRYCREATOR \
  CREATE_AT_END no \
  NUMBER_TO_CREATE 1 \
  OUTPUT FEATURE_TYPE NULLGEOMETRYCREATOR_CREATED \
  @SupplyAttributes(? ,creation_instance,0)

# -----
FACTORY_DEF * TeeFactory \
  FACTORY_NAME "NULLGEOMETRYCREATOR_CREATED -> dht Correlator" \
  INPUT FEATURE_TYPE NULLGEOMETRYCREATOR_CREATED \
  OUTPUT FEATURE_TYPE dht \
  @Transform(FME_GENERIC,VPF_DB) \
  @SupplyAttributes(database_name,test_db)

# -----
FACTORY_DEF * TeeFactory \
  FACTORY_NAME "LOTLINES -> SASAU\SND\BARRIERL Correlator" \
  INPUT FEATURE_TYPE LOTLINES \
```

```
OUTPUT FEATURE_TYPE SASAUS\BND\BARRIERL \  
      @Transform(FME_GENERIC,VPF_DB)
```

```
# -----  
SHAPE *  
VPF_DB *
```

```
# -----  
VPF_DB_DEF dht \  
  VPF_GEOMETRY All \  
  libraryname char(10)
```

```
# -----  
VPF_DB_DEF SASAUS\BND\BARRIERL \  
  VPF_GEOMETRY All
```

```
*****
```

Virtual Reality Modeling Language (VRML) Writer

The Virtual Reality Modeling Language (VRML) Writer module enables FME to generate VRML97 files. At the time of this writing, VRML97 (ISO 14772) is the most recent revision of the VRML specification. This section assumes familiarity with this format.

Overview

VRML is the standard file format for specifying dynamic and interactive three-dimensional (3D) virtual worlds on the Internet. VRML browsers are widely available for many different platforms.

VRML Quick Facts

Format Type Identifier	VRML
Reader/Writer	Writer
Licensing Level	Base
Dependencies	None
Dataset Type	File
Feature Type	Any***
Typical File Extensions	.wrl
Automated Translation Support	No
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Not applicable
Schema Required	No
Transaction Support	No
Geometry Type	vrml_type
Encoding Support	No

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	yes		solid	yes
elliptical arc	no		surface	yes
ellipses	no		text	yes
line	yes		z values	yes
none	no			

Writer Overview

The FME VRML writer creates a single `.wrl` file. The viewpoint of this VRML-created world is initially set so the viewer is directly centred above the translated features which allows all translated features to be seen. The centrepoint of all features is dependent on the features been translated.

The VRML writer provides the option to color features according to their height using the `COLOR_LEVEL_DEF` directive. When this option is enabled, features are automatically colored, based on their z-coordinate. Individual features may override this scheme by specifying their own colors. This provides a convenient way to specify the colors for the final output. The mechanism to do this is explained over the next several sections.

Besides translating geographical information, the VRML writer also provides the option for displaying user-defined attributes for features by using the `DISPLAY_ATTRIB` directive. This option is discussed in the next section.

Writer Directives

The directives listed below are processed by the VRML writer. The suffixes shown are prefixed by the current `<writerKeyword>` in a mapping file. By default, the `<writerKeyword>` for the VRML writer is `VRML`.

DATASET

Required/Optional: *Required*

The value for this directive is the name of the VRML file to be created. If a file with this name already exists, then the file will be overwritten. A typical mapping file fragment specifying an output VRML data set looks like:

```
VRML_DATASET /tmp/347v35.wrl
```

Workbench Parameter: *Destination Virtual Reality Modeling Language (VRML) File*

ZMULTIPLIER

Required/Optional: *Optional*

This directive controls the Z exaggeration for the output VRML file. The value for this directive is any real number. The syntax of a VRML `ZMULTIPLIER` line is:

```
<writerKeyword>_ZMULTIPLIER <realNumber>
```

The default value for this directive is `1.0`.

Workbench Parameter: *Z Exaggeration*

DISPLAY_ATTRIB

Required/Optional: *Optional*

The syntax of a VRML `DISPLAY_ATTRIB` line is:

```
<writerKeyword>_DISPLAY_TEXT_SIZE (YES|NO)
```

If the value of this directive is set to `NO`, then all user-defined attributes will be ignored and not written out to the VRML file.

This directive gives the option for each VRML feature that has an attribute of `vrml_tip_over` or `vrml_tip_click` to have their user-defined attributes written out to the VRML file. This option is enabled by setting the value of the directive to `YES`.

Each VRML feature has the following characteristics when a VRML browser is used to view the output file:

- When a user moves the cursor over the feature and the `vrml_tip_over` attribute was specified for the feature, the value of the attribute will be displayed. By setting the value of this attribute to `vrml_all_attrs`, all user-defined attributes for the feature are displayed.

- When the mouse button is clicked while the cursor is over the feature and with the `vrmI_tip_click` attribute specified, the value of the attribute will be displayed. By setting the value of this attribute to `vrmI_all_attrs`, all user-defined attributes for the feature are displayed.

Either one or both attributes may be given to the feature, thereby giving a different effect for each cursor event.

Note: Enabling this option produces a considerably larger VRML output file. It may also cause a performance penalty when using a VRML browser to view the output file.

Workbench Parameter: *Display Attributes*

DISPLAY_TEXT_SIZE

Required/Optional: *Optional*

The syntax of a VRML `DISPLAY_TEXT_SIZE` line is:

```
<writerKeyword>_DISPLAY_TEXT_SIZE <textSize>
```

where `textSize` must be any real number greater than zero.

This directive defines the text size of the attributes that would be displayed when the `DISPLAY_ATTRIB` directive was enabled. The `DISPLAY_TEXT_SIZE` directive only has an effect on the VRML output file when the value of the `DISPLAY_ATTRIB` directive is set to `YES`.

Workbench Parameter: *Text Size*

COLOR_LEVEL_DEF

Required/Optional: *Optional*

The syntax of a VRML `COLOR_LEVEL_DEF` line is:

```
<writerKeyword>_COLOR_LEVEL_DEF <height> <red> <green> <blue>
```

The height must be any real number greater than or equal to zero, whereas `red`, `green`, and `blue` must be real numbers in the close interval of `0.0` to `1.0`.

This directive defines the color to be used for features starting from a particular height. It provides the convenience for features to be automatically colored based on their z-coordinate. Several `COLOR_LEVEL_DEF` directives may be defined one after another so that, in effect, a height interval coloring scheme is specified.

A typical mapping file fragment specifying several `COLOR_LEVEL_DEF` directives looks like:

```
MACRO BrightRed      1.0    0.0    0.0
MACRO BrightGreen    0.0    1.0    0.0
MACRO BrightBlue     0.0    0.0    1.0

VRML_COLOR_LEVEL_DEF 0 $(BrightRed)
VRML_COLOR_LEVEL_DEF 100 $(BrightGreen)
VRML_COLOR_LEVEL_DEF 200 $(BrightBlue)
```

For this example, features with heights from 0 to 99 will be `BrightRed`, from 100 to 199 `BrightGreen`, and from 200 onwards the features will be colored `BrightBlue`.

There is no limit on the number of `COLOR_LEVEL_DEF` directives.

PRETTY_PRINT

Required/Optional: *Optional*

This directive gives the option for the VRML writer to print the output file in a more attractive format.

The syntax of a VRML `PRETTY_PRINT` line is:

```
<writerKeyword>_PRETTY_PRINT (YES|NO)
```

The default value for this directive is NO.

Tip: Enabling this option produces a considerably larger VRML output file due to the extra blank spaces.

FACE_SET_ATTR

Required/Optional: *Optional*

Features having their `vrml_type` set to `vrml_face` are written in the output dataset as a VRML IndexedFaceSet nodes. Normally one IndexedFaceSet node corresponds to one `vrml_face` feature. This directive allows an IndexedFaceSet node to represent more than one `vrml_face` feature. The directive allows `vrml_face` features carrying a common value for the specified attribute to be grouped into one IndexedFaceSet node.

If we are writing out the provinces of Canada, each of them represented as a `vrml_face`, and each of them containing a `code` attribute whose value is made common among polygons belonging to the same province, then a specification of

```
<WriterKeyword>_FACE_SET_ATTR code
```

instructs the VRML writer to group all the polygons from the same province into a single IndexedFaceSet node. In addition, this IndexedFaceSet will be named by the value of the `code` attribute. Using the VRML DEF, for example, all `vrml_face` features containing the `code` attribute with value CA10 will be written into the following indexedFaceSet node:

```
geometry DEF CA10 IndexedFaceSet {  
  ...  
}
```

Workbench Parameter: *Face Grouping Attribute*

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), special FME feature attributes direct the VRML writer as it writes the feature into the VRML file. The most important of these is the `vrml_type` attribute because it controls the overall interpretation of the feature. The acceptable values for `vrml_type` are `vrml_text`, `vrml_line`, `vrml_face`, `vrml_point`, `vrml_surface`, and `vrml_solid`. The parameters specified to each of these are described in subsequent sections.

The VRML features may have their individual colors specified with the following attributes. These colors take precedence over the COLOR_LEVEL_DEF colors.

Attribute Name	Contents
<code>vrml_mat_diffuseColor_r</code>	The amount of red color to be mixed with the other RGB components for the final color of the feature. Range: 0.0 .. 1.0 Default: 0.8
<code>vrml_mat_diffuseColor_g</code>	The amount of green color to be mixed with the other RGB components for the final color of the feature. Range: 0.0 .. 1.0 Default: 0.8
<code>vrml_mat_diffuseColor_b</code>	The amount of blue color to be mixed with the other RGB components for the final color of the feature. Range: 0.0 .. 1.0

Attribute Name	Contents
	Default: 0.8
vrml_mat_emissiveColor_r	The amount of red color to be mixed with the other RGB components for the final glow color of the feature. Range: 0.0 .. 1.0 Default: 0.0
vrml_mat_emissiveColor_g	The amount of green color to be mixed with the other RGB components for the final glow color of the feature. Range: 0.0 .. 1.0 Default: 0.0
vrml_mat_emissiveColor_b	The amount of blue color to be mixed with the other RGB components for the final glow color of the feature. Range: 0.0 .. 1.0 Default: 0.0
vrml_mat_transparency	Specifies the transparency factor for the feature—a factor of 0.0 creates opaque shapes, a factor of 1.0 makes a shape completely transparent. Range: 0.0 .. 1.0 Default: 0.0

The following table lists other attributes for the VRML features:

Attribute Name	Contents
vrml_tip_over vrml_tip_click	The vrml_tip_over and vrml_tip_click attributes are used in conjunction with the DISPLAY_ATTRIB directive. If this directive is set to YES , then features with this attribute will have all user-defined attributes displayed by the VRML browser, as described in the <i>Writer Directives</i> section, under the heading DISPLAY_ATTRIB .
vrml_url	VRML features may have a Uniform Resource Locator (URL) bound to them. To specify the URL for a feature, include a vrml_url attribute for that feature. The value of the vrml_url attribute should specify the URL of a destination web page to which the viewer travels when the viewer clicks on that feature. Note: If the feature already contains vrml_tip_over or vrml_tip_click attributes, the vrml_url attribute will have no effect when writing out the VRML output file.

Faces

vrml_type: `vrml_face`

VRML face features are used to represent solid polygons. Face features must have at least three points. Solid geometry will be written out as the decomposed faces of that solid. The VRML writer writes out a `vrml_face` feature as a VRML `IndexedFaceSet` node.

Lines

vrml_type: `vrml_line`

Linear features must have at least two points. The VRML writer writes out a `vrml_line` feature as a VRML `IndexedLineSet` node.

Points

vrml_type: `vrml_point`

Point features must have exactly one coordinate. The VRML writer writes out a `vrml_point` feature as a VRML `PointSet` node.

Solid

vrml_type: `vrml_solid`

Solid features can have a geometry that is a box or a collection of faces.

Box geometries have a position and values for the width, height and length of the box. The VRML writer writes out a box geometry by first translating to the center of the box's coordinates and then writing a VRML `Box` node.

If the feature does not have a box then the solid will be written as multiple `vrml_face` features.

Surface

vrml_type: `vrml_surface`

Surface features are 3D geometries that may or may not form a solid. The geometry will be written as multiple `vrml_face` features.

Text

vrml_type: `vrml_text`

Text features must have exactly one coordinate. The `vrml_text` features are written to the output file first by translating by the text coordinate, then by writing out a VRML `Text` node.

VRML text features have the following attributes:

Attribute Name	Contents
<code>vrml_text_string</code>	The text string to be drawn in the VRML file. It may contain blanks and there is no limit on its length. This attribute must be present for all <code>vrml_text</code> features.
<code>vrml_font_family</code>	The name of the font used to draw the text. Range: SERIF SANS TYPEWRITER

Attribute Name	Contents
	Default: SERIF
vrml_font_style	<p>The text style to use.</p> <p>Range: PLAIN BOLD ITALIC BOLDITALIC</p> <p>Default: PLAIN</p>
vrml_font_size	<p>The height of the characters measured in VRML units.</p> <p>Range: real number > 0</p> <p>Default: 1.0</p>
vrml_rotation	<p>The rotation of the text about the z axis, measured in degrees counterclockwise from horizontal.</p> <p>Range: -360.0..360.0</p> <p>Default: 0.0</p>

Wavefront OBJ Reader/Writer

The Obj Reader and Writer module enables FME to read and write the Wavefront Obj format.

The Obj format, originally developed for use with Wavefront's Advanced Visualizer, is now used primarily to exchange 3D models between different modeling and rendering applications.

Overview

An Obj file consists of a main .obj file, that can reference an optional Material (.mtl) file. A format specification for Obj and the optional mtl file can be found here:

- www.fileformat.info/format/wavefrontobj
- www.fileformat.info/format/material

Supported OBJ File Syntax

The obj file format supports both polygonal and free-form surface objects. The Obj Reader/Writer currently supports polygonal face (f) objects (l) line and (p) point. Free-form surfaces are typically not used in .obj file exchange and are not supported at this time. For completeness, the syntax supported at this time is summarized below.

Syntax and Description

v <x> <y> <z>

Vertex position. Coordinates are floating point numbers.

vn <i> <j> <k>

Vertex normal. Coordinates are floating point numbers.

vt <u> <v> <w>

Texture coordinate.

<u> is the horizontal direction,

<v> is the vertical direction (optional in the case of a 1D raster/texture, and defaults to 0),

<w> is the depth in the case of a 3D raster/texture map (optional in the case of a 2D raster/texture, and defaults to 0).

f <v1>/<vt1>/<vn1> <v2>/<vt2>/<vn2> <v3>/<vt3>/<vn3> ...

Faces are stored as a series of three or more vertices in clockwise order. Vertices are described by their position, optional texture coordinate, and optional normal, encoded as an integer index into the respective coordinate lists. A face is generally limited to triangle or quad planar surface.

l <v1>/<vt1> <v2>/<vt2> <v3>/<vt3> ...

Lines are stored as a series of one or more vertices and optional texture coordinate. Textures are generally not mapped to lines but may be used to store tabular data or an index.

p <v1> <v2> <v3> ...

Points are stored as a series of one or more vertices.

mtllib <filename.mtl>

Syntax and Description

Material (.mtl) file references the material file that defines the materials used in the object that follow. Material files contain color illumination components and texture filename references.

usemtl <mtlname>

Material name string tag that references the material to use for the faces that follow in the obj file.

g <groupname1> <groupname2> ...

Group name string tag for the elements (faces) that follow.

o <objectname>

Object name string tag for the elements (faces) that follow.

s <smoothgroup>

Smoothgroup number to group elements together for smooth shading operations.

File Name Extension	Contents
.obj	Geometric data

A single .obj file can contain many types of geometry. However, in practice obj files generally contain only faces.

Obj files generally contain three-dimensional geometry. 2D data will be represented with one of the coordinates (generally y or z) being 0.0 for all geometry.

Wavefront OBJ Quick Facts

Format Type Identifier	OBJ
Reader/Writer	Both
Licensing Level	Professional
Dependencies	None
Dataset Type	Reader: File Writer: Directory
Feature Type	"OBJ_ELEMENT"
Typical File Extensions	.obj
Automated Translation Support	Yes
User-Defined Attributes	No
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	No
Transaction Support	No
Enhanced Geometry	Yes
Encoding Support	No
Geometry Type	obj_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	no		polygon	yes
circular arc	no		raster	no
donut polygon	no		solid	no
elliptical arc	no		surface	yes
ellipses	no		text	no
line	yes		z values	yes
none	no			

Reader Overview

The Obj reader produces FME features for geometry data in an obj file. The obj reader extracts all the geometry in an Obj file and then presents the elements one at a time to FME for further processing. Each obj element will create an FME feature.

Reader Directives

The directives processed by the OBJ reader are listed below. The suffixes shown are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the Obj reader is OBJ.

DATASET

Required/Optional: *Required*

The value for this directive is the path to the obj file.

```
OBJ_DATASET /usr/data/obj/teapot.obj
```

Workbench Parameter: *Source Wavefront OBJ File*

MOVE_TO_WORLD_COORDSYS

Required/Optional: *Optional*

Possible values are 'Yes' and 'No' with default value being 'No'. If the value is 'Yes', the companion '.prj' and '.wld' files (having the same name as the '.obj' file) will be read in order to acquire the coordinate system and the data necessary in order to convert points to the world coordinate system. Note that in the absence of a companion '.wld' file with the same name as the '.obj' file, a file named 'global.wld' will be looked for in the same directory. Similarly for the companion '.prj' file, only in that case we will only look for a file named 'global.prj'.

Workbench Parameter: *Move to World Coordinate System*

MERGE_MESH_PARTS

If the value is Yes, the group name, object name, and smoothing group information will be lost for each face in the source file, and the corresponding FME feature will contain only one mesh that contains all of the faces from the source file. This results in a more efficient representation of the data if the user does not wish to keep the additional face information.

The OBJ writer will maintain this information so it is recommended to leave this option set to "No" for OBJ to OBJ translations. If the value is set to 'No', in the case that multiple group names, object names or smoothing groups are used, this information will be preserved as traits on multiple meshes containing faces that have been grouped by these values.

Values

Yes | No (default)

* Workbench Parameter

Merge Mesh Parts

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the `mitab.dll` in the FME home directory to `mapinfo.dll`.

The syntax of the `MAPINFO_SEARCH_ENVELOPE` directive is:

```
MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

```
MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

Writer Overview

The Obj writer creates and writes feature data to an obj file.

Any old Obj file in the directory is overwritten with the new Obj file with the same name. If the Obj file can not be written the translation fails.

Note: The maximum number of digits allowed after the decimal is 6. The writer will automatically truncate any extra digits.

Note: Some viewers may not correctly render OBJ files containing 64-bit precision of coordinates. The data within the OBJ file is still correct; however, if the primary concern is visualization instead of data precision, then offsetting the x,y,z coordinates such that the model's origin is moved to (0,0,0) or another point close to this should resolve the display issue.

Writer Directives

The directives that are processed by the Obj writer are listed below. The suffixes shown are prefixed by the current <writerKeyword>_ in a mapping file. By default, the <writerKeyword> for the Obj writer is OBJ.

DATASET

The value for this directive is the path to the output directory. If the output directory does not exist, then the writer will create a new directory.

An output .obj file will be created for each feature type within the specified directory.

For example, if you are writing to feature types named *house* and *barn* to dataset `c:\data\obj` then you will have files `c:\data\obj\house.obj` and `c:\data\obj\barn.obj`.

One common mtl file will be created for the writer and shared by all output files. Feature type fanout is supported.

Mapping File Syntax

```
OBJ_DATASET c:\data\obj
```

MATERIAL_LIB

Required/Optional: *Optional*

This option specifies a full path to a mtl file to use as the Material Library (mtllib) when creating an obj file during writing. Leave this blank if you do not have an existing material library or do not use materials in your obj model.

Note: Some obj viewer applications have been known to require that there are no spaces in the material file name.

Default: *Blank*

Workbench Parameter: *Material Library File*

MATERIAL_LIB_LINKAGE

Required/Optional: *Optional*

This option specifies how the material library (.mtl) file is referenced during writing.

Range: *Relative* | *Absolute* | *Copy*

Default: *Relative*

- **Relative** - references the mtl file relative to the obj file.

In this case, the mtl file will need to be placed in a directory that is relative to the obj file being created. The mtllib directive in the obj file will reference the mtl file using a relative path.

- **Copy** - makes a copy of the mtl file and places it in the same directory with the obj file. The mtllib directive in the obj file will reference this copy with no directory in the mtl path reference.

Note: This option will copy the .mtl file. However, if there are additional files referenced from the .mtl file such as texture files, they will not be copied and will need to be manually copied.

- **Absolute** - references the mtl file using an absolute location. The mtllib directive in the obj file will reference the template file using an absolute path.

Note: The Absolute reference will use the mtl file name as is specified with the MATERIAL_LIB key word which should itself specify an absolute path to the .mtl file.

Workbench Parameter: *Material Library Linkage*

REVERSE_FACE_ORDER

Required/Optional: *Optional*

This option forces faces and lines to be written in a reverse order. This option is useful when faces are all back faced when rendering only the front side and you need to reverse all faces.

Range: *Yes* | *No*

Default: *No*

Workbench Parameter: *Reverse Face Order*

TRIANGULATE_FACES

This option forces faces to be broken into triangles. Donuts and concave faces are always triangulated, regardless of whether this option is set.

Required/Optional

Optional

Values

Yes (default) | No

*** Workbench Parameter**

Triangulate Faces

WRITE_FME_VERSION

Required/Optional: Optional

This option controls whether the writer creates a comment line specifying the FME version that was used to create the resulting output obj file. Disabling writing version information is useful to support regression tests.

Range: Yes | No

Default: Yes

Workbench Parameter: *Write FME Version to OBJ File Header*

WRITE_POINTS_AND_LINES

Required/Optional: Optional

This option controls whether the writer includes points and lines when writing to the output file. When this is set to 'No', point and line features will be silently dropped. Some applications do not render points and lines and others do not accept the file if it contains them (eg. Autodesk 3ds Max).

Range: Yes | No

Default: No

Workbench Parameter: *Write Points and Lines*

MOVE_TO_LOCAL_COORDSYS

Required/Optional: *Optional*

Possible values are 'Yes', 'No' and 'PRJ_ONLY' with default value being 'No'. If the value is 'PRJ_ONLY', a companion '.prj' file containing the coordinate system and having the same name as the '.obj' file will be written in the same directory as the '.obj' file. If the value is 'Yes', in addition to writing the '.prj' file as in the 'PRJ_ONLY' option, a companion '.wld' file with the same name as the '.obj' file will be written in the same directory as the '.obj' file and the coordinates of all the points in the written features will be normalized to the interval [-0.5, 0.5] on the largest side of their XY-bounding box. The other dimensions will be scaled proportionally. This can be used to improve precision of the written coordinates.

Workbench Parameter: *Move To Local Coordinate System*

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes*), this format adds the format-specific attributes described in this section.

Obj elements (features) consist of geometry and geometry attributes. All obj elements have one predefined attribute, `obj_type`, which identifies the type of the geometry. Geometry types are 3D (x,y,z) and `obj_face` can contain texture and normal coordinates. See the *FME Fundamentals* help file (*FME Architecture > FME 3D Support > Vertex Normals* and *FME Architecture > FME 3D Support > Texture Coordinates*).

The format-specific data `obj_group`, `obj_material_ref`, `obj_object`, and `obj_smooth_group` are treated as traits.

All obj geometry attributes are optional.

Attribute Name	Value
<code>obj_type</code>	The type of geometry read from the table. This attribute will contain one of: <code>obj_point</code> <code>obj_line</code> <code>obj_face</code> <code>obj_collection</code> Default: No default

FME Geometry Attributes Supported

These are attributes that map to the FME geometry model.

Attribute Name	Value
<code>fme_texture_coordinate_x</code> <code>fme_texture_coordinate_y</code> <code>fme_texture_coordinate_z</code>	These are named measures that hold texture coordinates on the vertices of <code>obj_face</code> .
<code>fme_vertex_normal_x</code> <code>fme_vertex_normal_y</code> <code>fme_vertex_normal_z</code>	These are the named measures that hold <code>obj_face</code> vertex normal components.

WFS (Web Feature Service) Reader

The Web Feature Service (WFS) Reader enables FME to retrieve geographic information from a WFS-compliant server.

Overview

WFS is an OpenGIS® Implementation Specification. The WFS specification defines the request and response rules for the retrieval of geographic information using Hypertext Transfer Protocol (HTTP).

The WFS reader adheres to versions 1.0.0 and 1.1.0 of this specification, which can be found at the OpenGIS Consortium website www.opengis.org.

WFS Quick Facts

Format Type Identifier	WFS
Reader/Writer	Reader
Licensing Level	Professional
Dependencies	None
Dataset Type	Universal Resource Locator
Feature Type	layer name
Typical File Extensions	Not applicable
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Not applicable
Schema Required	Optional
Transaction Support	No
Geometry Type	xml_type
Encoding Support	Yes

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	no		polygon	yes
circular arc	yes		raster	no
donut polygon	yes		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	yes		z values	no
none	yes			

Reader Overview

At minimum, all compliant WFS servers are required to encode their geodata using the OpenGIS Geography Markup Language (GML). The WFS reader invokes FME's GML reader for processing of this data.

Messages that are logged during a WFS read session are produced by both the GML reader and the WFS reader. All requests that originate from the WFS reader are prepended with a **<WFS>** string.

Reader Directives

The directives processed by the WFS reader are listed below. The suffixes shown are prefixed by the current **<ReaderKeyword>** in a mapping file. By default, the **<ReaderKeyword>** for the WFS reader is **WFS**.

DATASET

The value for this directive is the URL for the WFS server.

A typical URL specifying a WFS server looks like:

```
WFS_DATASET http://www.mywfs.com/wfs_service_path/
```

Required/Optional

Required

✳ Workbench Parameter

Web Feature Service URL

MINX, MINY, MAXX, MAXY

These optional directives are used to specify the value of the rectangular bounding box parameter value that is submitted during a request for feature information.

Required/Optional

Optional

Mapping File Syntax

The syntax for the values of these directives is:

```
<ReaderKeyword>_MINX <value>  
<ReaderKeyword>_MINY <value>  
<ReaderKeyword>_MAXX <value>  
<ReaderKeyword>_MAXY <value>
```

All values must be specified in decimal, integer or scientific notation.

If all values for MINX, MINY, MAXX and MAXY have a value of 0, then no bounding box parameter will be submitted in the WFS server feature request.

✳ Workbench Parameter

Search Envelope Min X, Search Envelope Min Y, Search Envelope Max X, Search Envelope Max Y

DEF

Mapping File Syntax

The syntax for a WFS DEF line is the same as a GML DEF line:

```
<ReaderKeyword>_DEF <elementName> \  
    xml_type <xml_type> \  
    [<attrName> <attrType>]*
```

Required/Optional

Optional

HTTP_AUTH_USER

This optional directive specifies the user name when accessing a password protected HTTP server.

The directive has no default value.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_HTTP_AUTH_USER someusername
```

* Workbench Parameter

Http Authentication User

HTTP_AUTH_PASSWORD

This optional directive specifies the password when accessing a password protected HTTP server. The directive has no default value.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_HTTP_AUTH_PASSWORD my123secret
```

* Workbench Parameter

Http Authentication Password

HTTP_AUTH_METHOD

This optional directive specifies the authentication method when accessing a password protected HTTP server.

Note that the HTTP basic access authentication is a mechanism designed to allow a client to provide credentials to a server on the assumption that the connection between them is trusted and secure. That is, any credentials passed from client to server can be easily intercepted through an insecure connection.

Values

Basic (default) | Digest | NTLM

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_HTTP_AUTH_METHOD Digest
```

* Workbench Parameter

Http Authentication Method

HTTP_PROXY

This optional directive specifies the HTTP proxy to be used for network fetches. The port number may be specified at the end the proxy by appending `:[port number]` or through the `HTTP_PROXY_PORT` directive.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_HTTP_PROXY www.someproxy.net
```

or

```
<ReaderKeyword>_HTTP_PROXY www.someproxy.net:8081
```

Note: Users may bypass the `HTTP_PROXY` and `HTTP_PROXY_PORT` directives and still have http proxy support by specifying the `http_proxy` environment variable.

The value for this environment variable should be of the form `[protocol://][user:password@]machine[:port]`, where components within `[]` are optional.

An example value for the `http_proxy` environment variable is: `www.someproxy.net:8081`.

* Workbench Parameter

Http Proxy Address

HTTP_PROXY_PORT

This optional directive is used if the HTTP proxy port was not specified in the `HTTP_PROXY` directive.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_HTTP_PROXY_PORT 8081
```

* Workbench Parameter

Http Proxy Port

HTTP_PROXY_USER

This optional directive specifies the user name when accessing a password protected proxy server. The directive has no default value.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_HTTP_PROXY_USER someusername
```

* Workbench Parameter

Http Proxy User

HTTP_PROXY_PASSWORD

This optional directive specifies the password when accessing a password protected proxy server. The directive has no default value.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_HTTP_PROXY_PASSWORD my123secret
```

* Workbench Parameter

Http Proxy Password

HTTP_PROXY_AUTH_METHOD

Required/Optional: *Optional*

This optional directive specifies the authentication method when accessing a password protected proxy server.

Required/Optional

Optional

Values

Basic (default) | Digest | NTLM

Mapping File Syntax

```
<ReaderKeyword>_HTTP_PROXY_AUTH_METHOD Digest
```

* Workbench Parameter

Http Proxy Authentication Method

XSD_DOC

This optional directive allows the reader to bypass the WFS DescribeFeatureType operation by allowing the user to explicitly specify a locally stored GML application schema.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword> XSD_DOC c:\gml\schemas\roads.xsd
```

* Workbench Parameter

Application Schema

FILTER_EXPRESSION

This directive allows the reader to send a custom OGC XML fragment filter for the GetFeature operation.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_FILTER_EXPRESSION <Filter><PropertyIsEqualTo><PropertyName>NAME  
</PropertyName><Literal>Swan Lake</Literal></PropertyIsEqualTo></Filter>
```

*** Workbench Parameter**

XML Filter Expression

MAX_RESULT_FEATURES

This optional directive is used to limit the number of features that a WFS GetFeature request retrieves.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_MAX_RESULT_FEATURES 200
```

*** Workbench Parameter**

Max Features

FME_FEATURE_IDENTIFIER

This directive may be used when the WFS reader is used in a third-party application that requires each feature in a layer be identified by a numeric identifier.

The directive allows the user to specify the attribute name for this numeric identifier.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_FME_FEATURE_IDENTIFIER feature_id
```

*** Workbench Parameter**

Numeric Identifier Attribute

SRS_AXIS_ORDER

This optional directive overrides the axis order when reading a coordinate tuple in a GML <pos> or <posList> element.

Values

1,2 | 2,1 | 1,2,3 | 2,1,3

The default value, blank, uses the coordinate system's axis order.

Required/Optional

Optional

Mapping File Syntax

For example, if the srsName is set to "urn:ogc:def:crs:EPSG:6.6.4326", and the user is sure that the coordinate order in the document is lon-lat and not lat-lon order, then this directive should be set to "1,2" so that the reader reads the data in lon-lat order:

```
<ReaderKeyword>_SRS_AXIS_ORDER 1,2
```

✳ Workbench Parameter

SRS Axis Order

READ_PREDEFINED_GML_PROPERTIES

This directive specifies whether the default and optional GML feature properties, name and description, should be read.

Values

YES | NO (default)

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_READ_PREDEFINED_GML_PROPERTIES YES
```

✳ Workbench Parameter

Read Predefined Properties

COMPLEX_PROPERTIES_AS_NESTED_LISTS

This directive specifies whether GML properties that are defined as a complex type with complex content (that is, those that have embedded children elements) should be mapped as nested list attributes within FME features.

Some complex properties, such as those that are recursively defined, cannot be mapped as nested lists. These complex properties will always be mapped as XML fragments, regardless of the value of this directive.

Values

YES (default) | NO

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_COMPLEX_PROPERTIES_AS_NESTED_LISTS NO
```

✳ Workbench Parameter

Complex Properties as Nested Lists

XML_FRAGMENTS_AS_DOCUMENTS

This directive specifies whether GML properties that are mapped as XML fragments should be converted into XML documents.

The conversion will add missing namespace declarations to the fragments, it will maintain CDATA sections, and it will also prefix an XML header declaration to the fragment. Converting the XML fragments into XML documents allows XML-based parsers, e.g., XSLT and XQuery based processors, to further process the fragments.

Values

YES (default) | NO

Required/Optional

Optional

Mapping File Syntax

```
<Reader_Keyword>_XML_FRAGMENTS_AS_DOCUMENTS NO
```

Workbench Parameter

Map XML Fragments as XML Documents

MAP_GEOMETRY_COLUMNS

This directive specifies whether the GML geometric properties should be represented as individual, and possibly multiple, geometry columns in FME feature type definitions.

A geometric column in an FME data feature is represented either as a single named geometry, or, if multiple geometry columns are present, as an aggregate geometry with multiple named geometry components, this aggregate geometry will also have its "Contains Individual Geometries" interpretation flag set.

A new attribute type has also been introduced for specifying the order and/or position of a geometric column in the feature type definition. If an attribute X has its type set to "xml_geometry" then this attribute X becomes a placeholder in the feature type definition. It is a placeholder because actual data features for the feature type definitions will not have this attribute; instead, the data features will have a geometry named "X".

Values

YES (default) | NO

Required/Optional

Optional

Mapping File Syntax

```
<Reader_Keyword>_MAP_GEOMETRY_COLUMNS NO
```

Workbench Parameter

Map Geometry Columns

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

*** Workbench Parameter**

Additional Attributes to Expose

Feature Representation

All processing for geodata returned from a WFS server is performed by the GML reader; therefore, all feature representation information is equivalent to the GML features. See the [GML Reader/Writer](#) chapter for further details.

XML (Extensible Markup Language) Reader/Writer

Note: This format is not supported by FME Base Edition.

The XML modules allow FME to read and write XML (Extensible Markup Language) documents.

This chapter assumes familiarity with XML.

Overview

XML is a recommendation of the World Wide Web Consortium (W3C), and is a meta-language for defining markup languages. This means that it allows specific markup languages to be created for specific data. More information on XML can be found in the W3C website, www.w3.org.

XML Quick Facts

Format Type Identifier	XML
Reader/Writer	Reader/Writer
Licensing Level	Professional
Dependencies	None
Dataset Type	File
Feature Type	.xml
Typical File Extensions	Yes - if an xfMap for a particular XML format already exists.
Automated Translation Support	Varies: it depends on the xfMap document.
User-Defined Attributes	Yes
Coordinate System Support	Yes
Generic Color Support	Yes, but it depends on the xfMap document.
Spatial Index	Never
Schema Required	No
Transaction Support	No
Geometry Type	xml_type

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	yes		point	yes
circles	yes		polygon	yes
circular arc	yes		raster	no
donut polygon	yes		solid	yes (reader only)
elliptical arc	yes		surface	yes (reader only)

Geometry Support				
Geometry	Supported?		Geometry	Supported?
ellipses	yes		text	yes
line	yes		z values	yes
none	yes			

Reader Overview

The XML Reader works by mapping XML elements into FME features. These mappings are defined by an XML application called *xfMap*. By extracting these mapping strategies into an *xfMap* document, the XML Reader is not tied to any particular XML format. Because of this, the XML Reader can read many disparate XML applications, ranging from GIS data to purchase orders. *xfMap* is described in more detail in *xfMap*.

Reader Directives

The following directives are processed by the XML reader. The suffixes listed are prefixed by the current <ReaderKeyword> in a mapping file. By default, the <ReaderKeyword> for the XML reader is XML.

DATASET

Required/Optional: *Required*

This keyword specifies the location of the input XML document. The XML Reader is capable of reading XML documents that are *gzipped*.

Workbench Parameter: *Source XML File(s)*

Example:

```
XML_DATASET http://www.safe.com/data/points1.xml
```

or

```
XML_DATASET C:\tmp\data\points1.gz
```

SYSTEM_ENCODING

This directive specifies the system's encoding. Its default value is the system encoding.

Required/Optional

Optional

Mapping File Syntax

```
XML_SYSTEM_ENCODING ISO-8859-3
```

* Workbench Parameter

System Encoding

XFMAP

This directive specifies the location of the *xfMap* document.

Multiple XFMAP keywords may be specified within a mapping file. Each *xfMap* will map features from the same input dataset. Alternatively, multiple *xfMaps* may be specified in a single value quoted XFMAP directive by separating each *xfMap* path with a semicolon.

Required/Optional

Optional

Mapping File Syntax

```
XML_XFMAP C:\tmp\data\features.xmp
```

or

```
XML_XFMAP "C:\tmp\drainages.xmp;C:\tmp\pits_pipes.xmp"
```

or

```
XML_XFMAP C:\tmp\drainages.xmp  
XML_XFMAP C:\tmp\pits_pipes.xmp
```

✳ Workbench Parameter

XML Map File

XFMAP_FEATURE_PATHS

This directive specifies whitespace separated xfMap match expressions.

The match expressions specify which XML elements in the dataset should be extracted into XML fragments. The fragments will be held in non-geometrical FME features under their "xml_fragment" attribute.

Two additional attributes are added to the feature. One records the element that was matched (the "xml_matched_element" attribute); the other holds an ID for that element (the "xml_id" attribute):

- The "xml_matched_element" may be used to identify which element matched the expression in the case that the last component of the matched expression is a wildcard, "*". Note the value for the "xml_matched_element" is also set as the feature type for the features.
- The "xml_id" attribute is not globally unique but is guaranteed to be unique only in the context of the dataset.

The XFMAP_FEATURE_PATHS directive is useful for decomposing large XML documents into parts, where these parts may be further operated on via downstream XML, XQuery, XSLT or text processing Workbench Transformers.

See the "Match and Except Expression" section in the xfMap documentation for details regarding the match expression.

Required/Optional

Optional

Mapping File Syntax

This example extracts the <dc:metadata> element from the dataset into an XML fragment:

```
XML_XFMAP_FEATURE_PATHS "csw:SearchResults/dc:metadata"
```

XFMAP_FEATURE_PATHS_STRUCTURE

This directive is to be used in conjunction with XFMAP_FEATURE_PATHS and allows children of the matched elements to be exposed as attributes on FME Features.

See the "Structure Element" section in the xfMap documentation for details regarding the options available and more examples.

Required/Optional

Optional

Mapping File Syntax

This example extracts the children of the elements into attributes on FME Features:

```
XML_XFMAP_FEATURE_PATHS_STRUCTURE <structure/>
```

VALIDATE_XFMAP

This directive specifies whether the input xfMap document should be validated against its Document Type Definition (DTD).

Required/Optional

Optional

Values

auto | yes | no (default)

Mapping File Syntax

```
XML_VALIDATE_XFMAP yes
```

*** Workbench Parameter**

Validate XML Map File

VALIDATE_DATASET

This directive specifies whether the input XML document should be validated against a DTD or an XML schema.

Required/Optional

Optional

Values

auto | yes | no (default)

Mapping File Syntax

```
XML_VALIDATE_DATASET yes
```

*** Workbench Parameter**

Validate XML Dataset File

FEATURE_ENCODING

This directive specifies which encoding the mapped FME features should be in.

Required/Optional

Optional

Values

The default value is the system encoding.

Mapping File Syntax

```
XML_FEATURE_ENCODING Shift-JIS
```

MAPPING_FILE_ENCODING

This directive specifies which encoding the FME mapping file is in.

Required/Optional

Optional

Values

When not specified, the FME mapping file is assumed to be encoded in the system encoding.

Mapping File Syntax

```
XML_MAPPING_FILE_ENCODING ISO-8859-3
```

DOCUMENT_STREAM

This directive specifies as its value the input XML document to parse; that is, the XML document is specified inline in the FME mapping file.

Note: If present, this directive overrides the DATASET directive.

Required/Optional

Optional

XFMAP_STREAM

This directive specifies as its value an inline xfMap document in the FME mapping file.

Note: If present, it overrides the XFMAP directive.

Required/Optional

Optional

XFMAP_SCHEMA

This directive specifies the xfMap(s) that are to be used when reading schema features. Multiple XFMAP_SCHEMA directives may also be specified as per the XFMAP directive.

Required/Optional

Optional

Mapping File Syntax

```
XML_XFMAP_SCHEMA C:\tmp\data\schema_features.xmp
```

or

```
XML_XFMAP_SCHEMA "C:s_drainages.xmp;C:s_pits_pipes.xmp"
```

or

```
XML_XFMAP_SCHEMA C:\tmp\schema_drainages.xmp  
XML_XFMAP_SCHEMA C:\tmp\schema_pits_pipes.xmp
```

XRS

This directive specifies the path for an XRS document. An XRS (XML Reader Switch) document allows the XML Reader to automatically configure itself to read "known" XML datasets without the need to specify in advance the appropriate xfMaps.

The directive only applies when both the XFMAP and XFMAP_STREAM directives are absent or empty. A default XRS document is also provided, so the XRS directive is optional even when it is applicable.

The default XRS document is named xrs.xml, and it is located in the xml/xrs subdirectory of the FME installation directory.

The documentation for the XRS can be found in the xml/xrs/xrs_doc.txt file.

Required/Optional

Optional

Mapping File Syntax

```
XML_XRS C:\tmp\my_xrs.xml
```

* Workbench Parameter

XRS File

XR_PIPELINE

The XML reader allows several xfMaps to be specified on the same document stream. Each xfMap may construct its own feature representations for the input stream, for example by deconstructing the hierarchy of the input stream into a flattened structure. The XR_PIPELINE directive allows an FME factory pipeline to be applied on features constructed across xfMaps.

For example, an XML format may define their areas in a hierarchy such that XML representation of an area element contains, either directly or by reference, line children, and each line contains, either directly or by reference, point elements. An xfMap can only construct one feature at a time, so if an xfMap is mapping the area elements to construct area features, then additional xfMaps are required to map line and point elements to construct line and point features, respectively. The XR_PIPELINE directive can then be used to apply a factory pipeline on the areas, lines, and point features for further processing before these are output to FME, say to assemble their topology, if the XML format was topologically based.

Required/Optional

Optional

Mapping File Syntax

```
XML_XR_PIPELINE C:\tmp\my_topology_assembler.fmi
```

XFMAP_KEYWORD

This directive allows the specification for name-value pairs that become accessible in an xfMap with the <keyword> expression wherever expression sequences are allowed. See the xfMap <keyword> expression section documentation for more information about its usage.

This directive may occur a multiple number of times in the FME mapping file for multiple name-value pairs.

Required/Optional

Optional

Mapping File Syntax

```
XML_XFMAP_KEYWORD key0 value0  
XML_XFMAP_KEYWORD key1 "my other value"
```

XFMAP_KEYWORD_FILE

This directive is similar to XFMAP_KEYWORD, but it allows name-value pairs to be specified in an external file.

The XFMAP_KEYWORD_FILE is an XML document. Each keyword name-value pair in the document is specified with a <keyword> element:

```
<keyword name="..." value="..."/>
```

Each keyword may in addition belong to a named group, thereby allowing keyword names to repeat when these belong to different groups. A keyword belongs to a named group if it is a child of the <group> element:

```
<group name="...">
    <keyword .../>
    <keyword .../>
    ...
    <keyword .../>
</group>
```

Keywords that are not specified as children of a group are in the default group. Keywords that belong to the default group must be children of the root element. The root element for a XFMAP_KEYWORD_FILE document is the <keywords> element.

Note that the keywords defined in the FME mapping file with XFMAP_KEYWORD are also in the default group, and they will take precedence over keywords defined in the XFMAP_KEYWORD_FILE document if a keyword clash occurs.

Required/Optional

Optional

Mapping File Syntax

sample_keyword_file.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<keywords>
    <group name="MyGroup">
        <keyword name="key1" value="val-1"/>
        <keyword name="key2" value="val-2"/>
    </group>
</keywords>

XML_XR_PIPELINE C:\tmp\sample_keyword_file.xml
```

SEARCH_ENVELOPE

This keyword specifies the spatial extent of the feature retrieval. Only features that intersect this bounding box are returned by the reader. If this directive is not specified, then all features are returned.

Note that this directive is only honoured by the MITAB-based MapInfo reader in FME. This is the only MapInfo reader available on the UNIX platforms supported by FME, and can optionally be enabled on Windows platforms by renaming the `mitab.dll` in the FME home directory to `mapinfo.dll`.

The syntax of the `MAPINFO_SEARCH_ENVELOPE` directive is:

```
MAPINFO_SEARCH_ENVELOPE <minX> <minY> <maxX> <maxY>
```

The coordinate values specified are measured in the ground units of the input data.

The example below selects a small area in a lat/long dataset for extraction:

```
MAPINFO_SEARCH_ENVELOPE -130 49 -128 50.1
```

SEARCH_ENVELOPE_COORDINATE_SYSTEM

This directive specifies the coordinate system of the search envelope if it is different than the coordinate system of the data.

The COORDINATE_SYSTEM directive, which specifies the coordinate system associated with the data to be read, must always be set if the SEARCH_ENVELOPE_COORDINATE_SYSTEM directive is set.

If this directive is set, the minimum and maximum points of the search envelope are reprojected from the SEARCH_ENVELOPE_COORDINATE_SYSTEM to the reader COORDINATE_SYSTEM prior to applying the envelope.

Required/Optional

Optional

Mapping File Syntax

```
<ReaderKeyword>_SEARCH_ENVELOPE_COORDINATE_SYSTEM <coordinate system>
```

* Workbench Parameter

Search Envelope Coordinate System

CLIP_TO_ENVELOPE

This directive specifies whether or not FME should clip features to the envelope specified in the SEARCH_ENVELOPE directive.

Values

YES | NO (default)

Mapping File Syntax

```
<ReaderKeyword>_CLIP_TO_ENVELOPE [yes | no]
```

* Workbench Parameter

Clip To Envelope

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

* Workbench Parameter

Additional Attributes to Expose

Writer Overview

The XML Writer allows the FME to write out XML documents. The XML Writer can work in three modes: TABLES_ATTRIBUTES_ONLY, TABLES, and XFMAP mode. Currently only the TABLES_ATTRIBUTES_ONLY mode is implemented. In this mode, the FME features are written as rows, which are represented as XML elements in the output document belonging to a particular table, and defined through an XML DEF line.

Writer Directives

The directives processed by the XML Writer are listed below. The suffixes shown are prefixed by the current <WriterKeyword> in a mapping file. By default, the <WriterKeyword> for the XML writer is XML.

DATASET

Required/Optional: *Required*

This keyword specifies the location for the output XML document.

Workbench Parameter: *Destination XML (ExtensibleMarkupLanguage) File*

Example:

```
XML_DATASET c:\data\purchases.xml
```

WRITER_MODE

Note: For more information on this directive, see the chapter Database Writer Mode.

This controls the XML writer operation mode. The valid values are:

- TABLES_ATTRIBUTES_ONLY: In this mode, the XML Writer creates an XML document containing XML elements representing tables and rows. Each table is defined by a DEF line, and each FME feature corresponding to that DEF line defines a row for the table. Geometry is not supported in this mode.

Two documents are created in this mode: An XML Schema document, whose contents are controlled through the DEF lines, and an instance document that conforms to the schema document; the instance document is specified by the DATASET keyword.

A third XML document is also created if the GENERATE_XFMAP is set to yes. This document can be used by the XML Reader to read back the output dataset into the FME. See the GENERATE_XFMAP directive for details.

- TABLES: This mode has not yet been implemented.
- XFMAP: This mode has not yet been implemented.

If it is not specified then the WRITER_MODE defaults to TABLES_ATTRIBUTES_ONLY.

Required/Optional

Required

XSD_DOC

Required/Optional: *Optional if WRITER_MODE is XFMAP, otherwise Required*

Specifies the location for the XML Schema document. This keyword is optional. If the keyword is not specified or if it does not contain a path, then the XML Schema document is generated in the same directory as the one specified through the DATASET keyword. If this keyword is set to a URI, then XSD output will be suppressed.

Workbench Parameter: *XML Schema Document*

Example:

```
XML_XSD_DOC c:\data\purchases.xsd
```

TARGET_NS_URI

Required/Optional: *Optional*

This keyword allows the specification of the target namespace URI for the generated XML Schema document. All elements in the XML Schema document will reside in this namespace.

The default value for this keyword is:

```
http://www.safe.com/xml/xmltables
```

Workbench Parameter: *Target Namespace URI*

Example:

```
XML_TARGET_NS_URI http://www.mytables.com/purchases
```

TARGET_NS_PREFIX

This directive allows the specification of the target namespace prefix for the generated XML Schema document.

The default value depends on the value of TARGET_NS_URI. If TARGET_NS_URI is using the default value, the default value for this directive is **fme**. If TARGET_NS_URI is not using the default value, the default value for this directive becomes the default prefix, which is the empty string.

Required/Optional

Optional

Mapping File Syntax

```
XML_TARGET_NS_PREFIX ps
```

*** Workbench Parameter**

Target Namespace Prefix

TABLES_SEQUENCE

Required/Optional: *Optional*

The sequence order for the rows in the output DATASET defaults to following the order of the XML DEF line specification in the FME mapping file. This keyword allows the user to change that order by specifying a sequence of table names; the sequence must be a subset of the tables defined by the DEF lines.

Example:

If the XML DEF lines define four tables with names – state, river, city, and road – then the TABLES_SEQUENCE may be used to control the output sequence order to be city, state, river, and road by specifying:

```
XML_TABLES_SEQUENCE 'city state river road'
```

or

```
XML_TABLES_SEQUENCE 'city state'
```

The second alternative is valid since the remaining tables that are not listed will be output following the order of the mapping file XML DEF lines.

The value of the XML_TABLES_SEQUENCE keyword must be enclosed in quotes if more than one table is listed.

TABLES_ROOT_ELEMENT

Required/Optional: *Optional*

Allows the specification of the root element name for the output DATASET. If it is not specified, the root element name defaults to xml-tables.

Workbench Parameter: *Tables Root Element Name*

Example:

```
XML_TABLES_ROOT_ELEMENT purchases
```

WRAP_TABLES

Required/Optional: *Optional*

The valid values for this keyword are yes and no. When this keyword is set to yes, the elements which represent the rows for a particular table within the output DATASET document will be wrapped by a container element. The default value for this keyword is yes.

The wrapper element name will be the name of the rows prepended by the value of the TABLE_WRAPPER_PREFIX and suffixed by the value of the TABLE_WRAPPER_SUFFIX.

Workbench Parameter: *Wrap Table Elements*

Example:

```
XML_WRAP_TABLES yes
```

TABLE_WRAPPER_PREFIX

Required/Optional: *Optional*

This keyword is to be used in conjunction with the WRAP_TABLES keyword, and will take effect only if that keyword is set to yes. The default value for this keyword is the empty string.

Workbench Parameter: *Table Wrap Prefix*

Example:

```
XML_TABLE_WRAPPER_PREFIX prefix-
```

TABLE_WRAPPER_SUFFIX

Required/Optional: *Optional*

This keyword is to be used in conjunction with the WRAP_TABLES keyword and will take effect only if that keyword is set to yes. The default value for this keyword is -table.

Workbench Parameter: *Table Wrap Suffix*

Example:

```
XML_TABLE_WRAPPER_SUFFIX -mytables
```

SUPPRESS_XML_DOCUMENT

Required/Optional: *Optional*

The valid values for this keyword are yes and no. This keyword allows the suppression of the actual XML document. The default value for this keyword is no.

Workbench Parameter: *Suppress XML Document*

Example:

```
XML_SUPPRESS_XML_DOCUMENT no
```

SUPPRESS_XSD_DOCUMENT**Required/Optional:** *Optional*

The valid values for this keyword are yes and no. This keyword allows the suppression of the XML Schema document. If the XSD_DOC keyword is set to a URI, XSD output is suppressed regardless of this value. The default value for this keyword is no.

Workbench Parameter: *Suppress XSD Output*

Example:

```
XML_SUPPRESS_XSD_DOCUMENT no
```

GENERATE_XFMAP**Required/Optional:** *Optional*

The valid values for this keyword are yes and no. This keyword allows the generation of a tailored xfMap document that can be used by the XML Reader to read the output DATASET document back to the FME. The default value for this keyword is yes.

Workbench Parameter: *Generate XML Map Document*

Example:

```
XML_GENERATE_XFMAP no
```

XFMAP**Required/Optional:** *Optional*

This keyword takes effect only if GENERATE_XFMAP is set to yes; it specifies the location and filename for the xfMap document to be generated. If it is not specified, then the location defaults to the same directory as the one in DATASET, and the filename becomes the basename DATASET plus the .xmp extension.

Workbench Parameter: *XML Map Document*

Example:

```
XML_XFMAP c:\data\purchases.xmp
```

GENERATE_ROW_ID**Required/Optional:** *Optional*

The valid values for this keyword are yes and no. When this keyword is set to yes, an ID attribute of XML Schema type ID will be generated for each element that represents a row of a table. The name of the attribute can be controlled by the ROW_ID_ATTR_NAME keyword. The values generated for the ID attribute will be unique for the entire output DATASET. The unique values are simply generated from a positive integer number count starting from 1, and since the XML Schema ID type does not allow an ID to start with a digit, the ROW_ID_PREFIX keyword's value is used to prefix the ID. The default value for this keyword is no.

Workbench Parameter: *Generate Row ID Attributes*

Example:

```
XML_GENERATE_ROW_ID yes
```

ROW_ID_ATTR_NAME

Required/Optional: *Optional*

This keyword only takes effect if GENERATE_ROW_ID is set to yes. It specifies the name for an ID attribute for each element that represents a row of a table. The default value for this keyword is "row-id".

Workbench Parameter: *Row ID Attribute Name*

Example:

```
XML_ROW_ID_ATTR_NAME myID
```

ROW_ID_PREFIX

Required/Optional: *Optional*

This keyword only takes effect if GENERATE_ROW_ID is set to yes. It specifies the prefix for the unique positive integers that are generated as the values for the row IDs. This value may not start with a digit and its default value is "id".

Workbench Parameter: *Row ID Prefix*

Example:

```
XML_ROW_ID_PREFIX fid
```

Will generate ID values: fid1, fid2, fid3,

APPLY_STYLESHEET

Required/Optional: *Optional*

This keyword allows an XSLT style sheet to be applied to the final output DATASET document. The STYLESHEET_RESULT keyword may be used in conjunction with this keyword to specify the location and filename of the resulting transformation. There are no default values for this keyword.

Workbench Parameter: *XSLT Style Sheet to Apply*

Example:

```
XML_APPLY_STYLESHEET c:\data\myTransform.xsl
```

STYLESHEET_RESULT

Required/Optional: *Optional*

This keyword only takes effect if APPLY_STYLESHEET is specified. When this keyword is not present or its value is the empty string, then the resulting XSLT transformation will have the same location and filename as the output DATASET with the exception that the filename will be prefixed with transformed_.

Workbench Parameter: *Style Sheet File to Write*

Example:

```
XML_STYLESHEET_RESULT c:\data\myTransformedDoc.xml
```

OUTPUT_ENCODING

Required/Optional: *Optional*

Specifies the encoding for the output DATASET document. The default value for this keyword is UTF-8.

Workbench Parameter: *Output Dataset Encoding*

Example:

```
XML_OUTPUT_ENCODING UTF-16
```

XSD_ENCODING**Required/Optional:** *Optional*

Specifies the encoding for the generated XML Schema document. The default value for this keyword is the encoding value for the OUTPUT_ENCODING.

Workbench Parameter: *Output XML Schema Encoding***Example:**

```
XML_XSD_ENCODING ISO-8859-1
```

XFMAP_ENCODING**Required/Optional:** *Optional*

Specifies the encoding for the generated xfMap document. The default value for this keyword is the encoding value for the OUTPUT_ENCODING.

Workbench Parameter: *Output XML Map Encoding***Example:**

```
XML_XFMAP_ENCODING UTF-8
```

SYSTEM_ENCODING**Required/Optional:** *Optional*

Specifies what the system encoding should be. The default value for this keyword is the default system encoding.

Workbench Parameter: *System Encoding***Example:**

```
XML_SYSTEM_ENCODING Windows-1252
```

FEATURE_ENCODING**Required/Optional:** *Optional*

Specifies the encoding for which the FME feature attribute and feature type information are stored. The default value for this keyword is the encoding value for the SYSTEM_ENCODING.

Example:

```
XML_FEATURE_ENCODING ISO-8859-2
```

MAPPING_FILE_ENCODING**Required/Optional:** *Optional*

Specifies the encoding of the FME mapping file. The default value for this keyword is the encoding value for the SYSTEM_ENCODING keyword.

Example:

```
XML_MAPPING_FILE_ENCODING ISO-8859-1
```

DEF Lines

The DEF lines control the generation of the XML Schema document. The interpretation of a DEF line depends on the value of the WRITER_MODE keyword.

TABLES_ATTRIBUTES_ONLY Mode

When the XML Writer is in the TABLES_ATTRIBUTES_ONLY mode, the syntax of the XML DEF line is:

```
<writerkeyword>_DEF <table name>
    [<attribute name> <attribute type>]*
```

The valid values for <attribute type> are: xml_char(width), xml_int32, xml_real32, xml_decimal(width, decimal), xml_boolean, and xml_real64; these are mapped into the XML Schema built-in types: string, int, float, long, boolean, and double, respectively.

A DEF line specifies the form an element takes in the output XML document that represents a table row. The element type is defined in the generated XML Schema document as a Complex Type definition.

For example, the following DEF line:

```
XML_DEF row
    area          xml_real64
    code          xml_char(5)
    num           xml_decimal(3,0)
```

generates in the XML Schema document the following XML Schema Complex Type:

```
<complexType name="rowType">
  <sequence>
    <element name="area" type="double"/>
    <element name="code">
      <restriction base="string">
        <maxLength value="5"/>
      </restriction>
    </element>
    <element name="num">
      <restriction base="decimal">
        <totalDigits value="3"/>
        <fractionDigits value="0"/>
      </restriction>
    </element>
  </sequence>
</complexType>
```

An FME feature corresponding to the DEF line, that is, a feature with feature type row, will be written in the output DATASET document as:

```
<fme:row>
  <fme:area>28002.325</fme:area>
  <fme:code>MX02</fme:code>
  <fme:num>345</fme:num>
</fme:row>
```

Note: The example assumes that the TARGET_NS_PREFIX has been set to **fme** which is the default value.

Example 1

The following FME mapping file reads from an ESRI Shapefile and translates into XML using the XML Writer in TABLES_ATTRIBUTES_ONLY mode.

```
READER_TYPE SHAPE
WRITER_TYPE XML

SHAPE_DATASET C:\work\data\shape\MEXICO
```

XML_DATASET C:\tmp\out.xml

XML_WRITER_MODE TABLES_ATTRIBUTES_ONLY
XML_WRAP_TABLES yes
XML_TABLE_WRAPPER_PREFIX prefix-
XML_TABLE_WRAPPER_SUFFIX -suffix
XML_TARGET_NS_PREFIX xf
XML_GENERATE_ROW_ID yes
XML_ROW_ID_ATTR_NAME myid
XML_ROW_ID_PREFIX someid
XML_TABLES_SEQUENCE "states cities"
XML_APPLY_STYLESHEET C:\tmp\stylesheet.xml

XML_DEF cities		\
NAME	xml_char(51)	\
CAPITAL	xml_char(1)	\
STATE_NAME	xml_char(25)	\
POPULATION	xml_decimal(11,0)	\
XML_DEF rivers		\
NAME	xml_char(40)	\
SYSTEM	xml_char(40)	\
XML_DEF roads		\
LENGTH	xml_decimal(16,3)	\
TYPE	xml_char(40)	\
ADMN_CLASS	xml_char(20)	\
TOLL_RD	xml_char(1)	\
RTE_NUM1	xml_char(3)	\
RTE_NUM2	xml_char(3)	\
ROUTE	xml_char(40)	\
XML_DEF states		\
AREA	xml_decimal(16,3)	\
CODE	xml_char(4)	\
NAME	xml_char(25)	\
POP1990	xml_decimal(11,0)	\
POP90_SQMI	xml_decimal(20,6)	\
P_URBAN90	xml_decimal(20,6)	\
P_ING_LANG	xml_decimal(20,6)	\
P_EMPL_SEC	xml_decimal(20,6)	\
HSE_UNIT90	xml_decimal(11,0)	\

SHAPE cities		\
NAME	%NAME	\
CAPITAL	%CAPITAL	\
STATE_NAME	%STATE_NAME	\
POPULATION	%POPULATION	\
XML cities		\
xml_type	xml_point	\
NAME	%NAME	\
CAPITAL	%CAPITAL	\
STATE_NAME	%STATE_NAME	\
POPULATION	%POPULATION	\

SHAPE rivers		\
NAME	%NAME	\
SYSTEM	%SYSTEM	\

```

XML rivers
xml_type          xml_line
NAME              %NAME
SYSTEM            %SYSTEM

# -----

SHAPE roads
LENGTH            %LENGTH
TYPE              %TYPE
ADMN_CLASS        %ADMN_CLASS
TOLL_RD           %TOLL_RD
RTE_NUM1          %RTE_NUM1
RTE_NUM2          %RTE_NUM2
ROUTE             %ROUTE

XML roads
xml_type          xml_line
LENGTH            %LENGTH
TYPE              %TYPE
ADMN_CLASS        %ADMN_CLASS
TOLL_RD           %TOLL_RD
RTE_NUM1          %RTE_NUM1
RTE_NUM2          %RTE_NUM2
ROUTE             %ROUTE

# -----

SHAPE states
AREA              %AREA
CODE              %CODE
NAME              %NAME
POP1990           %POP1990
POP90_SQMI        %POP90_SQMI
P_URBAN90         %P_URBAN90
P_ING_LANG        %P_ING_LANG
P_EMPL_SEC        %P_EMPL_SEC
HSE_UNIT90        %HSE_UNIT90

XML states
xml_type          xml_area
AREA              %AREA
CODE              %CODE
NAME              %NAME
POP1990           %POP1990
POP90_SQMI        %POP90_SQMI
P_URBAN90         %P_URBAN90
P_ING_LANG        %P_ING_LANG
P_EMPL_SEC        %P_EMPL_SEC
HSE_UNIT90        %HSE_UNIT90

```

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see [About Feature Attributes](#)), this format adds the format-specific attributes described in this section.

The geometry of an FME XML feature may be identified by its `xml_type` attribute. The valid values for this attribute are:

<code>xml_type</code>	Description
<code>xml_no_geom</code>	FME Feature with no geometry.

xml_type	Description
xml_point	Point feature or an aggregate of point features.
xml_line	Linear feature or an aggregate of linear features.
xml_area	Areal feature; may be a donut, or an aggregate of areal features.
xml_text	Text feature.
xml_surface	Surface feature, may contain gaps.
xml_solid	Solid feature, may contain voids.
xml_aggregate	A feature whose geometry is a possibly heterogeneous aggregate.

Other attributes, including the feature's feature type, are dependent on the mappings that are defined in an xFMap document.

No Geometry

xml_type: xml_no_geom

Features output by the XML Reader having its xml_type attribute set to xml_no_geom do not contain any geometry data.

Points

xml_type: xml_point

Features output by the XML Reader having its xml_type set to xml_point are single coordinate features or an aggregate of single coordinate features.

Lines

xml_type: xml_line

Features output by the XML Reader having its xml_type set to xml_line are polyline features and have at least two coordinates, or an aggregate of polyline features.

Area

xml_type: xml_area

Features output by the XML Reader having its xml_type set to xml_area are either a single closed polyline feature (simple closed polygon), a donut, or an aggregate of donuts and/or simple polygons). A simple closed polygon contains at least four coordinates, with the first and last coordinate being equal.

Text

xml_type: xml_text

These are annotation features. Text features have the following special attributes associated with them.

Attribute Name	Content
xml_text_string	The text string.

Attribute Name	Content
xml_rotation	The rotation of the text measured in degrees counter-clockwise from the horizontal. Range: 0...360
xml_text_size	The size of each character in ground units. Range: Any real number ≥ 0

Surfaces

xml_type: xml_surface

Features output by the XML Reader having its xml_type set to xml_surface are either simple, topologically contiguous surfaces or aggregates of surface features. Surfaces may contain gaps.

Solids

xml_type: xml_solid

Features output by the XML Reader having its xml_type set to xml_solid are either simple, topologically contiguous solids or aggregates of solid features. Solids may contain voids.

Aggregate

xml_type: xml_aggregate

Features output by the XML Reader having its xml_type set to xml_aggregate have an aggregate geometry where the members of the aggregate may be heterogenous. For example, it is possible to create an aggregate consisting of a polygon, a point, and an aggregate of lines.

xfMap

The xfMap document contains instructions for the XML Reader to interpret XML elements into FME features. This section describes the xfMap and its relation to the XML Reader.

Reading the Input XML Document

The XML Reader reads the input document sequentially in a streaming fashion, so that the entire XML document is not kept in memory at one time. This means that very large documents may be processed.

The input document is processed according to instructions, called *mapping rules*.

Mapping Rules

An xfMap document contains mapping rules that specify the construction of certain objects, with the most important one being an FME feature. Other objects that may be constructed are group and reference objects, but these are just "helper" objects that the XML Reader uses to construct its FME features.

A feature is a generic container capable of holding attribute and geometric information. The FME contains a variety of feature processing facilities such as functions and factories. These functions and factories are available internally to the XML Reader through xfMap group objects. The XML Reader may further process the FME features with these group objects before outputting them to the FME.

The advantage of a streaming reader is the ability to read very large XML documents. A streaming reader does not hold an entire XML document in memory, so problems may arise if the current element being read requires information from previously read elements. This inherent streaming limitation may be overcome by using xfMap reference objects. Reference objects can hold information needed by the construction of features and groups when these require data from elements that were seen much earlier in the input XML document stream.

The XML Reader can currently construct three types of objects: FME features, xfMap groups, and xfMap references. Each type of object is specified through its corresponding xfMap mapping rule type.

Types of Mapping Rules

A mapping rule must be defined inside an <X-map> or <X-content-map> element, where X may be substituted by either feature, group, or reference. This means that the containing element determines the mapping rule's type.

All **mapping rules**, regardless of their type, are represented in xfMap by the <mapping> element. A mapping rule has the following form:

```
<mapping match="...">
  <!-- mapping rule contents -->
</mapping>
```

The types of mapping rule and their corresponding containing elements are:

- **feature mapping rules:** These specify the construction of FME features, and they must be defined inside the <feature-map> or <feature-content-map> elements.
- **group mapping rules:** These specify the construction of xfMap groups, and they must be defined inside the <group-map> or <group-content-map> elements.
- **reference mapping rules:** These specify the construction of xfMap references, and they must be defined inside the <reference-map> or <reference-content-map> elements.

Each mapping rule type will be described in separate sections: Feature Mapping Rules, Group Mapping Rules, and Reference Mapping Rules.

The following section describes the mapping rule's match expression and states. Every mapping rules may also contain some optional elements, which are discussed in Mapping Rules (Optional Elements).

Match and Except Expression

A mapping rule has a match expression that specifies which elements from the input XML document stream may trigger its activation. The except expression has precisely the same syntax as the match expression, but serves to limit which elements trigger the mapping rule activation. The match expression has the mechanism to identify a desired element based on the element's:

- name
- ancestors, and/or
- attribute names and/or values.

The **match expression** is represented, in xfMap, by the <mapping> element's match attribute. The syntax for the match expression's value is:

```
(ancestorElement/)*element({index})?([booleanExpr])?
```

Where *element* specifies the QName of an element *E* in the input document stream. *E* is called the **match QName**.

Note: Since *element* specifies a QName the prefix for the QName must be bound to a particular URI. All XML Namespace declarations in an xfMap must appear at the root of the xfMap document, i.e., in the <xfMap> element.

But if the prefix is found not to have a corresponding XML Namespace declaration, then the prefix will be ignored in comparisons, i.e., only the local-name of the element becomes significant. This is to keep backward compatibility with existing xfMaps written before supporting XML Namespaces.

The *match QName* may be optionally prefixed by the QName of its ancestor elements *ancestorElement* with each ancestor QName separated by the forward slash /. This prefix is called the **match ancestors**.

The *match QName* may be followed by an *index*, a positive number, that is enclosed within '{' and '}'. The number enclosed within the curly braces is called the **match index**. The match index it indicates, not the position, but the count of that particular element, in the context of its parent.

The *match QName* may be suffixed by a Boolean expression, *booleanExpr*, that evaluates on *E*'s attributes. When present, the Boolean expression must be enclosed in square brackets []. This suffix is called the **match condition**.

The following illustrates the grammar of the *match condition*:

```

booleanExpr = attrCondition
             |andExpr
             |orExpr
             | '(' booleanExpr ')'

andExpr = booleanExpr 'and' booleanExpr

orExpr = booleanExpr 'or' booleanExpr

attrCondition = '@'attrName('+'|'-')
               | '@'attrName('='|'!=')('"'|"'")attrValue('"'|"'")

```

Note: The XML Reader evaluates the Boolean expression in a right associative way. For complex Boolean expressions, the use of parentheses is recommended to indicate the intended precedence of evaluation.

The *match condition* is a Boolean expression *booleanExpr*. A Boolean expression may be a single attribute condition *attrCondition*, or it may be a sequence of parenthesized logically connected attributes conditions. The value of the *match condition* then depends on the individual attribute conditions in the Boolean expression.

From the grammar above, an attribute condition production *attrCondition* may be specified in four different ways; assume that *E* is the element referred to by the *match QName* and *match ancestors* then:

- @attrName+* evaluates to true if *E* contains an attribute with the name *attrName*, else the production evaluates to false.
- @attrName-* evaluates to true if *E* does not contain an attribute with the name *attrName*, else the production evaluates to false.
- @attrName = 'attrValue'* evaluates to true if *E* contains an attribute with the name *attrName* and the value *attrValue*, else the production evaluates to false.
- @attrName != 'attrValue'* evaluates to true if *E* contains an attribute with the name *attrName* and the value of that attribute does not equal *attrValue*; else the production evaluates to false.

If the enclosing quote of the whole match expression value was a single quotation mark, then the *attrValue* in c) and d) should be enclosed in double quotation marks.

We are now ready to state the conditions for which an element in the input XML document may trigger the activation of a mapping rule.

A mapping rule *M* **matches** an element *E* when all of the following three conditions are satisfied:

- if *M*'s *match QName* equals *E*'s element QName; and
- if *M* contains an optional *match index* *i*, then *E* must be the *i*th child type refer to in 1) in the context of its parent.
- if *M* contains the optional *match ancestors* prefix, then *E*'s ancestors must equal, in the same order, as the ones listed in the prefix; and
- if *M* contains the optional *match condition* suffix, then the suffix must evaluate to true according to the *E*'s attributes.
- if *M*'s *except* expression does *not* match under the previous three rules.

2, 3 and 4 are also satisfied when the match expression does not contain a match index, match ancestors prefix or a match condition suffix, respectively.

Using wildcards: In addition, each QName either in the match QName or in the match ancestors may be substituted by a wildcard, *, that matches any QName. The wildcard can also be specified either in the prefix and/or the local-name of a QName: *:local-name, or prefix:*, or *:.* (which is the same as a single *).

When a match expression *M* matches an element *E* in the input XML document stream, we also say that the mapping rule *R* containing *M* matches *E*. That is, **R matches E**.

Consider, for example, the following input XML document fragment:

```

<player gender="female" name="Laura" id="3453" position="defense">
  <coach> <id>1234</id> </coach>
  <coach> <id>5678</id> </coach>
  <manager> <id>7889</id> </manager>
</player>

<player gender="male" name="Juan" id="1234" position="forward">
  <coach> <id>1234</id> </coach>
  <coach> <id>5678</id> </coach>
  <manager> <id>7889</id> </manager>
</player>

<player gender="male" name="Lucas" id="1234">
  <coach> <id>1234</id> </coach>
  <coach> <id>5678</id> </coach>
  <manager> <id>7889</id> </manager>
</player>

```

The following mapping rules matches various elements from the above XML document fragment:

```

<mapping match="player">
  <!-- matches all player elements -->
</mapping>

<mapping match="player{3}">
  <!-- matches the 3rd player element -->
</mapping>

<mapping match="coach/id">
  <!-- matches all id elements having a parent coach element -->
</mapping>

<mapping match="coach{2}/id">
  <!-- matches the id for the second child coach of a player -->
</mapping>

<mapping match="manager/id">
  <!-- matches all id elements having a manager parent -->
</mapping>

<mapping match="manager{1}/id">
  <!-- matches the first manager's id in a player element, note that we don't set the
match
index to 3 since the index does not specify the position (manager is the 3rd child
of every player). ->
</mapping>

<mapping match="player[@gender='female' or @position='forward']">
  <!-- matches all player elements that are female in gender
playing a forward position -->
</mapping>

<mapping match="player[@position-]">
  <!-- matches all player elements containing no position
attribute -->
</mapping>

<mapping match="player[@gender='male' and @position='backward']">
  <!-- matches all players that are male, playing a backward
position and are not named Juan -->
</mapping>

<mapping match="player/*">
  <!-- matches any element having a player element as its parent -->

```

```

</mapping>

<mapping match="player/*{2}">
  <!-- matches the second element having a player element as its parent -->
</mapping>

```

Specifying several match expressions for one mapping rule

A mapping rule can have more than one **match expression**. The syntax for multiple **match expressions** in the <map- ping> element's match attribute is:

```

(ancestorElement/*element({index})?([booleanExpr])?
 [<whitespace>(ancestorElement/*element({index})?([booleanExpr])?)]*

```

The match expressions are evaluated in order and a mapping rule **R matches an element E** if any of the **match expressions** in the mapping rule **matches** element E.

For example, consider the following rule XML file:

```

<?xml version="1.0"?>
<c:colours xmlns:c="http://my.colours.com/colours"
  xmlns:p="http://my.colours.com/primary"
  xmlns:o="http://my.colours.com/other">
  <p:red>255,0,0</p:red>
  <p:green>0,255,0</p:green>
  <p:blue>0,0,255</p:blue>
  <o:orange category="oranges">255,165,0</o:orange>
  <o:DarkOrange category="oranges">255,140,0</o:DarkOrange>
  <o:pink>255,192,203</o:pink>
  <o:brown category="browns">165,42,42</o:brown>
  <o:beige category="browns">245,245,220</o:beige>
</c:colours>

```

And the following mapping rule fragments:

```

<xfMap xmlns:c="http://my.colours.com/colours"
  xmlns:p="http://my.colours.com/primary"
  xmlns:o="http://my.colours.com/other">
  <!-- Note that we bounded the 'c', 'p', and 'o' prefixes to their corresponding URIs
  in the namespace
  declarations in the xfMap root element. -->
  ...
  <mapping match="c:colours/p:red c:colours/p:green">
  <!-- Matches either the red or the green element; notice that
  the match expressions are whitespace separated. -->
  </mapping>
  ...
  <mapping match="c:colours/o:*[@category='oranges' or @category='browns']
  c:colours/p:*">
  <!-- Matches any element that belong to the oranges or brown categories or any ele-
  ment that
  belong to the primary colours namespace. i.e, the 'orange', 'DarkOrange', 'brown',
  'beige', 'red', 'green' and 'blue' elements -->
  </mapping>
  ...
  <mapping match="c:colours/*[@category-]>
  <!-- Matches any element having no category attribute, i.e., the 'red', 'green',
  'blue', and
  'pink' elements. -->
  </mapping>
  ...
  <mapping match="c:colours/o:*[@category-]>
  <!-- Matches any element in the "http://my.colours.com/other" having no category
  attribute,
  i.e., the 'pink' element. -->
  </mapping>

```

```
...
<xfMap>
```

Limiting mapping rule activation with except expressions.

A mapping rule can have one or more **except expression**. The syntax for **except expressions** in the <mapping> element's except attribute is identical to match expressions, i.e:

```
(ancestorElement/*element([booleanExpr])?
[<whitespace>(ancestorElement/*element([booleanExpr])?)*)
```

The except expressions are evaluated in order and a mapping rule **R fails to match an element E** if any of the **except expressions** in the mapping rule **matches** element **E**.

For example, consider the following rule XML file and mapping rule fragments:

```
<?xml version="1.0"?>
<family-tree>
  <grandparent> ...</grandparent>
  <grandparent> ...</grandparent>
  <grandparent> ...</grandparent>
  <grandparent> ...</grandparent>
  <parent>
    <name>Tristan Read</name>
    <gender>m</gender>
    <age>40</age>
  </parent>
  <parent>
    <name>Danielle Read</name>
    <gender>f</gender>
    <age>43</age>
  </parent>
  <child>
    <name>Griffen Read</name>
    <gender>M</gender>
    <age>12</age>
  </child>
</family-tree>
```

Then the following mapping rule would match only the children and parents:

```
<mapping match="family-tree/*" except="grandparent">
  ...matches all family members except the grandparents
</mapping>
```

This is quite useful when it is easier/briefer to define a match set negatively (everything but ...) rather than positively (the match is a, or b, or c, or d, ...).

Mapping Rule States (activation, execution, suspension, and de-activation)

This section describes the different states of a mapping rule. A mapping rule can either be activated, suspended, executing, or de-activated.

Let *E* be the element in the input XML document whose start-tag is being read,

let *C* be the context element; we define the **context element** to be the most recently read element for which its end-tag is yet to be read, and

let *R* be a mapping rule in the xfMap document that is considered for matching (see the *search-sets* section below, which describes when a mapping rule can be considered for matching),

then the following lists the possible states for a mapping rule *R*:

- a. *R* is **activated**, if *R* matches *E*.
- b. *R* is **executing** as long as it is activated and *E* equals *C*.

- c. *R* is **suspended** as long as it is activated and *E* does not equal *C*.
- d. *R* is **de-activated** after *E*'s end-tag is read.

Note: All mapping rules in the xfMap document are initially deactivated.

Using Force Elements During Mapping Rule Activation and Deactivation

In order to provide hooks directly into the mapping rules, There are force elements which allow you to evaluate an expression when a mapping rule is activated, deactivated or both. The force element also provides a way to halt execution of the translation. While the expression evaluated will not be used, some expressions have side effects (such as logging, or arbitrary TCL scripts) which will only be activated when the expression is evaluated.

The force element must be an immediate child of the mapping element. There are four possible attributes:

- a. **onActivation**: the legal values are "true" or "false"
- b. **onDeactivation**: the legal values are "true" or "false"
- c. **halt**: the legal values are "true" or "false"
- d. **halt-on**: where the legal value is any, including the empty, string

The halt attribute defaults to false if it is not specified.

The onActivation and onDeactivation attributes specify when the contained expression should be evaluated. At least one of onActivation or onDeactivation must be set to 'true' or the expression will not be evaluated. It is possible to specify that an expression be evaluated both when the mapping rule is activated and when it is deactivated. This is often useful when combined with the <logexpr> element (discussed later) to determine when various mapping rules match.

There is no default values for the halt-on attribute, this attribute provides the ability for a <force> element to conditionally halt the xfMap. The attribute will cause the xfMap to halt, if it is present and the expression sequence within the <force> element evaluates exactly to the string set on the halt-on attribute.

The following example will halt the reader when a <Tablet> element is matched, the message "Tablet' - is not supported" will also be printed as an error to the FME logfile.

```
<mapping match="Tablet">
  <force onActivation="true" halt="true">
    <logexpr severity="error">
      <arg>
        <literal expr=""/>
        <matched expr="qname"/>
        <literal expr="" - is not supported"/>
      </arg>
    </logexpr>
  </force>
  ...
</mapping>
```

Note: There is one significant limitation on <force> elements if the contained XML expression sequence has an 'extract' element. Since the XML reader is a streaming reader, then when a mapping rule is activated, the entire content of the XML element that triggered it is not yet read. In fact, only attributes of the triggering element will be available for extracting during the activation phase. Extract elements which attempt to reference child elements of the triggering element will evaluate to the empty string on activation of the mapping rule. No such restriction is imposed on mapping rule deactivation.

Search-sets

The XML Reader partitions the mapping rules that are specified in an xfMap document into subsets. A subset of mapping rules is called a *search-set*. Every search-set is not examined for matching mapping rules when an element is being read. The XML Reader only looks into a search-set when that search-set becomes an *active-search-set*.

The XML Reader maintains several active-search-sets:

- a. **one or more feature-search-set(s)**: the multi-feature-construction attribute on the <feature-map> element controls the number of feature-search-sets. By default only one feature-search-set exist, several feature-search-set may exist at a time when the multi-feature-construction attribute is set to true. The XML reader allows multiple features to be constructed at a time by always having an additional feature-search-set set to the <feature-map>, only the activation of the mapping rules in the <feature-map> trigger the construction of a new FME feature.
- b. **one group-search-set**: the active-search-set containing *group mapping rules*.
- c. **one reference-search-set**: the active-search-set containing *reference mapping rules*.

The next section describes how the contents, i.e., the mapping rules, of an active-search-set change.

Contents of an active-search-set (Default Contents)

The contents of an active-search-set change as mapping rules activate and de-activate. This section describes only the default contents of an active-search-set; however, it is possible for an executing mapping rule to override the defaults. The section titled **Mapping Rules (Optional Elements)** describes how the default contents of an active-search-set may be explicitly changed.

When no mapping rules are activated, the active-search-sets contain the following *default contents*:

- a. *feature-search-set*: contains all the mapping rules defined under the <feature-map> element.
- b. *group-search-set*: contains all the mapping rules defined under the <group-map> element.
- c. *reference-search-set*: contains all the mapping rules from the <reference-map> element.

Note: If any of the <feature-map>, <group-map> or <reference-map> elements are not present in the xfMap document, then their corresponding active-search-sets will always be empty.

When activated mapping rules exist, the active-search-sets contain the following *default contents*:

- a. *feature-search-set*: if at least one feature mapping rule is activated, then it contains all the mapping rules defined under the <feature-content-map> element.
- b. *group-search-set*: if at least one group mapping rule is activated, then it contains all the mapping rules defined under the <group-content-map> element.
- c. *reference-search-set*: if at least one reference mapping rule is activated, then it contains all the mapping rules defined under the <reference-content-map> element.

Note: If any of the <feature-content-map>, <group-content-map> or <reference-content-map> elements are not present in the xfMap document, then their corresponding active-search-set will be empty.

For example, if a *feature mapping rule* is activated, the feature-search-set can only contain mapping rules that are defined under the feature-content-map element.

Expression Elements (Extract and Literal)

Many of the elements that make up a mapping rule need some kind of information as input. The xfMap provides a general input mechanism with **expression elements**. We can therefore start to discuss these expression elements regardless of the xfMap elements that may use them.

Expression elements are so called because they contain an `expr` attribute whose value we call the **expression string**. This section describes two types of expression elements: the *extract* and *literal* expressions. Other types of expression elements are described in later sections.

Extract Expressions

The **extract expression** provides a mechanism to locate and extract data from elements in the input XML document stream.

When we define a mapping rule R , we intend it to match an element E in the input stream. An extract expression that is defined inside R , allows R to locate and extract data from E or E 's children.

The **extract expression** is represented in xfMap by the `<extract>` element. Its `expr` attribute holds the value of its *expression string*:

```
<extract expr="..."/>
```

The *expression string* allows the following to be extracted from the input XML document:

- a. **the matched element's text content** – when the expression string is the sole: `'`
- b. **any of the matched element's attribute values** – when the expression string is of the form:
`'@'attributeName`
- c. any of the matched element's descendant text content – when the expression string is of the form:
`'./'immediateChild('/'descendants)+`
- d. any of the matched element's descendant attribute values – when the expression string is of the form:
`'./'immediateChild('/'descendants)+'['@'attributeName']'`

Note: ImmediateChild and descendants in c and d above are QNames. Therefore as in the match expressions the prefixes for the QNames if any must be bound in the namespace declarations in the xfMap's root element, i.e., the `<xfMap>` element.

An element may contain many child elements at the same level with the same name – if this is the case, then the first encountered child element will be the one from which data will be extracted. Currently, the extract expression does not support indexed access to the matched element's children.

The example below illustrates the usage of the extract expression. Consider the following input XML document fragment:

```
<pfx:Test xmlns:pfx="my-test-uri">
  <pfx:myElement a1="val1" a2="val2" ... an="valN">
    this is the text context.
  </pfx:myElement>
  <pfx:myOtherElement>
    <pfx:someChild>the child value</pfx:someChild>
  </pfx:myOtherElement>
</pfx:Test>
```

First, we define a mapping rule R in the xfMap document that matches `<myElement>` element. R may contain any number of extract expressions, e_0, e_1, \dots, e_n , in its definition. (We'll ignore how R is defined - for now we only need to know that some elements in R use these extract expressions.)

```
<xfMap xmlns:pfx:"my-test-uri">
  ...
  <!-- call this mapping rule R -->
  <mapping match="pfx:myElement">
    ...
    <!-- call this e0 -->
    <extract expr="."/>
    ...
    <!-- call this e1 -->
    <extract expr="@a1"/>
    ...
    <!-- call this en -->
    <extract expr="@an"/>
    ...
    <!-- call this c0 -->
    <extract expr="./pfx:someChild"/>
```

```

    </mapping>
    ...
</xfMap>

```

The expression string in e_0 , ".", refers to the text content of `<px:myElement>`, therefore e_0 extracts "this is the text content."

The expressions strings in e_1, \dots, e_n refer to the values of the attributes a_1, \dots, a_n , therefore each of the e_1, \dots, e_n , extract val_1, \dots, val_n , respectively.

The expression string in c_0 , "./px:someChild", refers to the text content of the `<px:someChild>` element, therefore c_0 extracts "the child value"

A default value may be specified for the extract expression when the data pointed to by the expression string is not present in the input XML document stream. This **default value** is represented in `xfMap` as the default attribute of the `<extract>` element.

```
<extract expr="..." default="some default value"/>
```

The extract expression may also specify the optional `as-xml`, `preserve-cdata`, `escape-characters` and `declare-namespaces` attributes. This is done as in the following:

```

<extract expr="..."
  as-xml="[true|false]"
  escape-characters="[true|false]"
  declare-namespaces="[true|false]"
  write-xml-header="[true|false]"/>

```

When the **as-xml** attribute is set to true, the target of the extract expression will be extracted as an XML fragment with the target as the root. By default, when retrieving this XML fragment, `preserve-cdata` is set to true.

When the **preserve-cdata** attribute is set to true, the extract expression will be handled typically, with the exception that `CDATA` entities will not be ignored. That is, the opening and closing `CDATA` tags will be treated as text.

The **escape-characters** attribute defaults to false, when set to true the reader will escape characters that coincide with the XML markup, e.g., "<" and "&" are escaped to "<" and "&", respectively. Note that it is not necessary to escape these characters when the data is used outside the context of an XML document.

The **declare-namespaces** attribute defaults to false and it is only applicable when `as-xml` is set to true, i.e., when the extract expression is being used to mapped an XML subtree from the source document into an XML fragment. Because of XML Namespace scoping the XML fragments mapped from the source document may not have all their prefixes bound, setting this attribute to true instructs the extract expression to add any missing namespace declarations in the resulting XML fragments. These XML namespace valid fragments may then be further consumed by alternate XML processes, e.g., XSLT and XQuery processors.

The **write-xml-header** attribute defaults to false and it is only applicable when `as-xml` is set to true, i.e., when the extract expression is being used to mapped an XML subtree from the source document into an XML fragment.

Literal Expressions

We can also provide literal data to a mapping rule with a **literal expression**.

The **literal expression** is represented in `xfMap` by the `<literal>` element, its `expr` attribute holds the value of its expression string:

```
<literal expr="some literal value"/>
```

Example

The following example illustrates the usage of extract and literal expressions. Consider the following element:

```

<parent>
  <child1 desc="first child">
    child1 text content
  </child1>
  <child2 desc="second child">
    <grandchild1 desc="first grandchild"/>

```

```

                <grandchild2 desc="second grandchild">
                    second grand child text content
                    <grandgrandchild>
                        grand grand child text content
                    </grandgrandchild>
                </grandchild2>
            </child2>
        </parent>

```

Let the following mapping rule match the <parent> element above. (Again, we do not specify which elements the mapping rule contains – for the purpose of this example, we only need to know that some of these elements use the expression elements.)

```

<mapping match="parent">
    ...
    <!-- e1 -->
    <extract expr="./child1"/>
    ...
    <!-- e2 -->
    <extract expr="./child1[@desc]"/>
    ...
    <!-- e3 -->
    <extract expr="./child2[@desc]"/>
    ...
    <!-- e4 -->
    <extract expr="./child2/grandchild1[@desc]"/>
    ...
    <!-- e5 -->
    <extract expr="./child2/grandchild2[@desc]"/>
    ...
    <!-- e6 -->
    <extract expr="./child2/grandchild2"/>
    ...
    <!-- e7 -->
    <extract expr="./child2/grandchild2/grandgrandchild"/>
    ...
    <!-- e8 -->
    <extract expr="@an-optional-attr" default="myDefVal"/>
    ...
    <!-- l -->
    <literal expr="this is literal data"/>
</mapping>

```

Then:

- e_1 extracts child1 text content.
- e_2 extracts first child.
- e_3 extracts second child.
- e_4 extracts first grandchild.
- e_5 extracts second grandchild.
- e_6 extracts second grandchild text content.
- e_7 extracts grand grand child text content.
- e_8 extracts myDefVal.
- l has the literal value this is literal data.

Expression Sequence

An **expression sequence** is a sequence of expression elements. The value of an expression sequence is the appended value of all its expression elements.

Feature Mapping Rules

Feature mapping rules specify the construction for FME features, and they must be defined inside the <feature-map> or <feature-content-map> elements.

All feature mapping rules, whether they are defined under the <feature-map> element or the <feature-content-map> element, have the same structure, but only mapping rules defined under the <feature-map> element can specify the construction of brand-new FME features.

FME Feature Construction

To construct an FME feature, a mapping rule, say R_0 , that is defined under the <feature-map> element must be activated. The feature is considered to be completely constructed when R_0 de-activates (see the section titled **Mapping Rule States (activation, execution, suspension, and de-activation)** for the possible states of an xfMap mapping rule).

We'll use the following input XML document, *points1.xml*, to illustrate how a feature mapping rule from the xfMap document, *features.xmp*, directs the XML Reader to construct FME features.

points1.xml

```
<?xml version="1.0"?>
<points>
  <point name="myPoint" num="0">
    <color>
      <red>0.324</red>
      <green>0.233</green>
      <blue>0.596</blue>
    </color>
    <location x="10.0" y="0.0"/>
  </point>
  <point name="myPoint" num="1">
    <color>
      <red>0.874</red>
      <green>0.948</green>
      <blue>0.554</blue>
    </color>
    <location x="5.0" y="5.0"/>
  </point>
</points>
```

The above XML document contains two <point> elements for which we want FME features to map into. To do this, we must define a feature mapping rule under the <feature-map> element that activates when a <point> element start-tag is read.

Recall that a mapping rule R , from an active-search-set, is activated when the XML Reader reads the start-tag of an element E , in the input XML document stream, when R matches E .

The following xfMap document, *features.xmp*, contains a feature mapping rule R_p , defined in the <feature-map> element, that matches a <point> element.

features.xmp

```
<?xml version="1.0"?>
<!DOCTYPE xfMap SYSTEM "xfMap.dtd">
<xfMap>
  <feature-map>
    <!-- call this mapping rule Rp -->
    <mapping match="point">
    </mapping>
  </feature-map>
</xfMap>
```

When the above XML (*points1.xml*) and xfMap (*features.xmp*) documents are fed into the XML Reader two FME features are constructed. The following is a FME log for the constructed features:

```

+++++
Feature Type: '
Attribute: xml_type' has value xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: '
Attribute: xml_type' has value xml_no_geom'
Geometry Type: Unknown (0)
=====

```

Here is what happens:

Step		feature-search-set contents (after step)	state of mapping rule R_p (after step)
1.	The feature-search-set is initialized: No feature mapping rules are activated, so all mapping rules defined in the <feature-map> element are included in the feature-search-set.	$\{R_p\}$	de-activated
2.	The XML Reader reads the start-tag of the <points> element: No mapping rules in the feature-search-set matches the <points> element. No mapping rules are activated. The reader continues reading the contents of the <points> element.	$\{R_p\}$	de-activated
3.	The XML Reader reads the start-tag a <point> element: The mapping rule R_p in the feature-search-set matches the <point> element. R_p is activated. R_p is also in the state of execution because the <point> element is currently the context element. Since R_p was defined under the <feature-map> element, R_p directs the XML Reader to start the construction of a new FME feature. Now that a feature mapping rule is activated, the feature-search-set is modified to contain mapping rules from the <feature-content-map> element. (The <i>features.xmp</i> xfMap document has no <feature-content-map> element, the feature-search-set then becomes the empty set).	$\{\}$	activated, executing
4.	The XML Reader reads the start-tag of a <color> element. No mapping rules in the feature-search-set match the <color> element (the feature-search-set is empty).	$\{\}$	activated, suspended
5.	The XML Reader reads the start-tag of a <red> element. Again, no mapping rules are activated.	$\{\}$	activated, suspended

Step		feature-search-set contents (after step)	state of mapping rule R_p (after step)
6.	The <red> element end-tag is read. The <red> element did not cause the activation of any mapping rules, therefore no mapping rules are de-activated.	{}	activated, suspended
7.	5) and 6) are repeated for the <green> and <blue> elements.	{}	activated, suspended
8.	The <color> element end-tag is read. The <color> element did not cause the activation of any mapping rules, therefore no mapping rules are de-activated.	{}	activated, executing
9.	The XML Reader the start-tag of the <location> element. No feature mapping rules are activated (the feature-search-map is empty).	{}	activated, suspended
10.	The <location> element end-tag is read. The <location> element did not cause activation of any mapping rules, therefore no mapping rules are de-activated.	{}	activated, executing
11.	The <point> element end-tag is read. R_p is de-activated (see the <i>mapping rule states</i> section). The XML Reader considers the FME feature under construction to be complete when R_p de-activates. There are now no feature mapping rules activated so the feature-search-set now contains all the mapping rules that defined in the <feature-map> element (i.e, R_p).	$\{R_p\}$	de-activated
12.	Steps 3) through 11) are repeated for the second <point> element of the input stream. A blank FME feature is again output by the XML Reader.
13.	The <points> element end-tag is read. The XML Reader has finished reading the input XML document.	{}	de-activated

FME features constructed by the XML Reader always carry an `xml_type` attribute. This attribute is always initialized to `xml_no_geom`. Until now, the XML Reader has only constructed blank features: for the reader to construct other parts of an FME feature, a *feature mapping rule* should contain the following elements:

1. **feature-type element:** is an optional element that directs the XML Reader to set the feature type of an FME feature.
2. **attributes element:** is an optional element that directs the XML Reader to set one or more attributes for an FME feature.
3. **geometry element:** is an optional element that directs the XML Reader to construct the geometry of an FME.

Feature-type Element

Every *feature mapping rule* may (this is an optional element) contain one feature-type element, the contents of this element is an *expression sequence* whose value becomes the feature type of the constructed FME feature. This is represented in xFMap by the <feature-type> element:

```

<feature-type>
  <!-- the content is some expression sequence -->
</feature-type>

```

The feature type for the FME feature that is under construction is set only by the first activated feature mapping rule that contains a <feature-type> element. Subsequent feature mapping rules with <feature-type> elements for the same FME feature under construction are ignored by the XML Reader.

The following xfMap document, *feature_type.xmp*, contains a mapping rule R_p that matches a <point> element from the input *points1.xml* document. Here R_p has a <feature-type> element that instructs the XML Reader to set the feature type for the FME feature that is under construction.

feature_type.xmp

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xfMap SYSTEM "xfMap.dtd">
<xfMap>
  <feature-map>
    <!-- Call this mapping rule Rp -->
    <mapping match="point">
      <feature-type>
        <!-- the feature-type element has as its
              contents an xfMap expression sequence.
              An expression sequence is a sequence
              of expression elements. -->
        <extract expr="@name"/>
        <literal expr="_"/>
        <extract expr="@num"/>
      </feature-type>
    </mapping>
  </feature-map>
</xfMap>

```

When the *points1.xml* and the above *feature_type.xmp* documents are fed into the XML Reader, the following FME features are constructed:

```

+++++
Feature Type: myPoint_0'
Attribute: xml_type' has value xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: myPoint_1'
Attribute: xml_type' has value xml_no_geom'
Geometry Type: Unknown (0)
=====

```

Attributes Element

A *feature mapping rule* can contain optional *attributes* that specifies one or more attributes for the FME feature that is under construction.

The *attributes* is composed by one or more *attribute*, and each *attribute* has a *name* and a *value* that have *expression sequences* as their values. They are represented in xfMap by the <attributes>, <attribute>, <name>, and <value> elements:

```

<attributes>
  <attribute>
    <name> <!-- an expression sequence --> </name>
    <value> <!-- an expression sequence --> </value>
  </attribute>
  ...
  <attribute> ... </attribute>
  ...
</attributes>

```

Note: In a feature mapping rule, the definition of the <attributes> element must appear after the definition of the <feature-type> element.

The following xfMap document, *attributes.xmp*, contains the mapping rule R_p , which matches a <point> element from the *points1.xml* document. R_p contains an <attributes> element which directs the XML Reader to set some attributes for the FME feature that is under construction.

attributes.xmp

```
<?xml version="1.0"?>
<!DOCTYPE xfMap SYSTEM "xfMap.dtd">
<xfMap>
  <feature-map>
    <mapping match="point">
      <feature-type>
        <extract expr="@name"/>
        <literal expr="_"/>
        <extract expr="@num"/>
      </feature-type>
      <attributes>

        <!-- a1 -->
        <attribute>
          <name> <literal expr="red"/> </name>
          <value> <extract expr="./color/red"/> </value>
        </attribute>

        <!-- a2 -->
        <attribute>
          <name> <literal expr="green"/> </name>
          <value> <extract expr="./color/green"/> </value>
        </attribute>

        <!-- a3 -->
        <attribute>
          <name> <literal expr="blue"/> </name>
          <value> <extract expr="./color/blue"/> </value>
        </attribute>

        <!-- a4 -->
        <attribute>
          <name> <literal expr="color"/> </name>
          <value>
            <!-- the expression sequence below
            construct the value of this attribute
            to be: "r=x,g=y,b=z" where x,y,z are reals -->
            <literal expr="r="/>
            <extract expr="./color/red"/>
            <literal expr=",g="/>
            <extract expr="./color/green"/>
            <literal expr=",b="/>
            <extract expr="./color/blue"/>
          </value>
        </attribute>

        <!-- a5 -->
        <attribute>
          <name> <literal expr="location"/> </name>
          <value>
            <!-- the expression sequence below construct
            the value of this attribute to
            be: "x:a y:b", where a,b are reals -->
            <literal expr="x:"/>
            <extract expr="./location[@x]"/>
          </value>
        </attribute>
      </attributes>
    </mapping>
  </feature-map>
</xfMap>
```

```

<literal expr=" y:"/>
<extract expr="./location[@y]"/>
</value>
</attribute>
</attributes>
</mapping>
</feature-map>
</xfMap>

```

When the *points1.xml* and the above *attributes.xmp* documents are fed into the XML Reader the following FME features are constructed:

```

+++++
Feature Type: ~myPoint_0'
Attribute: ~blue' has value 0.596'
Attribute: ~color' has value r=0.324,g=0.233,b=0.596'
Attribute: ~green' has value 0.233'
Attribute: ~location' has value x:10.0 y:0.0'
Attribute: ~red' has value 0.324'
Attribute: ~xml_type' has value xml_no_geom'
Geometry Type: Unknown (0) =====
+++++
Feature Type: ~myPoint_1'
Attribute: ~blue' has value 0.554'
Attribute: ~color' has value r=0.874,g=0.948,b=0.554'
Attribute: ~green' has value 0.948'
Attribute: ~location' has value x:5.0 y:5.0'
Attribute: ~red' has value 0.874'
Attribute: ~xml_type' has value xml_no_geom'
Geometry Type: Unknown (0)
=====

```

**FME Feature Construction
(defining mapping rules under the <feature-content-map> element)**

A mapping rule R_1 from the <feature-map> element that matches an element E might not be able to perform all the necessary mappings for E 's child elements. When this is the case, additional mapping rules that match E 's children should be specified under the <feature-content-map> element.

Recall that when an FME feature is under construction (i.e., at least one *feature mapping rule* is activated) then the feature-search-set can only include the mapping rules from the <feature-content-map> element. When activated, these mapping rules will work on the current feature that is under construction; they do not specify the construction of a new FME feature.

Consider the following input XML document, *points2.xml*:

points2.xml

```

<?xml version="1.0"?>
<points>
  <point name="myPoint" num="0">
    <color type="red" value="0.324"/>
    <color type="green" value="0.233"/>
    <color type="blue" value="0.596"/>
  </point>
  <point name="myPoint" num="1">
    <color type="red" value="0.874"/>
    <color type="green" value="0.948"/>
    <color type="blue" value="0.554"/>
  </point>
</points>

```

A mapping rule R_1 , defined under the <feature-map> element, that matches the <point> element will have difficulty extracting information from all the <color> elements.

Recall that the extract expression does not currently support index access for the matched element's children. All R_1 is able to do now, is to extract data from the first <color> element.

An extract expression defined in a mapping rule matching the <point> element will be unable to reach the other <color> elements, since its *expression string* can only refer to the first <color> element. To overcome this limitation, we need to define mapping rules in the <feature-content-map> element that matches a <color> element.

Let R_2 be a mapping rule under the <feature-content-map> element that matches a <color> element. When R_2 activates it will suspend R_1 , and work with the same FME feature that R_1 created. The following xfMap document, *feature_content.xml*, does this exactly:

feature_content.xml

```
<?xml version="1.0"?>
<!DOCTYPE xfMap SYSTEM "xfMap.dtd">

<xfMap>
  <feature-map>
    <!-- Call this mapping rule R1 -->
    <mapping match="point">
      </mapping>
    </feature-map>
    <feature-content-map>
      <!-- Call this mapping rule R2 -->
      <mapping match="color">
        <attributes>
          <attribute>
            <name>
              <literal expr="color."/>
              <extract expr="@type"/>
            </name>
            <value>
              <extract expr="@value"/>
            </value>
          </attribute>
        </attributes>
      </mapping>
    </feature-content-map>
  </xfMap>
```

When the *points2.xml* and *feature_content.xml* documents are fed into the XML Reader, the FME features output are:

```
+++++
Feature Type: '
Attribute: color.blue' has value 0.596'
Attribute: color.green' has value 0.233'
Attribute: color.red' has value 0.324'
Attribute: xml_type' has value xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: '
Attribute: color.blue' has value 0.554'
Attribute: color.green' has value 0.948'
Attribute: color.red' has value 0.874'
Attribute: xml_type' has value xml_no_geom'
Geometry Type: Unknown (0)
=====
```

Here is what happens:

Step	feature-search-set contents (after step)	state of mapping rule R_p (after step)
1. The feature-search-set is initialized: No feature mapping rules are executing, so the mapping rules defined in	$\{R_1\}$	R_1 : de-activated R_2 : de-activated

Step	feature-search-set contents (after step)	state of mapping rule R_p (after step)
	<feature-map> element in the <i>feature_content.xml</i> document are added to the feature-search-set.	
2.	The XML Reader reads the start-tag of the <points> element: No mapping rules in the feature-search-set matches the <points> element. The reader continues reading the contents of the <points> element.	$\{R_1\}$ R_1 : de-activated R_2 : de-activated
3.	The XML Reader reads the start-tag of a <point> element: The mapping rule R_1 in the feature-search-set matches the <point> element. R_1 is activated. R_1 is also in the state of execution because the <point> element is currently the context element. Since at least one feature mapping rule is activated (i.e., R_1), the contents of the feature-search-set will then default to contain the mapping rules under the <feature-content-map> element, in this case, it is just R_2 .	$\{R_2\}$ R_1 : activated, executing R_2 : de-activated
4.	The XML Reader reads the start-tag of a <color> element: The mapping rule R_2 in the feature-search-set matches the <color> element. R_2 is activated. No brand new FME features is created because R_2 was defined in the <feature-content-map> element. The <color> element is also the context element so that R_1 is suspended, and R_2 is now executing. The <i>feature mapping rule</i> R_2 has an <attributes> element, R_2 provides these attributes to the FME feature that is being constructed.	$\{R_2\}$ R_1 : activated, suspended R_2 : activated, executing
5.	The <color> element end-tag is read. This <color> element initially triggered the activation of R_2 , so R_2 is	$\{R_2\}$ R_1 : activated, executing R_2 : de-activated

Step		feature-search-set contents (after step)	state of mapping rule R_p (after step)
	de-activated (see the section titled <i>mapping rule states</i>). The feature-search-set does not change, since there is least one activated <i>feature mapping rule</i> , that is R_1 .		
6.	Step 4) and 5) are repeated for the next two <color> elements.	$\{R_2\}$...
7.	The <point> element end-tag is read. R_1 is de-activated. The XML Reader considers the FME feature under construction to be complete when R_1 de-activates, this is because R_1 is a <i>feature mapping rule</i> defined in the <feature-map> element. No <i>feature mapping rules</i> are activated, so the mapping rules defined in <feature-map> element in the <i>feature_content.xmp</i> document becomes the feature-search-set.	$\{R_1\}$	R_1 : de-activated R_2 : de-activated
8.	Steps 3) through 7) are repeated for the second <point> element.
9.	The <points> element end-tag is read. The XML Reader has finished reading the input XML document.	$\{\}$	R_1 : de-activated R_2 : de-activated

Consider another example:

points3.xml

```

<?xml version="1.0"?>
<points>
  <point name="myPoint" num="0">
    <outline>
      <color type="red" value="1.0"/>
      <color type="green" value="1.0"/>
      <color type="blue" value="1.0"/>
    </outline>
    <fill>
      <color component="red">0.324</color>
      <color component="green">0.233</color>
      <color component="blue">0.596</color>
    </fill>
    <location x="10.0" y="0.0"/>
  </point>
  <point name="myPoint" num="1">

```

```

        <outline>
            <color type="red" value="0.5"/>
            <color type="green" value="0.5"/>
            <color type="blue" value="0.5"/>
        </outline>
        <fill>
            <color component="red">0.874</color>
            <color component="green">0.948</color>
            <color component="blue">0.554</color>
        </fill>
        <location x="5.0" y="5.0"/>
    </point>
</points>

```

feature_content2.xmp

```

<?xml version="1.0">
<!DOCTYPE xfMap SYSTEM "xfMap.dtd">

<xfMap>
    <feature-map>
        <mapping match="point">
            <feature-type>
                <extract expr="@name"/>
                <literal expr="_"/>
                <extract expr="@num"/>
            </feature-type>
            <attributes>
                <attribute>
                    <name> <literal expr="location.x"/> </name>
                    <value> <extract expr="location[@x]"/> </value>
                </attribute>
                <attribute>
                    <name> <literal expr="location.y"/> </name>
                    <value> <extract expr="location[@y]"/> </value>
                </attribute>
            </attributes>
        </mapping>
    </feature-map>

    <feature-content-map>
        <mapping match="outline/color">
            <attributes>
                <attribute>
                    <name>
                        <literal expr="outline_color."/>
                        <extract expr="@type"/>
                    </name>
                    <value> <extract expr="@value"/> </value>
                </attribute>
            </attributes>
        </mapping>
        <mapping match="fill/color">
            <attributes>
                <attribute>
                    <name>
                        <literal expr="fill_color."/>
                        <extract expr="@component"/>
                    </name>
                    <value> <extract expr="."/> </value>
                </attribute>
            </attributes>
        </mapping>
    </feature-content-map>
</xfMap>

```

When the *points3.xml*, *feature_content2.xmp* documents are input into the XML Reader, the following FME features are output:

```
+++++
Feature Type: myPoint_0'
Attribute: fill_color.blue' has value 0.596'
Attribute: fill_color.green' has value 0.233'
Attribute: fill_color.red' has value 0.324'
Attribute: location.x ' has value 10.0'
Attribute: location.y ' has value 0.0'
Attribute: outline_color.blue' has value 1.0'
Attribute: outline_color.green' has value 1.0'
Attribute: outline_color.red' has value 1.0'
Attribute: xml_type' has value xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: myPoint_1'
Attribute: fill_color.blue' has value 0.554''
Attribute: fill_color.green' has value 0.948'
Attribute: fill_color.red' has value 0.874'
Attribute: location.x ' has value 5.0'
Attribute: location.y ' has value 5.0'
Attribute: outline_color.blue' has value 0.5'
Attribute: outline_color.green' has value 0.5'
Attribute: outline_color.red' has value 0.5'
Attribute: xml_type' has value xml_no_geom'
Geometry Type: Unknown (0)
=====
```

Attribute Element (Handling Multiple Values)

The xFMap <attribute> element provides two different ways to handle attributes that contain multiple values; these reflect the way in which an FME feature interpret all of its attributes, as (name, value) pairs with the name being the attribute's identifier. An FME feature can also store a collection of attributes by using an attribute list. Attribute lists behave like primitive attributes, i.e. (name, value) pairs, except that they contain indices enclosed in braces {} in its attribute name.

When we want to get information from the XML document stream into an attribute of the FME feature, and that information maps to an attribute with multiple values, then we can either:

- a. use a "primitive attribute" and append, with a separator character, the multiple values into one string, or
- b. use the FME attribute list, or
- c. retain one attribute value out of the multiple values.

The <attribute> element has an optional type attribute that specifies how an attribute is to be mapped into an FME feature. The valid values for the type attribute are: multi-value, list, and single-value, which correspond to the a), b) and c) options above, respectively.

Multi-value Option

When the value of the type attribute is multi-value, then the <attribute> element can carry an additional optional delim attribute that specifies the separator to be used for the appended string of values. The delim attribute defaults to a comma when it is not specified.

The following example sets the separator for the appended values to be the pipe character:

```
<attribute type="multi-value" delim="|" />
...
</attribute>
```

An extra attribute on the FME feature may also be created counting the number of values in the multi-value attribute. The create-count-attribute is an optional attribute that can be used when the type attribute is set to multi-value; its valid values are yes and no, with no being its default value.

The name of the count attribute is based on the same name specified in the *expression sequence* of the <attribute>'s <name> element augmented by an optional prefix and a mandatory suffix.

The count attribute's prefix and suffix may be specified through the count-attribute-name-prefix and count-attribute-name-suffix attributes, respectively.

Both count-attribute-name-prefix and count-attribute-name-suffix are optional attributes. The default value for the count-attribute-name-prefix is the empty string, while the default value for the count-attribute-name-suffix is Count. The count-attribute-name-suffix must not be the empty string when it is present.

List Option

When the value of the type attribute is list, then a multiple value attribute in the input XML document stream will be converted into an attribute list for the FME feature that is under construction.

Single-value Option

This is the default value for the type attribute. An attribute with a particular name can only have one value in the FME feature.

The <attribute> element can carry an additional optional use attribute that specifies if the first, last or any of the values in between the first or last value out of the multiple values should be used. The legal values for this attribute are: first-value, last-value, or any natural number in 0...(last-value - 1). When it is not specified, the use attribute defaults to the last-value. If the natural numbers are used instead and if they overspecified the number of multi-value attributes from the input XML stream, then the last-value will be assumed.

Example

The following example illustrates multiple value handling. Consider the following input XML and xfMap document:

purchase.xml

```
<?xml version="1.0"?>
<purchase date="1999-10-20">
  <items>
    <item>Radio</item>
    <item>Toothbrush</item>
    <item>Towel</item>
    <item>Soap</item>
    <item>Bottled Water</item>
  </items>
</purchase>
```

purchase.xmp

```
<?xml version="1.0"?>
<!DOCTYPE xfMap SYSTEM "xfMap.dtd">
<xfMap>

  <feature-map>
    <mapping match="purchase">
      <feature-type>
        <literal expr="purchase-"/> <extract expr="@date"/>
      </feature-type>
    </mapping>
  </feature-map>

  <feature-content-map>
    <!-- This mapping rule matching the 'item' element will be activated
         5 times (the number of 'item' elements). -->
    <mapping match="item">
      <attributes>
        <!-- Gather all of the items into the multiple value
             attribute 'items1', since the attribute element
             does not have a delim attribute, the default separator,
             i.e., the comma, is used as the delimiter for the appended
             string of values. -->
        <attribute type="multi-value">
          <name> <literal expr="items1"/> </name>
          <value> <extract expr="."/> </value>
```

```

</attribute>
<!-- Same as above except that we name the attribute 'items2'
      and we specify the delimiter to be the colon character. -->
<attribute type="multi-value" delim=":">
  <name> <literal expr="items2"/> </name>
  <value> <extract expr="."/> </value>
</attribute>
<!-- Make an FME attribute list named 'list' out of the items. -->
<attribute type="list">
  <name> <literal expr="list-attr"/> </name>
  <value> <extract expr="."/> </value>
</attribute>
<!-- Use the first value of the items. -->
<attribute type="single-value" use="first-value">
  <name> <literal expr="first-item"/> </name>
  <value> <extract expr="."/> </value>
</attribute>
<!-- Use the last value of the items. -->
<attribute type="single-value">
  <name> <literal expr="last-item"/> </name>
  <value> <extract expr="."/> </value>
</attribute>
<!-- Use the 4th item. -->
<attribute type="single-value" use="3">
  <name> <literal expr="item-at-3-(4th item)"/> </name>
  <value> <extract expr="."/> </value>
</attribute>
</attributes>
</mapping>
</feature-content-map>

</xfMap>

```

The following shows a log of the FME feature constructed when the *purchase.xml* and *purchase.xmp* documents are fed into the XML Reader:

```

+++++
Feature Type: purchase-1999-10-20'
Attribute: first-item' has value Radio'
Attribute: item-at-3-(4th item)' has value Soap'
Attribute: items1' has value Radio,Toothbrush,Towel,Soap,Bottled water'
Attribute: items2' has value Radio:Toothbrush:Towel:Soap:Bottled water'
Attribute: last-item' has value Bottled water'
Attribute: list-attr{0}' has value Radio'
Attribute: list-attr{1}' has value Toothbrush'
Attribute: list-attr{2}' has value Towel'
Attribute: list-attr{3}' has value Soap'
Attribute: list-attr{4}' has value Bottled water'
Attribute: xml_type' has value xml_no_geom'
Attribute: list-attr{0}' is sequenced
Attribute: list-attr{1}' is sequenced
Attribute: list-attr{2}' is sequenced
Attribute: list-attr{3}' is sequenced
Attribute: list-attr{4}' is sequenced
Geometry Type: Unknown (0)
+=====

```

Attribute Element (Handling Optional Attributes)

Each *attribute* specified with the <attribute> element is added by default to an FME feature. A user may not want an *attribute* included in an FME feature when the *expression sequence* of an *attribute's value* evaluates to the empty string. The <attribute> element has an optional required attribute that specifies if the *attribute* is to be given to the feature based on the evaluated *expression sequence* of its *value*. The valid values for the required attribute are true and false, with its default value being true.

If the optional required attribute is set to false, then the *attribute* will not be given to the feature when the *expression sequence* of its *value* evaluates to the empty string. The following xfMap fragment sets the required attribute to false.

```

<attribute required="false">
  <name> <literal expr="my-optional-attribute"/> </name>
  <value> <extract expr="."/> </value>
</attribute>

```

Attribute Element (Sequenced Attributes)

The <attribute> element's optional type attribute may be set to sequenced; these attributes correspond to the FME feature's sequenced attributes and are useful for creating FME schema features. The following xMap fragments sets sequenced attributes for the FME feature that is under construction.

```

<attribute type="sequenced">
  <name> <literal expr="my-sequenced-attribute"/> </name>
  <value> <extract expr="."/> </value>
</attribute>

```

Geometry Element

A *feature mapping rule* may contain an optional <geometry> element that specifies the construction of the FME feature's geometry through a **geometry builder**. A *geometry builder* can construct one of the following geometries:

- a. **point geometry**: is a single coordinate geometry, the coordinate maybe x,y, or x,y,z. When this geometry is given to the FME feature, the feature's xml_type attribute is set to xml_point.
- b. **line geometry**: is a polyline, it contains at least 2 coordinates, the coordinates may be x,y, or x,y,z. When this geometry is given to the FME feature, the feature's xml_type attribute of the feature is set to xml_line.
- c. **area geometry**: is either a single closed polyline, a donut, or an aggregate of donuts/and or simple closed polylines. The coordinates may be x,y or x,y,z. When this geometry is given to the FME feature, the feature's xml_type attribute of the feature is set to xml_area.
- d. **homogeneous aggregate geometry**: is a collection of like geometries. A member of the aggregate may itself be an aggregate as long its xml_type matches. When this geometry is given to the FME feature, the feature's xml_type attribute of the feature is set to either xml_point, xml_line, or xml_area.
- e. **heterogenous aggregate geometry**: is a collection of geometries which may differ from one another. A member of this aggregate type may be any type of geometry. When this geometry is given to the FME feature, the features's xml_type attribute of the feature is set to xml_aggregate.
- f. **text geometry**: is a text value, a size and a rotation and a location. When this text geometry is given to the FME feature, the feature's xml_type attribute of the feature is set to xml_text.
- g. **path geometry**: a curve geometry composed of curve segments. A curve segment maybe a a line or an arc. When this path geometry is given to the FME feature, the feature's xml_type attribute of the feature is set to xml_line.
- h. **arc geometry**: an arc geometry is a circular or elliptical curve segment, see the **xml-arc** and **xml-elliptical-arc** geometry builders for the supported arc definitions. When this path geometry is given to the FME feature, the feature's xml_type attribute of the feature is set to xml_arc.
- i. **surface geometry**: is a surface geometry composed of three dimensional areas. When this surface geometry is given to the FME feature, the feature's xml_type attribute of the feature is set to xml_surface.
- j. **solid geometry**: is a solid geometry composed of three dimensional surfaces. When this solid geometry is given to the FME feature, the feature's xml_type attribute of the feature is set to xml_solid.

The <geometry> element has an activate attribute that specifies the name of a geometry builder. For a geometry builder to do anything useful (that is, construct a geometry), we must supply it with data, we can do this with the *geometry builder's data parameters*. The data is supplied through the <geometry> element's child <data> elements. The general form of a <geometry> element is:

```

<geometry activate="...">
  <data name="...">
    <!-- the value is some expression sequence -->
  </data>
  ...
  <data name="...">
    <!-- an expression sequence -->

```

```
</data>
</geometry>
```

In a *feature mapping rule* the `<geometry>` element must come after the `<feature-type>` and `<attributes>` elements.

Later in this section, the XML Reader's built-in *geometry builders* and their data parameters are described, custom geometry builders are described in their own sections. In order to understand how a geometry builder constructs the geometry of a FME feature we first explain the states of a geometry builder.

Geometry Builder States (activation, execution, suspension, and de-activation)

Similar to mapping rules a geometry builder may be activated, set on execution, suspended, and de-activated.

Let R be the mapping rule containing a `<geometry>` element having a `activate` attribute that specifies the geometry builder B , and

let E be the element whose start-tag is being read from the input XML document stream,

then:

- B is **activated** when R is activated, (recall that R is activated when R is the first mapping rule in an active-search-set matching E),
- B is **executing** as long as E is the context element,
- B is **suspended** when E is not the context element

Note: B if suspended, may again be set to the execution state. For example, when a mapping rule $R1$, matching a child element of E , say $E1$, contains a `<geometry>` element with no `activate` attribute. The `<geometry>` element in $R1$ then indicates to the XML Reader that B should again be executing and that the geometry data parameters in $R1$ should be given to B . This mechanism is provided for input XML documents that might structured their information in a segmented manner, say:

```
<line>
  <coord>2.9,3.4</coord>
  ...
</line>
```

so that a separate mapping rule must be defined to match each `<coord>` element to complete one polyline feature.

- B is **de-activated** after E is completely read.

Note: A geometry builder can only be executing or suspended if it is not de-activated.

Geometry Construction

It is possible to activate several geometry builders at a time. We can think of it as a stack of activated geometry builders, each builder is pushed into the stack when activated and popped when it is de-activated.

An activated geometry builder B constructs exactly one type of geometry. When B de-activates, exactly two things may happen:

- the geometry constructed by B is given to the FME feature:** this occurs only if B is also the last activated geometry builder.
- the geometry constructed by B is given to a previous geometry builder B_t :** When B de-activates and B is not the last activated geometry builder, then the top activated geometry builder in the stack, say B_t , receives and integrates the geometry constructed by B into its own. For this to be meaningful, B_t must be willing to accept a geometry constructed by another geometry builder.

Composite Geometry Builders

Geometry builders that can integrate another geometry into their own are called **composite geometry builders**, those that do not are called **non-composite**. Note that the composite, non-composite qualifiers do not refer to the type of geometry that the builder constructs, instead, they refer to whether or not the builder is capable of integrating another geometry into its own.

Usually, the builders that construct aggregate geometries are composite geometry builders. One can envisage a custom non-composite geometry builder that builds an aggregate geometry. For example, if a multi-point feature in the input XML document is defined to be something like:

```
<multi-point dimension="2">  
    2.3,34.9 445.4,34.34 23.35,345.453  
</multi-point>
```

A custom geometry builder used solely to build an aggregate of point features out of the <multi-point> element does not need to be a composite geometry builder, since it does not need to accept a point geometry from other builders because the information for all of the point components are embedded in the <multi-point> element's text content.

Built-in Geometry Builders

This section describes the built-in geometry builders available to the XML Reader. These custom geometry builders are format-specific and are described in their own sections. The XML Reader comes with the following built-in geometry builders:

- **xml-point**: is a non-composite builder that constructs a point geometry.
- **xml-line**: is non-composite builder that constructs a line geometry.
- **xml-area**: is a composite builder that constructs an area geometry.
- **xml-donut**: is a composite builder that constructs a donut area geometry.
- **xml-aggregate**: is a composite builder that constructs an aggregate geometry.
- **xml-box**: is a non-composite builder that constructs a polygon geometry from two coordinates.
- **xml-text**: is a composite builder that constructs a text geometry.
- **xml-path**: is a composite builder that constructs a path from line segments.
- **xml-arc**: and **xml-elliptical-arc**: are non-composite/composite builders that constructs arc geometry.
- **xml-circle**: is a non-composite builder that constructs an arc geometry based on a circle definition.
- **xml-polygon**: is a composite builder that constructs a polygon geometry.
- **xml-face**: is a composite builder that constructs a simple surface geometry.
- **xml-composite-surface**: is a composite builder that constructs a composite surface geometry
- **xml-enclosed-surface**: is a composite builder that constructs a simple solid geometry.
- **xml-composite-solid**: is a composite builder that constructs a composite solid geometry.
- **xml-multi-point**: is a composite builder that constructs a homegenous point aggregate geometry.
- **xml-multi-curve**: is a composite builder that constructs a homegenous curve aggregate geometry.
- **xml-multi-area**: is a composite builder that constructs a homegenous area aggregate geometry.
- **xml-multi-text**: is a composite builder that constructs an aggregate of text.
- **xml-multi-surface**: is a composite builder that constructs a homogenous surface geometry
- **xml-multi-solid**: is a composite builder that constructs a homogenous solid aggregate geometry.
- **xml-null**: is a non-composite builder that constructs the null geometry.
- **xml-reverse-geometry**: is a composite builder that reverses the order of coordinates.
- **fme-geometry**: is a non-composite builder that constructs various type of geometries from various types of geometry serialization.

The following sections describes each built-in builder and its data parameters.

xml-point

This is a non-composite builder that construct one point geometry object.

builder type: non-composite.

geometry constructed: point geometry.

data parameters:

Data Name	Value	Required/ Optional
data-string	The string containing the coordinate data. Range: String	Required
axis-order	Indicates the axis for a coordinate. Range: A permutation of the numbers 1..N, where N is the number of dimensions. Each number is separated by a comma ",". Default: "1,2,...,N"	Optional
dimension	The dimension of the coordinates. Range: String representing a positive integer, or the "implicit" literal, which forces the dimension to be automatically determined even if axis-order is specified. Default: When not specified the dimension will be automatically determined.	Optional
axis-separator	The string separating each axis of a coordinate in the data-string. Range: String "whitespace" this includes the tab, newline, and space characters. Default: ","	Optional
coord-separator	The string separating each coordinate in the data-string. Range: String Default: "whitespace" this includes the tab, newline, and space characters.	Optional
decimal	The string representing the decimal point for each real number in the data-string. Range: String Default: "."	Optional
coordinate-system	Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set. Range: String Default: ""	Optional
name	Specifies the geometry's name. Range: String	Optional

The following sequence of examples illustrates the usage of the xml-point builder.

Example 1

The following *xml_point1.xmp* xfMap document maps a <point> element from the *xml_point.xml* input XML document. Notice that the mapping rule matching the <point> element has a <geometry> element that specifies an *xml-point geometry builder*. We assume all the default values for the *xml-point* data parameters, so we only specify the required one, the *data-string* parameter.

xml_point1.xml

```
<?xml version="1.0"?>
<point id="0">0.945,78.970</point>
```

xml_point1.xmp

```
<?xml version="1.0"?>
<!DOCTYPE xfMap SYSTEM "xfMap.dtd">
<xfMap>
  <feature-map>
    <mapping match="point">
      <geometry activate="xml-point">
        <data name="data-string"> <extract expr="."/> </data>
      </geometry>
    </mapping>
  </feature-map>
</xfMap>
```

FME feature constructed:

```
+++++
Feature Type: '
Attribute: fme_geometry' has value fme_point'
Attribute: xml_type' has value xml_point'
Geometry Type: Point (1)
Number of Coordinates: 1 -- Coordinate Dimension: 2 -- Coordinate System: '
(0.945,78.97)
+++++
```

Example 2

This example is a little bit more complicated than the previous one. It illustrates several ways in which the *xml-point* data parameters may be used to map the information from the input XML elements into a point geometry. Please refer to the comments in the example for details.

xml_point2.xml

```
<?xml version="1.0"?>
<points>
  <point-A type="xy">8.8,2.5</point-A>
  <point-A type="xyz">97.97,92.5,-35.8</point-A>
  <point-A type="yxz">29,-77.9,0.0</point-A>

  <point-B>89,07 89,06 89,05</point-B>

  <point-C x="10.0" y="5.0"/>
</points>
```

xml_point2.xmp

```
<?xml version="1.0"?>
<!DOCTYPE xfMap SYSTEM "xfMap.dtd">

<xfMap>
  <feature-map>
    <mapping match="point-A[@type='xy' or @type='xyz']">
      <!-- This mapping rule activates when the start-tag of a
           point-A element having a type of xy or xyz is read. -->
      <feature-type>
        <literal expr="point-A-"/> <extract expr="@type"/>
```

```

        </feature-type>
        <!-- We use the default values for the other data parameters.
             Therefore we only need to supply the data-string parameter. -->
        <geometry activate="xml-point">
            <data name="data-string"> <extract expr="."/> </data>
        </geometry>
    </mapping>

    <mapping match="point-A[@type='xyz']">
        <!-- This mapping rule activates when the start-tag of a
             point-A element having a type xyz is read. -->
        <feature-type> <literal expr="point-A-xyz"/> </feature-type>
        <!-- The axis-order specifies the order of the axis of the
             coordinates in the data-string back to x,y,z order. -->
        <geometry activate="xml-point">
            <data name="data-string"> <extract expr="."/> </data>
            <data name="axis-order"> <literal expr="2,1,3"/> </data>
        </geometry>
    </mapping>

    <mapping match="point-B">
        <!-- Match the point-B element. -->
        <feature-type> <literal expr="point-B"/> </feature-type>
        <!-- Notice that the point-B element has as its decimal the comma
             character, and the axis separator is whitespace. We specify
             the dimension as well, because when the axis-separator
             and the coord-separator equal in value, then the dimension
             cannot be determine implicitly (both axis and
             coord (the default value) separator are whitespace). -->
        <geometry activate="xml-point">
            <data name="data-string"> <extract expr="."/> </data>
            <data name="dimension"> <literal expr="3"/> </data>
            <data name="decimal"> <literal expr=","/> </data>
            <data name="axis-separator">
                <literal expr="whitespace"/>
            </data>
        </geometry>
    </mapping>

    <mapping match="point-C">
        <!-- Match the point-C element. -->
        <feature-type> <literal expr="point-C"/> </feature-type>
        <geometry activate="xml-point">
            <!-- The xml-point builder parses a coordinate string. So
                 using an expression sequence we construct the coordinate
                 string such that it's axis-separator is the comma (which is
                 default value), and we give this expression sequence as
                 the value of the data-string parameter. -->
            <data name="data-string">
                <extract expr="@x"/> <literal expr=","/> <extract expr="@y"/
            </data>
        </geometry>
    </mapping>
</feature-map>
</xfMap>

```

FME features constructed:

```

+++++
Feature Type: point-A-xy'
Attribute: fme_geometry' has value fme_point'
Attribute: xml_type' has value xml_point'
Geometry Type: Point (1)
Number of Coordinates: 1 -- Coordinate Dimension: 2 -- Coordinate System: '
(8.8,2.5)
=====
+++++
Feature Type: point-A-xyz'

```

```

Attribute: fme_geometry' has value fme_point'
Attribute: xml_type' has value xml_point'
Geometry Type: Point (1)
Number of Coordinates: 1 -- Coordinate Dimension: 3 -- Coordinate System: '
(97.97,92.5,-35.8)
=====
+++++
Feature Type: point-A-xyz'
Attribute: fme_geometry' has value fme_point'
Attribute: xml_type' has value xml_point'
Geometry Type: Point (1)
Number of Coordinates: 1 -- Coordinate Dimension: 3 -- Coordinate System: '
(-77.9,29,0)
=====
+++++
Feature Type: point-B'
Attribute: fme_geometry' has value fme_point'
Attribute: xml_type' has value xml_point'
Geometry Type: Point (1)
Number of Coordinates: 1 -- Coordinate Dimension: 3 -- Coordinate System: '
(89.07,89.06,89.05)
=====
+++++
Feature Type: point-C'
Attribute: fme_geometry' has value fme_point'
Attribute: xml_type' has value xml_point'
Geometry Type: Point (1)
Number of Coordinates: 1 -- Coordinate Dimension: 2 -- Coordinate System: '
(10,5)
=====

```

xml-line

This is a composite builder that constructs a line geometry. The coordinates of the line may be given via the data-string parameter or via point geometries.

builder type: composite: accepts point geometries.

geometry constructed: line geometry.

data parameters:

Data Name	Value	Required/Optional
data-string	The string containing the coordinate data. Range: String	Optional
axis-order	Indicates the axis order for a coordinate. Range: A permutation of the numbers 1..N, where N is the number of dimensions. Each number is separated by a comma ",". Default: "1,2,...,N"	Optional
dimension	The dimension of the coordinates. Range: String representing a positive integer, or the "implicit" literal, which forces the dimension to be automatically determined even if axis-order is specified. Default: When not specified the dimension will be automatically determined.	Optional
axis-separator	The string separating each axis of a coordinate in the data-string. Range: String "whitespace" this includes the	Optional

Data Name	Value	Required/Optional
	tab, newline, and space characters. Default: “,”	
coord-separator	The string separating each coordinate in the data-string. Range: String Default: “whitespace” this includes the tab, newline, and space characters.	Optional
decimal	The string representing the decimal point for each real number in the data-string. Range: String Default: “.”	Optional
coordinate-system	Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set. Range: String Default: “.”	Optional
demote-incomplete-geometry	A line is incomplete if it doesn’t have at least two distinct points, if only one coordinate is given then the line geometry builder will halt on error, setting this optional parameter to true allows a point geometry to be constructed when only one coordinate is given. Range: false true Default: false	Optional
allow-incomplete-geometry	A line is incomplete if it doesn’t have at least two distinct points, setting this optional parameter to true allows a degenerate one coordinate line geometry to be constructed. Note that this parameter overrides the demote-incomplete-geometry parameter. Range: false true Default: false	Optional
name	Specifies the geometry’s name. Range: String	Optional

The document below contains two different type of line elements that we wish to map into FME features:

lines.xml

```
<?xml version="1.0"?>
<lines>
```

```

    <line1>
      0.0 0.0 1.0
      10.0 10.0 1.0
      20.0 30.0 1.0
    </line1>
    <line2 coords="50.5,50.4 30.8,15.2 0,0"/>
  </lines>

```

The following xfMap document uses the xml-line geometry builder:

xml_line.xmp

```

<?xml version="1.0"?>
<!DOCTYPE xfMap SYSTEM "xfMap.dtd">

<xfMap>
  <feature-map>
    <mapping match="line1">
      <feature-type <literal expr="line1"/> </feature-type>
      <geometry activate="xml-line">
        <data name="data-string"> <extract expr="."/> </data>
        <data name="dimension"> <literal expr="3"/> </data>
      </geometry>
    </mapping>

    <mapping match="line2">
      <feature-type <literal expr="line2"/> </feature-type>
      <geometry activate="xml-line">
        <data name="data-string">
          <extract expr="@coords"/>
        </data>
      </geometry>
    </mapping>
  </feature-map>
</xfMap>

```

When the *lines.xml* and *xml_line.xmp* documents are fed into the XML Reader the following FME features are output:

```

+++++
Feature Type: line1'
Attribute: fme_geometry' has value fme_line'
Attribute: xml_type' has value xml_line'
Geometry Type: Line (2)
Number of Coordinates: 3 -- Coordinate Dimension: 3 -- Coordinate System: '
(0,0,1) (10,10,1) (20,30,1)
=====
+++++
Feature Type: line2'
Attribute: fme_geometry' has value fme_line'
Attribute: xml_type' has value xml_line'
Geometry Type: Line (2)
Number of Coordinates: 3 -- Coordinate Dimension: 2 -- Coordinate System: '
(50.5,50.4) (30.8,15.2) (0,0)
=====

```

xml-area

The xml-area geometry builder constructs area geometries. The coordinates given to the data-string parameter must form a single closed polygon (first and last coord must equal).

Notice that all of the <data> elements are optional; that is, they do not need to appear in the <geometry> element. When this is the case, this builder cannot build an area geometry unless the XML Reader at some point passes other area geometries (either simple polygons, a donuts, or an aggregate of areas) to it; the builder geometrically integrates all of these area geometries into one area geometry, for example, if 3 area geometries (say 3 simple polygons: 1 outer shell, and 2 holes) are given to it in any arbitrary order, the builder will geometrically integrate the 3 areas into one donut geometry.

builder type: composite - but it only accepts area geometries.

geometry constructed: area geometry.

data parameters:

Data Name	Value	Required/Optional
data-string	The string containing the coordinate data. Range: String	Optional
axis-order	Indicates the axis order for a coordinate. Range: A permutation of the numbers 1..N, where N is the number of dimensions. Each number is separated by a comma ",". Default: "1,2,...,N"	Optional
dimension	The dimension of the coordinates. Range: String representing a positive integer, or the "implicit" literal, which forces the dimension to be automatically determined even if axis-order is specified. Default: When not specified the dimension will be automatically determined.	Optional
axis-separator	The string separating each axis of a coordinate in the data-string. Range: String "whitespace" this includes the tab, newline, and space characters. Default: ","	Optional
coord-separator	The string separating each coordinate in the data-string. Range: String Default: "whitespace" this includes the tab, newline, and space characters.	Optional
coordinate-system	Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set. Range: String Default: ""	Optional
demote-incomplete-geometry	An area is incomplete if it does not have at least three distinct points. If only one or two coordinates are given then the area geometry builder will halt on error. Setting this optional parameter to true allows a point geometry to be constructed when only one coordinate is given and a line geometry to be	Optional

Data Name	Value	Required/Optional
	constructed when only two coordinates are given. Range: false true Default: false	
aggregate-to-multi	Specifies whether an aggregate of areas should be converted into a multi area. Range: false true Default: false	Optional
name	Specifies the geometry's name. Range: String	Optional

example 1, building simple polygons

The following `xml_area_simple_polygon.xmp` document instructs the XML Reader to output two FME features that are simple closed polygons from the input `polygons.xml` document.

polygons.xml

```
<?xml version="1.0"?>
<polygons>
  <polygon>60.0,60.0 80.0,60.0 80.0,80.0 60.0,80.0 60.0,60.0</polygon>
  <polygon>40.0,40.0 50.0,40.0 50.0,50.0 40.0,50.0 40.0,40.0</polygon>
</polygons>
```

xml_area_simple_polygon.xmp

```
<?xml version="1.0"?>
<!DOCTYPE xfMap SYSTEM "xfMap.dtd">

<xfMap>
  <feature-map>
    <mapping match="polygon">
      <feature-type> <literal expr="simple polygon"/> </feature-type>
      <geometry activate="xml-area">
        <data name="data-string"> <extract expr="."/> </data>
      </geometry>
    </mapping>
  </feature-map>
</xfMap>
```

FME features constructed:

```
+++++
Feature Type: simple closed polygon'
Attribute: fme_geometry' has value fme_polygon'
Attribute: xml_type' has value xml_area'
Geometry Type: Polygon (4)
Number of Coordinates: 5 -- Coordinate Dimension: 2 -- Coordinate System: '
(60,60) (80,60) (80,80) (60,80) (60,60)
=====
+++++
Feature Type: simple closed polygon'
Attribute: fme_geometry' has value fme_polygon'
Attribute: xml_type' has value xml_area'
Geometry Type: Polygon (4)
Number of Coordinates: 5 -- Coordinate Dimension: 2 -- Coordinate System: '
(40,40) (50,40) (50,50) (40,50) (40,40)
=====
```

example 2, building donuts

The *donuts.xml* document contains an element representing a donut. In this example we show how to use the *xml-area* geometry builder, in the *xml_area_donut.xmp*, to make the XML Reader output an FME donut feature. The *xfMap* document has two feature mapping rules matching the `<donut>` and `<polygon>` elements. Note that the `<outershell>` and `<hole>` elements are ignored by the mapping rules; this distinction is not necessary for the construction of the area-donut geometry, since the *xml-area* builder geometrically determines which polygon is an outer boundary or a hole.

The *xml-area* geometry builder that is activated in the feature mapping rule matching the `<donut>` element just waits for geometry builders to pass it area geometries so that it can geometrically integrate them into one donut feature.

donuts.xml

```
<?xml version="1.0"?>
<donuts>
  <donut>
    <outer-shell>
      <polygon>0.0,0.0 100.0,0.0 100.0,100.0 0.0,100.0 0.0,0.0</polygon>
    </outer-shell>
    <hole>
      <polygon>60.0,60.0 80.0,60.0 80.0,80.0 60.0,80.0 60.0,60.0</polygon>
    </hole>
    <hole>
      <polygon>40.0,40.0 50.0,40.0 50.0,50.0 40.0,50.0 40.0,40.0</polygon>
    </hole>
  </donut>
</donuts>
```

xml_area_donut.xmp

```
<?xml version="1.0"?>
<!DOCTYPE xfMap SYSTEM "xfMap.dtd">

<xfMap>
  <feature-map>
    <mapping match="donut">
      <feature-type> <literal expr="donut"/> </feature-type>
      <!-- Note that no data parameters are required for the
           xml-area geometry builder since its whole purpose
           is to sit and wait for other builders to hand it
           area geometries. -->
      <geometry activate="xml-area"/>
    </mapping>
  </feature-map>

  <feature-content-map>
    <mapping match="polygon">
      <!-- This xml-area geometry builder uses the content of the polygon
           element to construct a single closed polygon area geometry.
           When this geometry builder de-activates the area geometry is
           handed over to the top geometry builder on the stack (in this
           case it is the xml-area that was activated in the mapping rule
           above. -->
      <geometry activate="xml-area">
        <data name="data-string"> <extract expr="."/> </data>
      </geometry>
    </mapping>
  </feature-content-map>
</xfMap>
```

FME feature constructed:

```
+++++
Feature Type: donut'
Attribute: fme_geometry' has value fme_donut'
Attribute: xml_type' has value xml_area'
```

```

Geometry Type: Donut (8)
Total Number of Coords: 15 -- Coordinate Dimension: 2 -- Coordinate System: '
Number of Shells: 3
Outer Shell -- Part Number: 0 -- Number of Coordinates: 5
(0,0) (100,0) (100,100) (0,100) (0,0)
Inner Shell -- Part Number: 1 -- Number of Coordinates: 5
(40,40) (50,40) (50,50) (40,50) (40,40)
Inner Shell -- Part Number: 2 -- Number of Coordinates: 5
(60,60) (80,60) (80,80) (60,80) (60,60)
=====

```

xml-donut

The xml-donut geometry builder is a composite geometry builder that may construct either donut or polygon geometries. The xml-donut geometry builder does nothing on its own, when activated it just waits and accepts either curve or polygon geometries that form the outer and inner boundaries of the donut. The outer boundary of the donut is the first geometry accepted, subsequent geometries are set as inner boundaries.

builder type: composite - it accepts polygon and curve geometries.

geometry constructed: area geometry.

data parameters:

Data Name	Value	Required/Optional
keep-as-donut	If the value for this data parameter is set to false and if only the outer boundary of the donut is set, then a polygon, rather than a donut, will be constructed. Range: false true Default: true	Optional
coordinate-system	Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set. Range: String Default: "."	Optional
demote-incomplete-geometry	A donut is incomplete if it does not have the outer boundary set. If there are no polygon or curve geometries given, the donut geometry builder will halt on error. Setting this optional parameter to true allows a non-geometrical object to be constructed when no geometries are given. Range: false true Default: false	Optional
name	Specifies the geometry's name. Range: String	Optional

xml-aggregate

The xml-aggregate geometry builder builds aggregate objects. This builder does not require any data parameters, so the only way for it to build an aggregate geometry is from existing geometry objects. This means that before the xml-

aggregate builder is de-activated, other geometry builders should have been activated and de-activated, moreover, their constructed geometries should have been passed to it.

builder type: composite - accepts any type of geometry.

geometry constructed: aggregate.

data parameters:

Data Name	Value	Required/Optional
xml_type	<p>This data parameter applies only if an aggregate was constructed. This data parameter sets the xml_type for the feature under construction. Note that this xml_type may be overwritten by geometry builders or attribute rules from mapping rules that are activated before the one containing the activation of the current aggregate builder.</p> <p>Range: xml_point xml_line xml_area xml_aggregate</p> <p>Default: xml_aggregate</p>	Optional
keep-as-aggregate	<p>When the value for this data parameter is set to "false" and the xml-aggregate geometry consists solely of one feature, then an aggregate geometry will not be constructed.</p> <p>Range: false true</p> <p>Default: true</p>	Optional
contains-individual-geometries	<p>When the value for this data parameter is set to "true" then a flag will be set on the aggregate signaling that the aggregate is merely a placeholder containing individually named geometries; e.g., in a destination system each named geometry component may be interpreted as a separate geometry column in a table.</p> <p>Range: false true</p> <p>Default: false</p>	Optional
coordinate-system	<p>Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set.</p> <p>Range: String</p> <p>Default: "."</p>	Optional
name	<p>Specifies the geometry's name.</p> <p>Range: String</p>	Optional

The following example shows how the xml-aggregate geometry builder is used in the *xml_aggregate.xmp* to construct a geometry that is an aggregate of point geometries. The previous, *points1.xml*, is used as the input XML document and is reproduced below for convenience:

points1.xml

```
<?xml version="1.0"?>
<points>
  <point name="myPoint" num="0">
    <color>
      <red>0.324</red>
      <green>0.233</green>
      <blue>0.596</blue>
    </color>
    <location x="10.0" y="0.0"/>
  </point>
  <point name="myPoint" num="1">
    <color>
      <red>0.874</red>
      <green>0.948</green>
      <blue>0.554</blue>
    </color>
    <location x="5.0" y="5.0"/>
  </point>
</points>
```

xml_aggregate.xmp

```
<?xml version="1.0"?>
<!DOCTYPE xfMap SYSTEM "xfMap.dtd">

<xfMap>
  <feature-map>
    <mapping match="points">
      <feature-type> <literal expr="points"/> </feature-type>
      <!-- This geometry builder integrates all the
            geometries that it receives into one aggregate
            geometry -->
      <geometry activate="xml-aggregate">
        <data name="xml_type">
          <literal expr="xml_point"/>
        </data>
      </geometry>
    </mapping>
  </feature-map>

  <feature-content-map>
    <mapping match="point">
      <geometry activate="xml-point">
        <data name="data-string">
          <extract expr="./location[@x]"/>
          <literal expr=","/>
          <extract expr="./location[@y]"/>
        </data>
      </geometry>
    </mapping>
  </feature-content-map>
</xfMap>
```

FME feature constructed:

```
+++++
Feature Type: points'
Attribute: fme_geometry' has value fme_aggregate'
Attribute: xml_type' has value xml_point'
Geometry Type: Aggregate (512)
Total Number of Coords: 2 -- Coordinate Dimension: 2 -- Coordinate System: '
```

Number of Aggregate Parts: 2

Geometry Type: Point (1) -- Part Number: 0 -- Number of Coordinates: 1
(10,0)

Geometry Type: Point (1) -- Part Number: 1 -- Number of Coordinates: 1
(5,5)
=====

xml-box

This is a non-composite builder that construct a polygon geometry out of two coordinates.

builder type: non-composite.

geometry constructed: polygon geometry.

data parameters:

Data Name	Value	Required/Optional
data-string	The string containing the coordinate data. Range: String	Required
axis-order	Indicates the axis for a coordinate. Range: A permutation of the numbers 1..N, where N is the number of dimensions. Each number is separated by a comma ",". Default: "1,2,...,N"	Optional
dimension	The dimension of the coordinates. Range: String representing a positive integer, or the "implicit" literal, which forces the dimension to be automatically determined even if axis-order is specified. Default: When not specified the dimension will be automatically determined.	Optional
axis-separator	The string separating each axis of a coordinate in the data-string. Range: String "whitespace" this includes the tab, newline, and space characters. Default: ","	Optional
coord-separator	The string separating each coordinate in the data-string. Range: String Default: "whitespace" this includes the tab, newline, and space characters.	Optional
decimal	The string representing the decimal point for each real number in the data-string. Range: String Default: "."	Optional

Data Name	Value	Required/Optional
coordinate-system	Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set. Range: String Default: "."	Optional
name	Specifies the geometry's name. Range: String	Optional

The following sequence of examples illustrates the usage of the xml-box builder.

box.xml

```
<?xml version="1.0"?>
<boxes>
  <box>60.0,60.0 80.0,80.0</box>
  <box>40.0,40.0 50.0,50.0</box>
</boxes>
```

box.xmp

```
<?xml version="1.0"?>
<!DOCTYPE xfMap SYSTEM "xfMap.dtd">

<xfMap>
  <feature-map>
    <mapping match="box">
      <feature-type> <literal expr="boxed-polygon"/> </feature-type>
      <geometry activate="xml-box">
        <data name="data-string"> <extract expr="."/> </data>
      </geometry>
    </mapping>
  </feature-map>
</xfMap>
```

FME features constructed:

```
+++++
Feature Type: boxed-polygon'
Attribute: fme_geometry' has value fme_polygon'
Attribute: xml_type' has value xml_area'
Geometry Type: Polygon (4)
Number of Coordinates: 5 -- Coordinate Dimension: 2 -- Coordinate System: '
(60,60) (80,60) (80,80) (60,80) (60,60)
=====
+++++
Feature Type: boxed-polygon'
Attribute: fme_geometry' has value fme_polygon'
Attribute: xml_type' has value xml_area'
Geometry Type: Polygon (4)
Number of Coordinates: 5 -- Coordinate Dimension: 2 -- Coordinate System: '
(40,40) (50,40) (50,50) (40,50) (40,40)
=====
```

xml-text

This is a composite builder that construct a text geometry. If required, the xml-text builder can be used as a non-composite xml-text builder, in which case, in addition to the data parameters below, one will also need to use the data parameters for an xml-point (see above). If used as a composite builder, it must be given an existing geometry (in fact, this is the preferred way of using the xml-text builder). In other words, another geometry must be activated and deactivated before the xml-text is deactivated.

builder type: composite

geometry constructed: point, or depends on builder

data parameters:

Data Name	Value	Required/Optional
text-string	The string containing text for the text geometry Range: String	Required
text-rotation	The orientation of the text in decimal degrees Default: String representing a real number	Required
text-size	The size of the text Range: String representing a positive integer.	Required
name	Specifies the geometry's name. Range: String	Optional

xml-path

The xml-path geometry builder constructs linear paths. This builder does not require any data parameters, so the only way for it to build a path geometry is from existing curve or point geometries. This means that before the xml-path builder is de-activated, other geometry builders that construct curves or points, such as xml-line, xml-arc or xml-point, should have been activated and de-activated, moreover, their constructed geometries should have been passed to it.

The xml-path geometry builder also accepts null geometries. Currently, these null geometries must have the "radius" and "direction" traits set. The null "arc" geometry in a path must be preceded and succeeded by a "curve" or "point" which correspond to the start and end point of the arc, respectively.

The xml-path geometry builder can return a polygon geometry when its convert-to-polygon data parameter is set to true. In this case the builder sets the constructed path as the boundary of the polygon.

builder type: composite - accepts curve, point and appropriately tagged null geometries.

geometry constructed: line geometry, point, or area geometry.

data parameters:

Data Name	Value	Required/Optional
demote-incomplete-geometry	A path needs at least one curve segment, setting this optional parameter to true avoids empty paths to be constructed by demoting a path with zero segments into a non-geometrical object.	Optional

Data Name	Value	Required/Optional
	Range: false true Default: true	
keep-as-path	When the value for this data parameter is set to "false" and the xml-path geometry consists solely of one feature, then a path geometry will not be constructed. A path with a one coordinate line segment is returned as a point. Range: false true Default: true	Optional
demote-single-arc	When the value for this data parameter is set to "true" and the xml-path geometry consists solely of one arc feature, then a path geometry will not be constructed. If the single feature is not an arc, handling will depend on the keep-as-path parameter. Range: false true Default: true	Optional
convert-to-polygon	This optional data parameter directs the xml-path geometry builder to return a polygon geometry with the constructed path set as the polygon's boundary. Range: false true Default: false	Optional
coordinate-system	Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set. Range: String Default: "."	Optional
name	Specifies the geometry's name. Range: String	Optional

xml-arc

This is a non-composite and composite builder that constructs a circular arc geometry. Currently the following arc definitions are supported:

- a. A circular arc defined by 3 coincident control points.
- b. A circular arc defined by a circle's center, start and end point and a direction.
- c. A circular arc defined by a start and end point, a radius and a direction.

builder type: non-composite/composite, accepts point geometries in composite mode.

geometry constructed: arc geometry.

data parameters:

Data Name	Value	Required/Optional
data-string	<p>a. For an arc defined by 3 coincident control points, the coordinate data denotes the coincident points in the order of arc traversal.</p> <p>The coordinate data may be received as point geometries, the order of the point geometries are assumed to be the order of arc traversal.</p> <p>b. The coordinate data string denotes the center, start and end point of an arc when the "direction" data parameter is present.</p> <p>The coordinate data may be received as point geometries. These control points maybe named or unnamed. The named point geometries maybe received in any order, there names must be one of "center", "start" or "end", or any of their equivalents, see below. If the point geometries are unnamed, then the order of the points is significant, the first point received is assumed as the center point, the next point as the start point, and the last point as the end point of the arc.</p> <p>c. The coordinate data string denotes the start and end point of an arc when the "direction" and "radius" data parameters are present</p> <p>The coordinate data may be received as point geometries. These control points maybe named or unnamed. The named point geometries maybe received in any order, there names must be one of "start" or "end", or any of their equivalents, see below. If the point geometries are unnamed, then the order of the points is significant, the first point received is assumed as the start point, the next point as the end point of the arc.</p> <p>Equivalent control point names: The following names maybe substituted for "center", "start" and "end":</p> <ul style="list-style-type: none"> • center: CentrePoint • start: FromPoint • end: ToPoint <p>Range: String</p>	Optional if the control points are to be passed in as point geometries; otherwise Required
direction	<p>Specifies the traversal direction from the start to the end point of an arc.</p> <p>Range: cw ccw</p> <p>Default: There are no default values.</p>	Required if the data-string is to specify the center, start and end points of an arc

Data Name	Value	Required/ Optional
radius	<p>The radius of the circular arc.</p> <p>Range: A positive real number.</p> <p>Default: There are no default values.</p>	Required, along with the direction, if the data-string specifies the start and end point of an arc.
axis-order	<p>Indicates the axis for a coordinate.</p> <p>Range: A permutation of the numbers 1..N, where N is the number of dimensions.</p> <p>Each number is separated by a comma ",".</p> <p>Default: "1,2,...,N"</p>	Optional
dimension	<p>The dimension of the coordinates.</p> <p>Range: String representing a positive integer, or the "implicit" literal, which forces the dimension to be automatically determined even if axis-order is specified.</p> <p>Default: When not specified the dimension will be automatically determined.</p>	Optional
axis-separator	<p>The string separating each axis of a coordinate in the data-string.</p> <p>Range: String "whitespace" this includes the tab, newline, and space characters.</p> <p>Default: ","</p>	Optional
coord-separator	<p>The string separating each coordinate in the data-string.</p> <p>Range: String</p> <p>Default: "whitespace" this includes the tab, newline, and space characters.</p>	Optional
decimal	<p>The string representing the decimal point for each real number in the data-string.</p> <p>Range: String</p> <p>Default: "."</p>	Optional
coordinate-system	<p>Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set.</p> <p>Range: String</p> <p>Default: "."</p>	Optional
number-of-arcs	<p>The data parameter is optional and is only applicable</p>	Optional

Data Name	Value	Required/ Optional
	<p>when an arc is defined by control points coincident to it. If the value given is greater than 1, then the number of coordinates in the data-string must be $(2*\text{number-of-arcs} + 1)$, each consecutive arc constructed share a common control point, and the geometry returned from these consecutive arcs cannot be an arc anymore, instead, a path geometry is returned.</p> <p>Range: a positive number Default: 1</p>	
demote-incomplete-geometry	<p>This optional data parameter allows the xml-arc geometry builder to return a non-geometrical object when no data-string is given or when the value of the data-string is the empty string.</p> <p>Range: false true Default: false</p>	Optional
incomplete-geometry-traits	<p>This optional data parameter takes effect only when the demote-incomplete-geometry data parameter is set to true. This parameter lists the xml-arc data parameters that should be loaded into the demoted arc geometry as traits. For example, when this geometry builder fails to construct an arc and demote-incomplete-geometry is set to true and the value for this parameter is set to "radius direction", then the builder will construct a null geometry with radius and direction traits.</p> <p>Range: whitespace separated list of arc data parameter names Default: There are no default values.</p>	Optional
name	<p>Specifies the geometry's name.</p> <p>Range: String</p>	Optional

xml-elliptical arc

This is a non-composite and composite builder that constructs an elliptical arc geometry. The following elliptical arc definition is supported:

- a. An elliptical arc defined by a circle's center, start and end point, primary-radius, secondary-radius, direction, rotation, and rotation-direction.

builder type: non-composite/composite, accepts point geometries in composite mode.

geometry constructed: arc geometry.

data parameters:

Data Name	Value	Required/ Optional
data-string	<p>a. The coordinate data string denotes the center, start and end point of an arc when the "direction" data parameter is present.</p> <p>The coordinate data may be received as point geometries. These control points maybe named or unnamed. The named point geometries maybe received in any order, there names must be one of "center", "start" or "end", or any of their equivalents, see below. If the point geometries are unnamed, then the order of the points is significant, the first point received is assumed as the center point, the next point as the start point, and the last point as the end point of the arc.</p> <p>Equivalent control point names: The following names maybe substituted for "center", "start" and "end":</p> <ul style="list-style-type: none"> • center: CentrePoint • start: FromPoint • end: ToPoint <p>Range: String</p>	Optional if the control points are to be passed in as point geometries; otherwise Required
primary-radius	<p>The length of the primary axis for the ellipse the arc is based upon.</p> <p>Range: A positive real number.</p> <p>Default: There are no default values.</p>	Required
secondary-radius	<p>The length of the secondary axis for the ellipse the arc is based upon.</p> <p>Range: A positive real number.</p> <p>Default: There are no default values.</p>	Required
direction	<p>Specifies the traversal direction from the start to the end point the elliptical arc.</p> <p>Range: cw ccw</p> <p>Default: There are no default values.</p>	Required
rotation	<p>The rotation of the ellipse that defines the arc. The rotation angle specifies the angle in degrees from the horizontal axis to the primary axis in the direction specified by the rotation-direction data parameter.</p> <p>Range: A positive real number.</p> <p>Default: 0.0</p>	Optional
rotation-direction	<p>Specifies the rotation direction. See the rotation data parameter.</p> <p>Range: cw ccw</p>	Required

Data Name	Value	Required/ Optional
	Default: ccw	
axis-order	Indicates the axis for a coordinate. Range: A permutation of the numbers 1..N, where N is the number of dimensions. Each number is separated by a comma ",". Default: "1,2,...,N"	Optional, used only if data-string is specified
dimension	The dimension of the coordinates. Range: String representing a positive integer, or the "implicit" literal, which forces the dimension to be automatically determined even if axis-order is specified. Default: When not specified the dimension will be automatically determined.	Optional, used only if data-string is specified
axis-separator	The string separating each axis of a coordinate in the data-string . Range: String "whitespace" – this includes the tab, newline, and space characters. Default: ","	Optional, used only if data-string is specified
coord-separator	The string separating each coordinate in the data-string . Range: String Default: "whitespace" – this includes the tab, newline, and space characters.	Optional, used only if data-string is specified
decimal	The string representing the decimal point for each real number in the data-string. Range: String Default: "."	Optional, used only if data-string is specified
coordinate-system	Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set. Range: String Default: "."	Optional
demote-incomplete-geometry	This optional data parameter allows the xml-arc geometry builder to return a non-geometrical object when no data-string is given or when the value of the data-string is the empty string. Range: false true Default: false	Optional

Data Name	Value	Required/Optional
incomplete-geometry-traits	This optional data parameter takes effect only when the demote-incomplete-geometry data parameter is set to true. This parameter lists the xml-arc data parameters that should be loaded into the demoted arc geometry as traits. For example, when this geometry builder fails to construct an arc and demote-incomplete-geometry is set to true and the value for this parameter is set to "radius direction", then the builder will construct a null geometry with radius and direction traits. Range: whitespace separated list of arc data parameter names Default: There are no default values.	Optional
name	Specifies the geometry's name. Range: String	Optional

xml-circle

This is a non-composite and composite builder that constructs a circular arc geometry. The following circle definition is supported:

- a. A circle defined by three distinct control points that are coincident to the circle.
- b. A circle defined by a center point and a radius.

builder type: non-composite

geometry constructed: arc geometry.

data parameters:

Data Name	Value	Required/Optional
data-string	<ol style="list-style-type: none"> a. The coordinate data string denoting the coincident points in the order of circle traversal. b. The coordinate data string denoting the circle's center point. Range: String	Required
radius	The radius of the circle. Range: A positive, real number Default: There are no default values.	Required, if the data-string specifies the center point of the circle.

Data Name	Value	Required/Optional
axis-order	Indicates the axis for a coordinate. Range: A permutation of the numbers 1..N, where N is the number of dimensions. Each number is separated by a comma ",". Default: "1,2,...,N"	Optional, used only if data-string is specified
dimension	The dimension of the coordinates. Range: String representing a positive integer, or the "implicit" literal, which forces the dimension to be automatically determined even if axis-order is specified. Default: When not specified the dimension will be automatically determined.	Optional, used only if data-string is specified
axis-separator	The string separating each axis of a coordinate in the data-string . Range: String "whitespace" this includes the tab, newline, and space characters. Default: ","	Optional, used only if data-string is specified
coord-separator	The string separating each coordinate in the data-string . Range: String Default: "whitespace" – this includes the tab, newline, and space characters.	Optional, used only if data-string is specified
decimal	The string representing the decimal point for each real number in the data-string. Range: String Default: "."	Optional, used only if data-string is specified
coordinate-system	Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set. Range: String Default: ""	Optional
name	Specifies the geometry's name. Range: String	Optional

xml-polygon

The xml-polygon geometry builder constructs a polygon from a closed curve geometry. This is a composite builder. The xml-polygon geometry builder does nothing on its own, when activated, it will wait and accept points and curves. The points and curves are interpreted into a path which becomes the boundary of the polygon. The curve is assumed to be closed, if it is not, then closure will be assumed as a straight line from the start point to the end point.

builder type: composite - accept curve and point geometries.

geometry constructed: area geometry.

data parameters:

Data Name	Value	Required/Optional
coordinate-system	Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set. Range: String Default: ""	Optional
demote-incomplete-geometry	A polygon is incomplete if it doesn't have at least one curve or point segment. If no segments are given then the polygon geometry builder will halt on error. Setting this optional parameter to true allows a non-geometrical object to be constructed when no segments are given. Range: false true Default: false	Optional
name	Specifies the geometry's name. Range: String	Optional

xml-face

The xml-face geometry builder builds simple surface geometries. This builder does not require any data parameters, and will construct the simple surface geometry from existing area geometries. Thus, before the xml-face geometry builder is deactivated, other geometry builders should have been activated, passing their constructed areas to the xml-face geometry builder.

Each of the areas passed to the xml-face geometry builder should share a common plane. The first area accepted forms the outer boundary, and subsequent areas accepted form inner gaps.

builder type: composite - accepts area geometries

geometry constructed: simple surface

data parameters:

Data Name	Value	Required/Optional
demote-incomplete-geometry	This Boolean parameter determines if incomplete geometry should be demoted instead of producing an error. If set to true, whenever the geometry builder receives no areas to create a simple surface from, the simple surface will be downgraded to a null geometry. If set to false, whenever the geometry builder receives no areas to create a simple surface from, it will raise an error and terminate the translation.	Optional

Data Name	Value	Required/Optional
	Range: true false Default: false	
coordinate-system	Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set. Range: String Default: "."	Optional
name	Specifies the geometry's name. Range: String	Optional

Example: Building a simple surface

simplesurface.xml

```
<?xml version="1.0"?>
<simplesurface>
  <area>
    0.0,0.0,0.0 5.0,0.0,5.0 5.0,5.0,10.0 0.0,5.0,5.0 0.0,0.0,0.0
  </area>
  <area>
    2.0,2.0,4.0 3.0,2.0,5.0 3.0,3.0,6.0 2.0,3.0,5.0 2.0,2.0,4.0
  </area>
</simplesurface>
```

simplesurface.xmp

```
<?xml version="1.0"?>
<xfMap>
  <feature-map>
    <mapping match="simplesurface">
      <feature-type> <literal expr="simplesurface" /> </feature-type>
      <geometry activate="xml-face" />
    </mapping>
  </feature-map>
  <feature-content-map>
    <mapping match="area">
      <geometry activate="xml-area">
        <data name="data-string"> <extract expr="." /> </data>
      </geometry>
    </mapping>
  </feature-content-map>
</xfMap>
```

xml-composite-surface

The xml-composite-surface geometry builder builds composite surface geometries. This builder does not require any data parameters, and will construct the composite surface geometry from existing surface geometries. Thus, before the xml-composite-surface geometry builder is deactivated, other geometry builders should have been activated, passing their constructed surfaces to the xml-composite-surface geometry builder.

Each of the surfaces passed to the xml-composite-surface geometry builder should be topologically connected along their boundaries.

builder type: composite - accepts surface geometries

geometry constructed: composite surface

data parameters:

The following demote-incomplete-geometry and allow-empty-composite data parameters change how the geometry is handled in error conditions. If both of the data parameters are set to false, whenever the geometry builder receives no simple surfaces to create a composite surface from, it will raise an error and terminate the translation.

Data Name	Value	Required/Optional
demote-incomplete-geometry	This boolean parameter determines if incomplete geometry should be demoted instead of producing an error. If set to true, whenever the geometry builder receives no simple surfaces to create a composite surface from, the composite surface will be downgraded to a null geometry. Range: true false Default: false	Optional
allow-empty-composite	This boolean parameter determines if incomplete geometry should be left incomplete instead of producing an error. If set to true, whenever the geometry builder receives no simple surfaces to create a composite surface from, the composite surface will be left empty and returned. Range: true false Default: false	Optional
coordinate-system	Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set. Range: String Default: "."	Optional
name	Specifies the geometry's name. Range: String	Optional

Example: Building a composite surface

compositesurface.xml

```

<?xml version="1.0"?>
<compositesurface>
  <simplesurface>
    <area>
      0.0,0.0,0.0 5.0,0.0,5.0 5.0,5.0,10.0 0.0,5.0,5.0 0.0,0.0,0.0
    </area>
    <area>
      2.0,2.0,4.0 3.0,2.0,5.0 3.0,3.0,6.0 2.0,3.0,5.0 2.0,2.0,4.0
    </area>
  </simplesurface>
  <simplesurface>
    <area>

```

```

                                0.0,0.0,0.0 5.0,0.0,0.0 5.0,0.0,5.0 0.0,0.0,0.0
                                </area>
                                </simplesurface>
</compositesurface>

```

compositesurface.xmp

```

<?xml version="1.0"?>
<xfMap>
  <feature-map>
    <mapping match="compositesurface">
      <feature-type>
        <literal expr="compositesurface" />
      </feature-type>
      <geometry activate="xml-composite-surface" />
    </mapping>
  </feature-map>
  <feature-content-map>
    <mapping match="simplesurface">
      <geometry activate="xml-face" />
    </mapping>
    <mapping match="area">
      <geometry activate="xml-area">
        <data name="data-string"> <extract expr="." /> </data>
      </geometry>
    </mapping>
  </feature-content-map>
</xfMap>

```

xml-enclosed-surface

The xml-enclosed-surface geometry builder builds simple solid geometries. This builder does not require any data parameters, and will construct the simple solid geometry from existing surface geometries. Thus, before the xml-enclosed-surface geometry builder is deactivated, other geometry builders should have been activated, passing their constructed surfaces to the xml-enclosed-surface geometry builder.

Each of the surfaces passed to the xml-enclosed-surface geometry builder should fully enclose a volume of space. The first surface accepted forms the outer boundary, and subsequent areas accepted form inner voids.

builder type: composite - accepts surface geometries

geometry constructed: simple solid

data parameters:

Data Name	Value	Required/Optional
demote-incomplete-geometry	This boolean parameter determines if incomplete geometry should be demoted instead of producing an error. If set to true, whenever the geometry builder receives no surfaces to create a simple solid from, the simple solid will be downgraded to a null geometry. If set to false, whenever the geometry builder receives no surfaces to create a simple solid from, it will raise an error and terminate the translation. Range: true false Default: false	Optional
coordinate-system	Specifies the coordinate system name. If the	Optional

Data Name	Value	Required/Optional
	expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set. Range: String Default: "."	
name	Specifies the geometry's name. Range: String	Optional

Example: Building a simple solid

simplesolid.xml

```

<?xml version="1.0"?>
<simplesolid>
  <compositesurface>
    <simplesurface>
      <area>0.0,0.0,0.0 0.0,1.0,0.0 1.0,0.0,0.0 0.0,0.0,0.0</area>
    </simplesurface>
    <simplesurface>
      <area>0.0,0.0,0.0 1.0,0.0,0.0 1.0,1.0,1.0 0.0,0.0,0.0</area>
    </simplesurface>
    <simplesurface>
      <area>0.0,0.0,0.0 1.0,1.0,1.0 0.0,1.0,0.0 0.0,0.0,0.0</area>
    </simplesurface>
    <simplesurface>
      <area>1.0,0.0,0.0 0.0,1.0,0.0 1.0,1.0,1.0 1.0,0.0,0.0</area>
    </simplesurface>
  </compositesurface>
</simplesolid>

```

simplesolid.xmp

```

<?xml version="1.0"?>
<xfMap>
  <feature-map>
    <mapping match="simplesolid">
      <feature-type> <literal expr="simplesolid" /> </feature-type>
      <geometry activate="xml-enclosed-surface" />
    </mapping>
  </feature-map>
  <feature-content-map>
    <mapping match="compositesurface">
      <geometry activate="xml-composite-surface" />
    </mapping>
    <mapping match="simplesurface">
      <geometry activate="xml-face" />
    </mapping>
    <mapping match="area">
      <geometry activate="xml-area">
        <data name="data-string"> <extract expr="." /> </data>
      </geometry>
    </mapping>
  </feature-content-map>
</xfMap>

```

xml-composite-solid

The xml-composite-solid geometry builder builds composite solid geometries. This builder does not require any data parameters, and will construct the composite solid geometry from existing simple solid geometries. Thus, before the xml-composite-solid geometry builder is deactivated, other geometry builders should have been activated, passing their constructed simple solids to the xml-composite-solid geometry builder.

Each of the simple solids passed to the xml-composite-solid geometry builder should be topologically connected along their boundaries.

builder type: composite - accepts simple solid geometries

geometry constructed: composite solid

data parameters:

The following demote-incomplete-geometry and allow-empty-composite data parameters change how the geometry is handled in error conditions. If both of the data parameters are set to false, whenever the geometry builder receives no simple solids to create a composite solid from, it will raise an error and terminate the translation.

Data Name	Value	Required/Optional
demote-incomplete-geometry	This boolean parameter determines if incomplete geometry should be demoted instead of producing an error. If set to true, whenever the geometry builder receives no simple solids to create a composite solid from, the composite solid will be downgraded to a null geometry. Range: true false Default: false	Optional
allow-empty-composite	This boolean parameter determines if incomplete geometry should be left incomplete instead of producing an error. If set to true, whenever the geometry builder receives no simple solids to create a composite solid from, the composite solid will be left empty and returned. Range: true false Default: false	Optional
coordinate-system	Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set. Range: String Default: "."	Optional
name	Specifies the geometry's name. Range: String	Optional

Example: Building a composite solid

compositesolid.xml

```

<?xml version="1.0"?>
<compositesolid>
  <simplesolid>
    <compositesurface>
      <simplesurface>
        <area>0.0,0.0,0.0 0.0,1.0,0.0 1.0,0.0,0.0 0.0,0.0,0.0</area>
      </simplesurface>
      <simplesurface>
        <area>0.0,0.0,0.0 1.0,0.0,0.0 1.0,1.0,1.0 0.0,0.0,0.0</area>
      </simplesurface>
      <simplesurface>
        <area>0.0,0.0,0.0 1.0,1.0,1.0 0.0,1.0,0.0 0.0,0.0,0.0</area>
      </simplesurface>
      <simplesurface>
        <area>1.0,0.0,0.0 0.0,1.0,0.0 1.0,1.0,1.0 1.0,0.0,0.0</area>
      </simplesurface>
    </compositesurface>
  </simplesolid>
  <simplesolid>
    <compositesurface>
      <simplesurface>
        <area>0.0,0.0,0.0 0.0,1.0,0.0 1.0,0.0,0.0 0.0,0.0,0.0</area>
      </simplesurface>
      <simplesurface>
        <area>0.0,0.0,0.0 1.0,0.0,0.0 1.0,1.0,-1.0 0.0,0.0,0.0</area>
      </simplesurface>
      <simplesurface>
        <area>0.0,0.0,0.0 1.0,1.0,-1.0 0.0,1.0,0.0 0.0,0.0,0.0</area>
      </simplesurface>
      <simplesurface>
        <area>1.0,0.0,0.0 0.0,1.0,0.0 1.0,1.0,-1.0 1.0,0.0,0.0</area>
      </simplesurface>
    </compositesurface>
  </simplesolid>
</compositesolid>

```

compositesolid.xmp

```

<?xml version="1.0"?>
<xfMap>
  <feature-map>
    <mapping match="compositesolid">
      <feature-type> <literal expr="compositesolid" /> </feature-type>
      <geometry activate="xml-composite-solid" />
    </mapping>
  </feature-map>
  <feature-content-map>
    <mapping match="simplesolid">
      <geometry activate="xml-enclosed-surface" />
    </mapping>
    <mapping match="compositesurface">
      <geometry activate="xml-composite-surface" />
    </mapping>
    <mapping match="simplesurface">
      <geometry activate="xml-face" />
    </mapping>
    <mapping match="area">
      <geometry activate="xml-area">
        <data name="data-string"> <extract expr="." /> </data>
      </geometry>
    </mapping>
  </feature-content-map>
</xfMap>

```

xml-multi-point

The xml-multi-point geometry builder constructs an aggregate geometry made up solely of points. This builder does not require any data parameters, it can only build a multi-point geometry from existing points.

builder type: composite - accept point geometries.

geometry constructed: multi-point geometry.

data parameters:

Data Name	Value	Required/Optional
keep-as-multi	Setting this optional parameter to "false" directs the builder to return a point rather than a multi-point geometry when only one point geometry is received. Range: false true Default: true	Optional
coordinate-system	Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set. Range: String Default: "."	Optional
name	Specifies the geometry's name. Range: String	Optional

xml-multi-curve

The xml-multi-curve geometry builder constructs a curve aggregate geometry. This builder does not require any data parameters, it can only build a multi-curve geometry from existing curves.

builder type: composite - accept curve, e.g., lines and arcs, geometries.

geometry constructed: multi-curve geometry.

data parameters:

Data Name	Value	Required/Optional
keep-as-multi	Setting this optional parameter to "false" directs the builder to return a single curve rather than a multi-curve geometry when only one curve geometry is received. Range: false true Default: true	Optional
coordinate-system	Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set.	Optional

Data Name	Value	Required/Optional
	Range: String Default: "."	
name	Specifies the geometry's name. Range: String	Optional

xml-multi-area

The xml-multi-area geometry builder constructs an aggregate area geometry. This builder does not require any data parameters, and so it can only build a multi-area geometry from existing areas.

builder type: composite - accept area geometries.

geometry constructed: multi-area geometry.

data parameters:

Data Name	Value	Required/Optional
keep-as-multi	Setting this optional parameter to "false" directs the builder to return a single area rather than a multi-area geometry when only one area geometry is received. Range: false true Default: true	Optional
coordinate-system	Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set. Range: String Default: "."	Optional
name	Specifies the geometry's name. Range: String	Optional

xml-multi-text

The xml-multi-text geometry builder constructs an aggregate text geometry. This builder does not require any data parameters, it can only build a multi-text geometry from existing text geometries.

builder type: composite - accept text geometries.

geometry constructed: multi-text geometry.

data parameters:

Data Name	Value	Required/Optional
keep-as-multi	Setting this optional parameter to "false" directs the builder to return a single text rather than a multi-text geometry when only one text geometry is received. Range: false true	Optional

Data Name	Value	Required/Optional
	Default: true	
coordinate-system	Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set. Range: String Default: "."	Optional
name	Specifies the geometry's name. Range: String	Optional

xml-multi-surface

The xml-multi-surface geometry builder builds an aggregate geometry made up solely of surface geometries. This builder does not require any data parameters, and will construct the multi surface geometry from existing surface geometries. Thus, before the xml-multi-surface geometry builder is deactivated, other geometry builders should have been activated, passing their constructed surfaces to the xml-multi-surface geometry builder.

Each of the surfaces passed to the xml-multi-surface geometry builder should be either simple or composite surfaces. The accepted surfaces do not have a restraint on their spatial relationship - they may be disjoint, overlapping, touching, or completely disconnected.

builder type: composite - accepts surface geometries

geometry constructed: multi surface

data parameters:

The following demote-incomplete-geometry and allow-empty-composite data parameters change how the geometry is handled in error conditions. If both of the data parameters are set to false, whenever the geometry builder receives no surfaces to create a multi surface from, it will raise an error and terminate the translation.

Data Name	Value	Required/Optional
keep-as-multi	Setting this optional parameter to "false" directs the builder to return a single surface rather than a multi-surface geometry when only one surface geometry is received. Range: false true Default: true	Optional
demote-incomplete-geometry	This Boolean parameter determines if incomplete geometry should be demoted instead of producing an error. If set to true, whenever the geometry builder receives no surfaces to create a multi surface from, the multi surface will be downgraded to a null geometry. Range: true false Default: false	Optional

Data Name	Value	Required/Optional
allow-empty-composite	This Boolean parameter determines if incomplete geometry should be left incomplete instead of producing an error. If set to true, whenever the geometry builder receives no surfaces to create a multi surface from, the multi surface will be left empty and returned. Range: true false Default: false	Optional
coordinate-system	Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set. Range: String Default: "."	Optional
name	Specifies the geometry's name. Range: String	Optional

Example: Building a multi surface

multisurface.xml

```
<?xml version="1.0"?>
<multisurface>
  <simplesurface>
    <area>
      0.0,0.0,0.0 5.0,0.0,5.0 5.0,5.0,10.0 0.0,5.0,5.0 0.0,0.0,0.0
    </area>
    <area>
      2.0,2.0,4.0 3.0,2.0,5.0 3.0,3.0,6.0 2.0,3.0,5.0 2.0,2.0,4.0
    </area>
  </simplesurface>
  <simplesurface>
    <area>
      0.0,0.0,0.0 5.0,0.0,0.0 5.0,5.0,0.0 0.0,5.0,0.0 0.0,0.0,0.0
    </area>
  </simplesurface>
</multisurface>
```

multisurface.xmp

```
<?xml version="1.0"?>
<xfMap>
  <feature-map>
    <mapping match="multisurface">
      <feature-type> <literal expr="multisurface" /> </feature-type>
      <geometry activate="xml-multi-surface" />
    </mapping>
  </feature-map>
  <feature-content-map>
    <mapping match="simplesurface">
```

```

        <geometry activate="xml-face" />
    </mapping>
    <mapping match="area">
        <geometry activate="xml-area">
            <data name="data-string"> <extract expr="." /> </data>
        </geometry>
    </mapping>
</feature-content-map>
</xfMap>

```

xml-multi-solid

The xml-multi-solid geometry builder builds an aggregate geometry made up solely of solid geometries. This builder does not require any data parameters, and will construct the multi solid geometry from existing solid geometries. Thus, before the xml-multi-solid geometry builder is deactivated, other geometry builders should have been activated, passing their constructed solids to the xml-multi-solid geometry builder.

Each of the solids passed to the xml-multi-solid geometry builder should be either simple or composite solids. The accepted solids do not have a restraint on their spatial relationship - they may be disjoint, overlapping, touching, or completely disconnected.

builder type: composite - accepts solid geometries

geometry constructed: multi solids

data parameters:

The following demote-incomplete-geometry and allow-empty-composite data parameters change how the geometry is handled in error conditions. If both of the data parameters are set to false, whenever the geometry builder receives no solids to create a multi solid from, it will raise an error and terminate the translation.

Data Name	Value	Required/Optional
keep-as-multi	Setting this optional parameter to "false" directs the builder to return a single solid rather than a multi-solid geometry when only one solid geometry is received. Range: false true Default: true	Optional
demote-incomplete-geometry	This boolean parameter determines if incomplete geometry should be demoted instead of producing an error. If set to true, whenever the geometry builder receives no solids to create a multi solid from, the multi solid will be downgraded to a null geometry. Range: true false Default: false	Optional
allow-empty-composite	This boolean parameter determines if incomplete geometry should be left incomplete instead of producing an error. If set to true, whenever the geometry builder receives no solids to create a multi solid from, the multi solid will be left empty and returned. Range: true false Default: false	Optional

Data Name	Value	Required/Optional
coordinate-system	Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set. Range: String Default: "."	Optional
name	Specifies the geometry's name. Range: String	Optional

Example: Building a multi solid

multisolid.xml

```

<?xml version="1.0"?>
<multisolid>
  <simplesolid>
    <compositesurface>
      <simplesurface>
        <area>
          0.0,0.0,0.0 0.0,1.0,0.0 1.0,0.0,0.0 0.0,0.0,0.0
        </area>
      </simplesurface>
      <simplesurface>
        <area>
          0.0,0.0,0.0 1.0,0.0,0.0 1.0,1.0,1.0 0.0,0.0,0.0
        </area>
      </simplesurface>
      <simplesurface>
        <area>
          0.0,0.0,0.0 1.0,1.0,1.0 0.0,1.0,0.0 0.0,0.0,0.0
        </area>
      </simplesurface>
      <simplesurface>
        <area>
          1.0,0.0,0.0 0.0,1.0,0.0 1.0,1.0,1.0 1.0,0.0,0.0
        </area>
      </simplesurface>
    </compositesurface>
  </simplesolid>
  <simplesolid>
    <compositesurface>
      <simplesurface>
        <area>
          0.0,0.0,0.0 0.0,-1.0,0.0 -1.0,0.0,0.0 0.0,0.0,0.0
        </area>
      </simplesurface>
      <simplesurface>
        <area>
          0.0,0.0,0.0 -1.0,0.0,0.0 -1.0,-1.0,-1.0 0.0,0.0,0.0
        </area>
      </simplesurface>
      <simplesurface>
        <area>
          0.0,0.0,0.0 -1.0,-1.0,-1.0 0.0,-1.0,0.0 0.0,0.0,0.0
        </area>
      </simplesurface>
    </compositesurface>
  </simplesolid>
</multisolid>

```

```

        <area>
            1.0,0.0,0.0 0.0,-1.0,0.0 -1.0,-1.0,-1.0 -1.0,0.0,0.0
        </area>
    </simplesurface>
</compositesurface>
</simplesolid>
</multisolid>

```

multisolid.xmp

```

<?xml version="1.0"?>
<xfMap>
    <feature-map>
        <mapping match="multisolid">
            <feature-type> <literal expr="multisolid" /> </feature-type>
            <geometry activate="xml-multi-solid" />
        </mapping>
    </feature-map>
    <feature-content-map>
        <mapping match="simplesolid">
            <geometry activate="xml-enclosed-surface" />
        </mapping>
        <mapping match="simplesurface">
            <geometry activate="xml-face" />
        </mapping>
        <mapping match="area">
            <geometry activate="xml-area">
                <data name="data-string"> <extract expr="." /> </data>
            </geometry>
        </mapping>
    </feature-content-map>
</xfMap>

```

xml-null

The xml-null geometry builder is a non-composite geometry builder, it constructs a geometry object that represents the null geometry.

builder type: non-composite.

geometry constructed: the null geometry.

data parameters:

Data Name	Value	Required/Optional
name	Specifies the geometry's name Range: String	Optional

xml-reverse-geometry

The xml-reverse-geometry geometry builder is a composite geometry builder. The builder accepts a single geometry and reverses the order of its coordinates.

builder type: composite.

geometry constructed: the geometry received with the order of its coordinates reversed

data parameters:

Data Name	Value	Required/Optional
reverse-geometry	Reverses the geometry if it is set to true. Range: true/false Default: true	Optional

fme-geometry

This is a non-composite builder that deserializes different geometry types from different geometry serializations.

builder type: non-composite.

geometry constructed: All FME supported geometries.

data parameters:

Data Name	Value	Required/Optional
data-string	The string containing the serialized geometry. Range: String	Required
encoding	Specifies the encoding of the serialized geometry. The encoding maybe the FME XML geometry representation, the FME hex encoded binary geometry serialization, the OGC hex encoded WKB serialization or the OGC WKT serialization. Range: fme-xml fme-binary-hex ogc-wkb-hex ogc-wkt. Default: fme-xml.	Optional
coordinate-system	Specifies the coordinate system name. If the expression sequence for the coordinate-system evaluates to the empty string, then the coordinate system for the feature being built will not be set. Range: String Default: "."	Optional
name	Specifies the geometry's name. Range: String	Optional

The following example illustrates the usage of the fme-geometry builder.

fmegeometry.xml

```
<?xml version="1.0"?>
<feature>
  <geom>POINT (-0.915929 0.482301)</geom>
</feature>
```

fmegeometry.xmp

```
<?xml version="1.0"?>
<xfMap>
  <feature-map>
    <mapping match="feature">
      <feature-type> <literal expr="feature" /> </feature-type>
      <geometry activate="fme-geometry">
        <data name="encoding"><literal expr="ogc-wkt"/></data>
        <data name="data-string">
          <extract expr="./geom"/>
        </data>
      </geometry>
    </mapping>
  </feature-map>
</xfMap>
```

```

        </geometry>
      </mapping>
    </feature-map>
  </xfMap>

```

Mapping Segmented Geometric Information

Sometimes it may take more than one mapping rule to successfully extract the geometric data from the elements in the input XML document stream. Consider the following polyline element:

```

<polyline>
  <coord>0.0,0.0</coord>
  <coord>1.1,2.2</coord>
  <coord>5.3,-1.9</coord>
  <coord>7.9,3.5</coord>
</polyline>

```

Refer to FME Feature Construction (defining mapping rules under the <feature-content-map> element). It explains when mapping rules should be define under the <feature-content-map> element to handle cases like this one.

What we need here is to activate just one geometry builder *B* – one that constructs a line geometry. If we just define one feature mapping rule matching the <polyline> element, then we will not be able to give *B* the contents of all the <coord> elements because *B* gets suspended when the <polyline> element ceases to be the context element.

Recall that a suspended geometry builder may be set to the state of execution if the currently executing mapping rule contains a <geometry> element having no activate attribute.

We need to re-execute *B* and pass it the contents of the <coord> element when this element is matched. To achieve, this we define an additional *feature mapping rule*, that matches a <coord> element and contains a <geometry> element with no activate attribute.

The following xfMap document fragment does exactly this. The built-in geometry builder xml-line constructs polyline features.

```

<feature-map>
  <mapping match="polyline">
    ...
    <geometry activate="xml-line">
  </geometry>
  </mapping>
</feature-map>

<feature-content-map>
  <mapping match="coord">
    ...
    <geometry>
      <!-- activate attribute missing, execute
           the suspended mapping rule passing it
           the parameters that are defined in the
           geometry.

           NOTE: It is an error to define a mapping rule
           of this form if no geometry builder was
           activated. -->

      <data name="data-string">
        <extract expr="."/>
      </data>
    </geometry>
  </mapping>
</feature-content-map>

```

Geometry Traits (trait element)

A <geometry> element may have any number of optional <trait> elements. These elements specify any number of geometry traits for the geometry that is under construction. Geometry traits passed to the geometry builder in this manner will appear on the geometry currently being constructed, but will not affect any component geometries in the case that the traits are passed to a composite builder.

The geometry traits supplied are represented as name and value pairs, both of which have expression sequences as their values. They are represented in the xfMap by the <trait>, <name>, and <value> elements:

```
<trait required="true|false">
  <name> <!-- an expression sequence --> </name>
  <value> <!-- an expression sequence --> </value>
</trait>
...
<trait>
...
</trait>
```

The optional *required* attribute on the trait element defaults to *true*, if set to *false* then the trait is added to the geometry only if the trait value is not the empty string.

Once again, if the <geometry> element does not have an activate attribute, then the traits will pass to the currently suspended geometry builder, reactivating it to receive traits.

The following xfMap document, traits.xmp contains a mapping rule which matches a <point> element from the points1.xml document. The mapping rule contains a geometry element with a couple of simple traits.

points1.xml

```
<?xml version="1.0"?>
<points>
  <point name="myPoint" num="0">
    <color>
      <red>0.324</red>
      <green>0.233</green>
      <blue>0.596</blue>
    </color>
    <location x="10.0" y="0.0" />
  </point>
  <point name="myPoint" num="1">
    <color>
      <red>0.874</red>
      <green>0.948</green>
      <blue>0.554</blue>
    </color>
    <location x="5.0" y="5.0" />
  </point>
</points>
```

traits.xmp

```
<?xml version="1.0"?>
<xfMap>
  <feature-map>
    <mapping match="point">
      <feature-type>
        <extract expr="@name" />
        <literal expr="_" />
        <extract expr="@num" />
      </feature-type>
      <geometry activate="xml-point">
        <data name="data-string">
          <extract expr="./location[@x]" />
          <literal expr="," />
        </data>
      </geometry>
    </mapping>
  </feature-map>
</xfMap>
```

```

        <extract expr=". /location[@y]" />
    </data>
    <trait>
        <name>
            <literal expr="my_geometry_trait" />
        </name>
        <value>
            <!-- We set the "my_geometry_trait" trait to be
                "sample_value" -->
            <literal expr="sample_value" />
        </value>
    </trait>
    <trait>
        <name>
            <literal expr="point_num" />
        </name>
        <value>
            <!-- The expression sequence below constructs the
                value of "point_num" to be the value of the
                attribute in points1.xml -->
            <extract expr="@num" />
        </value>
    </trait>
</geometry>
</mapping>
</feature-map>
</xfMap>

```

When the above points1.xml and traits.xmp documents are fed into the XML Reader, the following FME features are constructed:

```

+++++
Feature Type: myPoint_0'
Attribute(string): fme_feature_type' has value myPoint_0'
Attribute(string): fme_geometry' has value fme_point'
Attribute(string): fme_type' has value fme_point'
Attribute(string): xml_type' has value xml_point'
Coordinate System: '
Geometry Type: IFMEPoint
Number of Geometry Traits: 2
GeometryTrait(string): my_geometry_trait' has value sample_value'
GeometryTrait(string): point_num' has value 0'
Coordinate Dimension: 2
(10,0)
=====

+++++
Feature Type: myPoint_1'
Attribute(string): fme_feature_type' has value myPoint_1'
Attribute(string): fme_geometry' has value fme_point'
Attribute(string): fme_type' has value fme_point'
Attribute(string): xml_type' has value xml_point'
Coordinate System: '
Geometry Type: IFMEPoint
Number of Geometry Traits: 2
GeometryTrait(string): my_geometry_trait' has value sample_value'
GeometryTrait(string): point_num' has value 1'
Coordinate Dimension: 2
(5,5)
=====

```

FME Feature Construction (constructing multiple features at a time)

XML data is most often hierarchal than "flat", and so it is common to encounter elements embedded within elements for which we may want to map both child and parent as features. For example, consider a <building> with <name>, <location> and several <wall> child elements:

```

<building>
  <name>C10</name>

```

```

    <location>east side</location>
    <wall> ... </wall>
    <wall> ... </wall>
</building>

```

If we want to map both <building> and <wall> elements as FME features then we need one or more mapping rules matching the <building> and <wall> elements in the xFMap's <feature-map> element. But by default there's only one active feature-search-set. In the example above, the <building> element will trigger the construction of a new FME feature, then immediately, the one and only feature-search-set will be set to contain mapping rules from the <feature-content-map> and it will not be set back to the <feature-map> until the </building> end element tag is read, therefore the "wall" mapping rule in the <feature-map> will never be considered for activation.

The xFMap <feature-map> element may have an optional multi-feature-construction attribute whose default value is false but may be settable to true. Setting the multi-feature-construction to true allows the XML reader to construct multiple features a time by instructing it to always create an additional feature-search-set whose content is the <feature-map>, recall that only mapping rules in the <feature-map> trigger the construction of a new feature when activated.

For example:

```

<feature-map multi-feature-construction="true">
  <mapping match="building">
    ...
  </mapping>
  <mapping match="wall">
    ...
  </mapping>
</feature-map>

```

Structure Element

Every feature mapping rule may contain an optional <structure> element that allows an XML subtree, that is rooted in a mapping rule's matched element, be added as attribute lists to the FME feature under construction. FME attribute lists behave just as primitive attributes, except that they may contain an index enclosed in braces to identify an element of the list. Attribute list elements may contain primitives or other attribute lists.

The attribute lists indices will likely not correspond to the XML subtree element order. Consider the following XML subtree rooted at <a>:

```

<a>
  <b/><c/><b/><b/>
</a>

```

Element is a repeating child of <a>, but the indices for the list attributes are such that they must increase without gaps and as a consequence the ordering for the children of <a> is lost when we have interweaving repeating child elements:

```

a{0}.b{0}
a{0}.b{1}
a{0}.b{2}
a{0}.c{0}

```

The structure instruction in a feature mapping rule may be specified by a single empty <structure/> element. This directs the XML reader to start constructing FME attribute lists from the subtree rooted at the matched element:

```

<mapping match="...">
  <feature-type> ... </feature-type>
  <attributes> ... </attributes>
  <geometry> ... </geometry>
  <structure/>
</mapping>

```

Consider the following XML document, a_items.xml:

a_items.xml

```

<?xml version="1.0"?>
<a-items>
  <a>
    <b>a0b0</b>
    <c x="first x-val" y="first y-val">a0c0</c>
    <b>a0b1</b>
    <d><e>a0e</e></d>
    <b>a0b2</b >
      <f></f>
      <g>
    </a>
  </a-items>

```

The following xfMap document, a.xmp, maps each <a> element into an FME feature while turning the subtree that is rooted at <a> into attribute lists:

a.xmp:

```

<?xml version="1.0"?>
<xfMap>
  <feature-map>
    <mapping match="a">
      <feature-type><literal expr="a"/></feature-type>>
      <structure/>
    </mapping>
  </feature-map>
</xfMap>

```

The a.xmp constructs the following feature from a_items.xml:

```

+++++
Feature Type: a'
Attribute(string): a{0}.b{0}' has value a0b0'
Attribute(string): a{0}.b{1}' has value a0b1'
Attribute(string): a{0}.b{2}' has value a0b2'
Attribute(string): a{0}.c{0}' has value a0c0'
Attribute(string): a{0}.c{0}.x' has value first x-val'
Attribute(string): a{0}.c{0}.y' has value first y-val'
Attribute(string): a{0}.d{0}.e{0}' has value a0e'
Attribute(string): xml_type' has value xml_no_geom'
Geometry Type: Unknown (0)
=====

```

XML attributes in the FME attribute lists are represented without an index. Notice that the x and y attributes for the <c> element in the a_items.xml document do not have a list index in the FME feature.

It is important to notice that the <f> and <g> elements in the above example did not map over to the FME feature as attributes, this is because <f> and <g> do not have character content. To make the XML reader create the corresponding FME feature list attributes a{0}.f{0} and a{0}.g{0} for the empty <f> and <g> elements, respectively, the optional **map-empty-elements** xml attribute for the structure element should be set to yes. For example:

```

<?xml version="1.0"?>
<xfMap>
  <feature-map>
    <mapping match="a">
      <feature-type><literal expr="a"/></feature-type>>
      <structure map-empty-elements="yes"/>
    </mapping>
  </feature-map>
</xfMap>

```

In addition, XML attributes can also be differentiated from leaf elements, by letting the XML reader append a prefix to their name. The <structure> element may have an optional **attribute-identifier** xml attribute whose value becomes the prefix for the name in the FME attribute list.

As noted earlier, the list indices in the generated attribute names only preserve the ordering for elements with the same name. The **child-position-attribute** attribute can be used on the structure element to preserve the ordering

of all child elements, regardless of name. When this attribute is specified, each child element will generate an additional feature attribute whose value will be the position of the child element within its parent. The name of the attribute will be the list prefix representing the path to the element, followed by the value of the **child-position-attribute** attribute. If the **attribute-identifier** attribute is also specified, it will be used in the feature's position attribute.

Consider applying the following xfMap, a1.xmp, to the structures_items.xml document. The xfMap appends '@' to every list component whose name originated from an xml attribute for elements in the subtree rooted at <a>. Also, each child element has a 'pos' attribute containing its position within its parent.

a1.xmp:

```
<?xml version="1.0"?>
<xfMap>
  <feature-map>
    <mapping match="a">
      <feature-type><literal expr="a"/></feature-type>
      <structure attribute-identifier="@"
                child-position-attribute="pos" />
    </mapping>
  </feature-map>
</xfMap>
```

FME feature constructed:

```
+++++
Feature Type: a'
Attribute(string): a{0}.b{0}' has value a0b0'
Attribute(string): a{0}.b{1}' has value a0b1'
Attribute(string): a{0}.b{2}' has value a0b2'
Attribute(string): a{0}.c{0}' has value a0c0'
Attribute(string): a{0}.c{0}.@x' has value first x-val'
Attribute(string): a{0}.c{0}.@y' has value first y-val'
Attribute(string): a{0}.d{0}.e{0}' has value a0e'
Attribute(string): xml_type' has value xml_no_geom'
Geometry Type: Unknown (0)
=====
```

A prefix may also be attached to every FME attribute list that is generated through a structure for a matched element. The xfMap <structure> element may have an optional **structure-prefix** attribute whose value becomes the attribute lists prefix. The following a2.xmp xfMap document extends a1.xmp by adding the "myStructurePrefix-" prefix onto the attribute lists for the constructed feature.

a2.xmp:

```
<?xml version="1.0"?>
<xfMap>
  <feature-map>
    <mapping match="a">
      <feature-type><literal expr="a"/></feature-type>
      <structure structure-prefix="myStructurePrefix-"
                attribute-identifier="@"/>
    </mapping>
  </feature-map>
</xfMap>
```

Applying the a2.xmp xfMap to the structures_items.xml document makes the XML reader construct the following feature:

```
+++++
Feature Type: a'
Attribute(string): myStructurePrefix-a{0}.b{0}' has value a0b0'
Attribute(string): myStructurePrefix-a{0}.b{1}' has value a0b1'
Attribute(string): myStructurePrefix-a{0}.b{2}' has value a0b2'
Attribute(string): myStructurePrefix-a{0}.c{0}' has value a0c0'
Attribute(string): myStructurePrefix-a{0}.c{0}.@x' has value first x-val'
Attribute(string): myStructurePrefix-a{0}.c{0}.@y' has value first y-val'
Attribute(string): myStructurePrefix-a{0}.d{0}.e{0}' has value a0e'
Attribute(string): xml_type' has value xml_no_geom'
```

Geometry Type: Unknown (0)

In the examples above, the separator used for attributes on the feature is the period ('.'). Thus, the element 'e', child of element 'd', child of element 'a' is represented as a{0}.d{0}.e{0}. Each <structure> element can have an optional **separator** attribute (the default is the period character). If you need to change the separator, this is possible simply by specifying the separator as shown in the following example.

A3.xmp:

```
<?xml version="1.0"?>
<xfMap>
  <feature-map>
    <mapping match="a">
      <feature-type><literal expr="a"/></feature-type>
      <structure attribute-identifier="@ separator="-"/>
    </mapping>
  </feature-map>
</xfMap>
```

FME feature constructed:

```
+++++++
Feature Type: a'
Attribute(string): a{0}--b{0}' has value a0b0'
Attribute(string): a{0}--b{1}' has value a0b1'
Attribute(string): a{0}--b{2}' has value a0b2'
Attribute(string): a{0}--c{0}' has value a0c0'
Attribute(string): a{0}--c{0}--x' has value first x-val'
Attribute(string): a{0}--c{0}--y' has value first y-val'
Attribute(string): a{0}--d{0}--e{0}' has value a0e'
Attribute(string): xml_type' has value xml_no_geom'
Geometry Type: Unknown (0)
=====
```

It is also possible to tell the XML Reader to ignore the matched element when it is constructing an attribute list. This is done by setting the optional **skip-matched** attribute on the xfMap <structure> element to "yes". The valid values for this attribute are "yes" and "no", and its default value is "no". For example, applying the following xfMap a4.xmp to the a_items.xml document will remove the a{0} component from the attribute lists for the constructed FME feature :

a4.xmp:

```
<?xml version="1.0"?>
<xfMap>
  <feature-map>
    <mapping match="a">
      <feature-type><literal expr="a"/></feature-type>
      <structure skip-matched="yes"
        structure-prefix="myStructurePrefix-"
        attribute-identifier="@"/>
    </mapping>
  </feature-map>
</xfMap>
```

Notice that the a{0} component does not appear in the attribute lists:

```
+++++++
Feature Type: a'
Attribute(encoded: utf-16): `a{0}.b{0}' has value `a0b0'
Attribute(encoded: utf-16): `a{0}.b{0}.@pos' has value `0'
Attribute(encoded: utf-16): `a{0}.b{1}' has value `a0b1'
Attribute(encoded: utf-16): `a{0}.b{1}.@pos' has value `2'
Attribute(encoded: utf-16): `a{0}.b{2}' has value `a0b2'
Attribute(encoded: utf-16): `a{0}.b{2}.@pos' has value `4'
Attribute(encoded: utf-16): `a{0}.c{0}' has value `a0c0'
Attribute(encoded: utf-16): `a{0}.c{0}.@pos' has value `1'
Attribute(encoded: utf-16): `a{0}.c{0}.@x' has value `first x-val'
Attribute(encoded: utf-16): `a{0}.c{0}.@y' has value `first y-val'
Attribute(encoded: utf-16): `a{0}.d{0}.@pos' has value `3'
Attribute(encoded: utf-16): `a{0}.d{0}.e{0}' has value `a0e'
Attribute(encoded: utf-16): `a{0}.d{0}.e{0}.@pos' has value `0'
```

```

Attribute(encoded: utf-16): `a{0}.f{0}.@pos' has value `5'
Attribute(encoded: utf-16): `a{0}.g{0}.@pos' has value `6'
Attribute(string): xml_type' has value xml_no_geom'
Geometry Type: Unknown (0)
=====

```

It is also possible to control the appearance of the attributes in those instances where the xml is known to only allow a single instance of an element. In the example above for instance, the <c> and <d> elements might be constrained to only occur once. In these cases, the list-suffix is cluttering up the attribute name. The structure element introduces a mini-language to define the cardinalities of the elements. Below an example is given, followed by more detailed discussion.

A5.xmp:

```

<?xml version="1.0"?>
<xfMap>
  <feature-map>
    <mapping match="a">
      <feature-type><literal expr="a"/></feature-type>
      <structure skip-matched="yes" attribute-identifier="@"
cardinality="*/c */d{}/+ */+"/>
    </mapping>
  </feature-map>
</xfMap>

```

FME feature constructed:

```

+++++
Feature Type: a'
Attribute(string): b' has value a0b0'
Attribute(string): b{1}' has value a0b1'
Attribute(string): b{2}' has value a0b2'
Attribute(string): c' has value a0c0'
Attribute(string): c.@x' has value first x-val'
Attribute(string): c.@y' has value first y-val'
Attribute(string): d{0}.e' has value a0e'
Attribute(string): xml_type' has value xml_no_geom'
Geometry Type: Unknown (0)
=====

```

The **cardinality** attribute is a space separated list of cardinality directives. In the previous example, the strings are:

1. */c
2. */d{}/+
3. */+

Each forward-slash separated element indicates an element in the xml-document. The asterisk acts as a wildcard, matching any element. A literal string matches the name of an element. The use of the braces ({}) indicates that element should be treated like a list, while no braces indicates that the element should be treated as singular if possible. Finally, the trailing '+' or '+{}' indicates that any further elements along this path through the xml-document should be treated as non-list (+) or list ({}) elements.

In the above example, cardinality (1) matches the <a> element (the root) and the <c> element. This indicates that both the root and the <c> element should be treated as singular. Attributes are always singular.

Cardinality (2) matches the the <a> element, followed by the <d> element which should be treated as a list, followed by any number of attributes, all of which should be treated as non-list.

Finally, cardinality (3) matches the root <a>, followed by any other xml elements, all of which should be treated as singular. In this circumstance "*/+" fixes the cardinality of exactly the same set of attributes as "+" would have.

These three rules are applied in order to determine matches

1. Literal matches are preferred to wildcard matches. E.g. a/+ is preferred to */+
2. Literal matches occurring early in a cardinality expression are preferred to literal matches occurring late. E.g. a/** is preferred to */b/*.
3. Non-list elements are preferred to list elements. So */foo/ is preferred to */foo{}/.

This doesn't provide a total ordering on the cardinality expressions, since e.g. a/b/c should sort exactly the same way as d/e/f, but since these will not match the same elements, the order doesn't matter. The basic elements of a cardinality expression are: literal matches, consisting of characters matching the name of an xml-element; wildcard matches: "*"{"}, "*" matching exactly one element, and treating it as a list, or non-list (respectively); and an optional suffix: +{"}, + to indicate that any further matches should be treated as non-list or list (respectively).

In case there are no matches found, the default behaviour is to assume that the cardinality is specified as "+{"}. In order to match, an attribute path (e.g. <a><c/> is matched by a.b.c) must be exactly as long as the cardinality expression. The only exception is that the suffixes "+" and "+{"} extend the cardinality expression as long as is necessary to match a string.

In addition to specifying element names, forward-slash separated elements can also include a namespace prefix and a colon. If a namespace prefix is specified, then the colon must also be specified. In all cases, an element name must also be specified (possibly as a wildcard). If no namespace prefix is given, the effect is the same as specifying a wildcard for the prefix. It is possible to specify a "blank" prefix, by having nothing before the colon. In this case, it will only match if the actual element's prefix is the empty string.

In other words:

1. a/b/*/+ is the same as *:a/*:b/*:c/+
2. a:b/c will match a element but not a <test:b> element (where <a> and <c> match)

The '+' and '+{"} suffixes do not currently take a namespace specifier.

If one wants to include the prefix in the name of the attribute (in order to treat elements from different namespaces as different attributes on their FME feature), one must set the attribute "use-namespace-prefix" on the structure element to "yes".

Note that there is no interaction between the skip-matched attribute and the cardinality attribute. This means that even if skip-matched="yes", an element in the cardinality expression must still match it. For example, if we had specified the structure element in a5.xmp to be

```
<structure skip-matched="yes" cardinality="c d/+{*} */+"/>
```

The result would have been that the only match would be */+, since none of the other cardinality expressions would match the <a> element. In normal use, this simply means that the first element in the cardinality expression should be either a wildcard or the name of the matching element. This can be useful if one wishes to match a number of different elements, some of which have different cardinality constraints. It allows quite succinct xfMaps to be written.

a6.xmp

```
<?xml version="1.0"?>
<xfMap>
  <feature-map>
    <mapping match="a-list/*">
      <feature-type><matched expr="local-name"/></feature-type>
      <structure cardinality="a/b{/+ a/b{/c/+{/ z/c{/d{/e"/>
    </mapping>
  </feature-map>
</xfMap>
```

The above example will match any element that is a child of the element "a-list", name the feature according to the element matched, and then use the cardinalities given to determine how to write the attributes out.

Finally, we would often like to exclude some elements of an xml tree from conversion into FME attributes. An obvious case is one such as the following, were we want to map all the xml leaf elements to FME attributes, except those which are used to construct the geometry of the feature.

A7.xml:

```
<?xml version="1.0"?>
<features>
<feature>
  <name>Downtown Harbour</name>
  <age>132 years </age>
```

```

    <lat>100</lat>
    <lon>54.2</lon>
  </feature>
</feature>
  <name>EastSide Harbour</name>
  <age>38 years </age>
  <lat>101.2</lat>
  <lon>54.8</lon>
</feature>
</features>

```

a7.xmp:

```

<?xml version="1.0"?>
<xfMap>
  <feature-map>
    <mapping match="feature">
      <feature-type><literal expr="Harbour"/></feature-type>
      <geometry activate="xml-point">
        <data name="data-string">
          <extract expr="./lat"/>
          <literal expr=","/>
          <extract expr="./lon"/>
        </data>
      </geometry>
      <structure skip-matched="yes"
        cardinality="+"
        except="lat lon"/>
    </mapping>
  </feature-map>
</xfMap>

```

FME feature constructed:

```

+++++
Feature Type: Harbour'
Attribute(string): name' has value Downtown Harbour'
Attribute(string): age' has value 132'
Attribute(string): xml_type' has value xml_point'
Geometry Type: IFMEPoint
(100,0, 58.2)
Feature Type: Harbour'
Attribute(string): name' has value Eastside Harbour'
Attribute(string): age' has value 37'
Attribute(string): xml_type' has value xml_point'
Geometry Type: IFMEPoint
(101,2, 54.8)
=====

```

Here we explicitly exclude the xml elements <lat> and <lon> in order to extract them using the geometry tag (discussed elsewhere in the xfMap documentation). This avoids having attributes which mirror the geometry.

The **except** attribute accepts the same types of expressions as the match or except attribute of a mapping rule. For example, the expression `except="parent/child{2}"` could be used to exclude the second <child> element contained in a <parent> element from the output of the structure subrule.

Note that, currently, <structure> elements cannot be constructed in parallel on a feature – only one can be constructed at a time.

References Element

A *feature mapping rule* may also contain an optional <references> element. This element specifies reference-sets that hold name/value pairs that may be accessed via <refexpr> expression elements. A detailed description for the <references> element and its content may be found in the Reference Mapping Rules section.

In a *feature mapping rule* the <references> element should appear before the <feature-type> element, for example:

```

<feature-map>
  <mapping match="...">
    ...
    <references>...</references>
    <feature-type>...</feature-type>
    ...
  </mapping>
</feature-map>

```

Group Mapping Rules

Group mapping rules specify the construction of *xfMap groups*, and they must be defined inside the `<group-map>` or `<group-content-map>` elements.

FME features that are created through the *feature mapping rules* may be further processed by the *xfMap groups*, these are processing entities that reside inside the XML Reader in which features may enter and leave, but while inside a group, the features may be further modified through:

- a. **the attachment of group specific attributes**, and/or
- b. **the processing of an FME factory pipeline**.

All *group mapping rules* whether they're defined under the `<group-map>` or the `<group-content-map>` elements have the same structure.

Unlike the *feature mapping rules* every *group mapping rule* when activated, regardless of where it is defined, triggers the XML Reader to construct a new *group*.

Group Construction and Destruction

We now describe when an *xfMap group* is created and destroyed. If we let *R* be a *group mapping rule* defining the *group G*, then:

- a. *G* is **constructed** when *R* is activated, and
- b. *G* is **destroyed** when *R* is de-activated, but only if:
 1. *R* was defined in the `<group-map>` element, or
 2. *G* is not a *persistent group*.

The XML Reader keeps a stack of constructed *groups*; for expository convenience we'll name this stack the **g-stack**. When *R* activates, *G* is pushed into the *g-stack*, *G* is popped from the *g-stack* when *R* de-activates.

Note: If for the moment we ignore the existence of persistent groups, then what b) says is that a group *G* is destroyed whenever its corresponding mapping rule *R* is de-activated. Furthermore, if *R* was defined in the `<group-map>` element, then it does not matter if *G* is a persistent group or not, because it will always be destroyed when *R* de-activates. That is, there is no reason to define a group in the `<group-map>` element to be a persistent group: it will never persist because the condition in b) 1) does not allow it.

FME features that are constructed by *feature mapping rules* will enter all *groups* that are in the *g-stack* from the top until the bottom *group* of the stack. The FME features that leave the bottom *group* of the *g-stack* are output by the XML Reader.

Group Attribute-sets

A *group* can attach a set of attributes to the FME features that enter it. These attribute sets are defined in the *group mapping rule* by the optional `<apply-attribute-sets>` element.

The `<apply-attribute-sets>` element can have one or more `<attribute-set>` elements that describes an attribute collection through its `<attributes>` element (see the section "*feature mapping rules - attributes element*").

Each `<attribute-set>` element may also contain an optional `<condition>` element (see section "*group mapping rule - condition element*") that allows or prevents a feature from entering into the attribute set.

The general form of the `<apply-attribute-sets>` element is:

```

<apply-attribute-sets>
  <attribute-set>
    <!-- optional condition -->
    <condition .../>
    <attributes>...</attributes>
  </attribute-set>
  ...
  <attribute-set>...</attribute-set>
</apply-attribute-sets>

```

The following example illustrates the usage of the *group mapping rules's attribute sets*. Consider the *cleaning.xml* input XML document:

cleaning.xml

```

<?xml version="1.0"?>
<cleaning-schedule date="09 Mar 2001">
  <staff first-name="John" last-name="Norton" id="00098">
    <room>302</room>
    <room>210</room>
    <room>450</room>
  </staff>
  <staff first-name="Laura" last-name="Lee" id="00029">
    <room>192</room>
    <room>597</room>
  </staff>
</cleaning-schedule>

```

We map each of the above <room> elements into an FME feature. We use *group mapping rules* to add the information from the <cleaning-schedule> and <staff> elements (refer to the comments in the xfMap document below for further detail):

cleaning.xmp

```

<?xml version="1.0"?>
<!DOCTYPE xfMap SYSTEM "xfMap.dtd">

<xfMap>
  <group-map>
    <mapping match="cleaning-schedule">
      <!-- This group mapping rule activates when the cleaning-schedule
      element start-tag is read. The group constructed, called
      it G0, has one attribute set which attaches to the
      features that enters it. G0 is pushed into the XML Reader's
      g-stack, and it will be the first group in the stack. -->
      <apply-attribute-sets>
        <attribute-set>
          <attributes>
            <attribute>
              <name> <literal expr="cleaning date"/> </name>
              <value> <extract expr="@date"/> </value>
            </attribute>
          </attributes>
        </attribute-set>
      </apply-attribute-sets>
    </mapping>
  </group-map>

  <group-content-map>
    <mapping match="staff">
      <!-- This mapping rule is activated when the staff element
      start-tag is read. The group constructed, called it G1,
      is pushed into the g-stack, it is popped after the
      staff element's end-tag is read. G1 contains an attribute set
      with two attributes which get attached to features that enter

```

```

        the group. -->
    <!-- The 'room' features that are constructed by the
        feature mapping rule below will pass through G1 and
        then through G0 before being output. Notice the attribute
        set contents of G1 changes (there will be two G1s created
        since there are two staff elements in the input dataset. -->
    <apply-attribute-sets>
        <attribute-set>
            <attributes>
                <attribute>
                    <name> <literal expr="staff name"/> </name>
                    <value>
                        <extract expr="@first-name"/>
                        <literal expr=" "/>
                        <extract expr="@last-name"/>
                    </value>
                </attribute>
                <attribute>
                    <name> <literal expr="staff id"/> </name>
                    <value> <extract expr="@id"/> </value>
                </attribute>
            </attributes>
        </attribute-set>
    </apply-attribute-sets>
</mapping>
</group-content-map>

<feature-map>
    <mapping match="room">
        <feature-type> <literal expr="room"/> </feature-type>
        <attributes>
            <attribute>
                <name> <literal expr="room number"/> </name>
                <value> <extract expr="."/> </value>
            </attribute>
        </attributes>
    </mapping>
</feature-map>
</xfMap>

```

The two documents above make the XML Reader output the following 5 FME room features:

```

+++++
Feature Type: `room'
Attribute: `cleaning date' has value `09 Mar 2001'
Attribute: `room number' has value `302'
Attribute: `staff id' has value `00098'
Attribute: `staff name' has value `John Norton'
Attribute: `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `room'
Attribute: `cleaning date' has value `09 Mar 2001'
Attribute: `room number' has value `210'
Attribute: `staff id' has value `00098'
Attribute: `staff name' has value `John Norton'
Attribute: `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `room'
Attribute: `cleaning date' has value `09 Mar 2001'
Attribute: `room number' has value `450'
Attribute: `staff id' has value `00098'

```

```

Attribute: `staff name' has value `John Norton'
Attribute: `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `room'
Attribute: `cleaning date' has value `09 Mar 2001'
Attribute: `room number' has value `192'
Attribute: `staff id' has value `00029'
Attribute: `staff name' has value `Laura Lee'
Attribute: `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `room'
Attribute: `cleaning date' has value `09 Mar 2001'
Attribute: `room number' has value `597'
Attribute: `staff id' has value `00029'
Attribute: `staff name' has value `Laura Lee'
Attribute: `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====

```

Group Pipelines

A *group* may contain a sequence of FME factory pipelines that processes features entering it. A *group mapping rule*, that defines a *group* with pipelines, has an `<apply-pipelines>` element which contains a sequence of one or more `<pipeline>` elements.

Each `<pipeline>` element has a `<file>` element that specifies the location of the file containing the definition of an FME factory pipeline. The location of the file is specified with the `<file>` element's name attribute. In addition, optional pipeline directives may be specified through the `<directives>` element. The name and value for each optional directive are specified through expression sequences.

```

<pipeline>
  <file name="theFeatureProcessingPipeline.fmi"/>
  <directives>
    <directive>
      <name> ... some expression sequence ... </name>
      <value> ... some expression sequence ... </value>
    </directive>
    ...
    <directive> ... </directive>
  </directives>
</pipeline>

```

Note: The location of the pipeline file may be specified with an absolute or relative path. When the path is relative, then it is assumed that it is relative to the location of the xfMap document.

Each `<pipeline>` element may also contain an optional `<condition>` element (see section "*group mapping rule - condition element*") that allows or prevents a feature from entering into the pipeline.

The following illustrates the usage of group pipelines. Consider the following input XML and xfMap documents:

group.xml

```

<?xml version="1.0"?>
<group>
  <member id="290"/>
</group>

```

pipeline.xmp

```

<?xml version="1.0"?>
<xfMap>

```

```

<group-map>
  <mapping match="group">
    <!-- The group object contains one FME factory pipeline that is
         defined in the pipeline.fmi file. This files is in the
         same directory as the pipeline.xml xfMap.
         The features construted in the feature mapping rule
         below will enter this group and its pipeline for
         further processing.-->
    <apply-pipelines>
      <pipeline>
        <!-- The section titled 'condition element' describes the
             function of this element as a filter that prevents or
             allows features entering the the pipeline.
             Here only feature's having an attribute call id
             with the value of 290 are allowed into the pipeline
             other features do not enter, they just by pass it. -->
        <condition feature="@id='290'"/>
        <file name="pipeline.fmi"/>
      </pipeline>
    </apply-pipelines>
  </mapping>
</group-map>

<feature-map>
  <mapping match="member">
    <feature-type> <literal expr="member"/> </feature-type>
    <attributes>
      <attribute>
        <name> <literal expr="id"/> </name>
        <value> <extract expr="@id"/> </value>
      </attribute>
    </attributes>
  </mapping>
</feature-map>
</xfMap>

```

This is the FME factory defined in the *pipeline.fmi* file:

```

FACTORY_DEF * TeeFactory          \
INPUT FEATURE_TYPE member        \
OUTPUT FEATURE_TYPE member isCopy false \
OUTPUT FEATURE_TYPE member isCopy true

```

The FME features output by the XML Reader are:

```

+++++
Feature Type: `member'
Attribute: `id' has value `290'
Attribute: `isCopy' has value `false'
Attribute: `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `member'
Attribute: `id' has value `290'
Attribute: `isCopy' has value `true'
Attribute: `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====

```

Condition Element

Features that enter a *group* will also enter all its attribute sets and pipelines by default. Each attribute set and pipeline may contain an optional condition that allows an FME feature to enter it, or prevents and FME feature from entering it. The optional condition is represented in xFMap by the `<condition>` element:

```
<condition .../>
```

The `<condition>` element has two attributes named `feature` and `element` that take Boolean expressions as values. The `feature` attribute Boolean expression is evaluated on the attributes of the FME feature that entered the group. The `element` attribute Boolean expression is evaluated on the attribute of the element that matched the *group mapping rule*. At least one of the `feature` or `element` attributes must be present in the `<condition>` element. The following are the valid combinations:

```
<condition feature="..."/>
<condition element="..."/>
<condition feature="..." element="..."/>
```

The `<condition>` element also has an optional `type` attribute that specifies whether the `feature` and `element` Boolean expressions should form a conjunction or a disjunction. The valid values for the `type` attribute are `and` (for a conjunction), `or` (for a disjunction). Its default value is `and`:

```
<condition feature="..." type="and" element="..."/>
<condition feature="..." type="or" element="..."/>
```

The grammar for both the `feature` and the `element` Boolean expressions is:

```
booleanExpr = attrCondition
             | andExpr
             | orExpr
             | '(' booleanExpr ')'

andExpr = booleanExpr 'and' booleanExpr
orExpr = booleanExpr 'or' booleanExpr

attrCondition = '@' attrName( '+' | '-' )
              '@' attrName( '=' | '!=' ) '%' ? ( '"' | "'" ) attrValue( '"' | "'" )
```

Note: The XML Reader evaluates a Boolean expression in a right associative way. Use parentheses to indicate the intended precedence when using complex Boolean expressions.

Except for the `%` sign in the right-hand side of the `attrCondition` production, the grammar of the Boolean expression is identical to *mapping rule's match condition* (see the section titled "*mapping rules - the match expression*").

The `%` sign is used to access *let variables* inside the Boolean expression. *Let variables* are described in the section titled "*mapping rules (optional elements) - define element*".

Persistent Groups

A *group* is **persistent** if the *group mapping rule* that defines it has a `<persist>` element.

A *persistent group* that is not the last group in the *g-stack* will not be destroyed when its corresponding *mapping rule* de-activates.

Note: A group mapping rule defined under the `<group-map>` element may have a `<persist>` element, but this indication for the group to persist is always ignored by the XML Reader, this group, by construction, will always be the last one left in the *g-stack* (see *Group Construction and Destruction*).

When a *persistent group* is popped from the *g-stack* it can persist inside other *g-stack groups*. The `<persist>` element has an optional `in` attribute that specifies where it may persist. The valid values for this attribute are `parent-group` and `base-group` (where `parent_group` is the default value):

```
<persist in="...">
```

The `parent-group` of a *persistent group* is the *group* that will be at the top of the *g-stack* when the *persistent group* is popped. The `base-group` is the *group* that is at the base of the *g-stack*; it is the first *group* pushed into the stack; it is the *group* constructed from the activation of a *group mapping rule* that was defined under the `<group-map>` element.

FME features that are constructed by *feature mapping rules* will enter all the *groups* that are in the *g-stack* from the top until the bottom of the stack. If a *group* in the stack has *persistent groups*, then the features will first enter the *persistent groups* before entering into the *group's own attribute sets and pipelines*.

The following example shows why *persistent groups* are sometimes needed.

group_persist.xml

```
<?xml version="1.0"?>
<group>
  <group-property name="section">C-23</group-property>
  <group-property name="location">Z-Edifice</group-property>
  <group-property name="op-code">5801d-3</group-property>
  <member id="290"/>
  <member id="350"/>
  <member id="300"/>
</group>
```

We want to map each `<member>` element into an FME feature, but we'll also like to attach the information from each of the `<group-property>` elements to the member feature. The following xfMap document achieves this by the usage of persistent groups:

group_persist.xmp

```
<?xml version="1.0"?>
<!DOCTYPE xfMap SYSTEM "xfMap.dtd">
<xfMap>
  <group-map>
    <!-- We create a group here for the sole purpose of having a
         group to persist on. The groups constructed in the
         group-content-map may persist in this group.

         Features that enter this group will be processed
         by this group's 'persistent groups attribute sets'.
    -->
    <mapping match="group"/>
  </group-map>

  <group-content-map>
    <mapping match="group-property">
      <!-- The group is constructed and pushed into the g-stack when the
           group-property start-tag is read, when the end-tag is
           read, then the group is popped from the g-stack.

           The group is not destroyed, it will persist in its
           parent-group, in this case it is the group that
           is constructed in the group-map above.

           We make this group persist, since otherwise this group
           is destroyed when the group-property element end-tag
           is read so that this group attribute set will not be
           attached to any feature.-->
      <persist/>
      <apply-attribute-sets>
        <attribute-set>
          <attributes>
            <attribute>
              <name> <extract expr="@name"/> </name>
              <value> <extract expr="."/> </value>
            </attribute>
          </attributes>
        </attribute-set>
      </apply-attribute-sets>
    </mapping>
  </group-content-map>
```

```

    <feature-map>
      <mapping match="member">
        <feature-type>
          <literal expr="member-"/> <extract expr="@id"/>
        </feature-type>
      </mapping>
    </feature-map>
  </xfMap>

```

The two documents above make the XML Reader output the following FME features:

```

+++++
Feature Type: `member-290'
Attribute: `location' has value `Z-Edifice'
Attribute: `op-code' has value `5801d-3'
Attribute: `section' has value `C-23'
Attribute: `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `member-350'
Attribute: `location' has value `Z-Edifice'
Attribute: `op-code' has value `5801d-3'
Attribute: `section' has value `C-23'
Attribute: `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `member-300'
Attribute: `location' has value `Z-Edifice'
Attribute: `op-code' has value `5801d-3'
Attribute: `section' has value `C-23'
Attribute: `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====

```

A *group* actually contains two places where it may adopt *persistent groups*. By default the *group* adopts a *persistent group* in its low-priority list, the `<persist>` element has an optional priority attribute that specifies if a *persistent group* should be adopted into a group's low- or high-priority lists.

A *group* will always process FME features through its high-priority *persistent groups* before the low-priority ones. The valid values for the priority attribute are low and high, with low being the default value.

For example, the following *group mapping rule* defines a *persistent group* to persist in its parent group high-priority list:

```

<group-content-map>
  <mapping match="manifold">
    <persist in="parent-group" priority="high"/>
    ...
  </mapping>
</group-content-map>

```

Reference Mapping Rules

Reference mapping rules specify the construction of *xfMap reference-sets*, and they must be defined inside the `<reference-map>` or `<reference-content-map>` elements.

References allow an *xfMap* to store values that need to be accessed later in the input stream. All *reference mapping rules*, whether they're defined under the `<reference-map>` or the `<reference-content-map>` elements, have the same structure. Similar to *group mapping rules* but unlike *feature mapping rules*, every *reference mapping rule* when activated, regardless of where it was defined, triggers the XML Reader to construct a new *reference-set*.

Reference-Set Construction and Destruction

Let *R* be a *reference mapping rule* defining a set of references *S*, then:

- a. S is **constructed** when R is activated, and
- b. S is **destroyed** when R is de-activated, but only if:
 1. R was defined in the `<reference-map>` element, or
 2. S is not a *persistent reference-set*.

As with `xfMap` groups, constructed *reference-sets* can be pictured as residing in some sort of stack, we call this the **r-stack**, where the top of the stack houses the most immediate constructed *reference-set*. A destroyed *reference-set* is removed from the *r-stack* unless it was specified to be persistent.

Reference-Sets

A *reference-set* contains *references* that may be accessed by other mapping rules through the `<refexpr>` expression wherever an *expression sequence* is allowed. A **reference-set** is defined in a *reference mapping rule* by the `<references>` element.

The `<references>` element may contain zero or more `<reference>` elements. Each `<reference>` element defines a single *reference* in the set. In addition, each *reference* may belong to a particular named group, so that several *references* in a set may have the same name as long as they belong to different groups. An optional `group-name` attribute in a `<reference>` element names the group for a *reference*. *References* defined without a `group-name` belong in the default group of a *reference-set*.

```
<mapping match="...">
  <references>
    <!-- Defined 2 references in group-1 -->
    <reference group-name="group-1">
      ...
    </reference>
    <reference group-name="group-1">
      ...
    </reference>
    <!-- Define a reference in the default group -->
    <reference>
      ...
    </reference>
  </references>
</mapping>
```

A *reference* has a name and a value. The name and value for each *reference* are specified through expression sequences.

```
<reference>
  <name> ... some expression sequence ... </name>
  <value> ... some expression sequence ... </value>
</reference>
```

A *reference* has a *name* and a *value*. The *name* and *value* for each *reference* are specified through expression sequences.

```
<reference>
  <name> ... some expression sequence ... </name>
  <value> ... some expression sequence ... </value>
</reference>
```

The *name* along with the *reference's* *group-name* together form the handle for the interested *value* we wish to store. The value of a *reference* can be accessed with an *refexpr* expression, that is, the value for a stored *reference* can be accessed wherever an expression sequence is allowed.

The *r-stack*, the stack of constructed *reference-sets*, is searched from top to bottom for stored references. Therefore, newer *reference-sets* having the same *reference* handles as previous constructed and non-destroyed *reference-sets* always override older *references*.

Persistent Reference-Sets

A *reference-set* is removed from the *r-stack* and destroyed as soon as its originating mapping rule is deactivated. If there is a need to keep *references* for a longer period then its *reference-set* can be made to persist on the sets that

still reside in the *r-stack*. A *reference-set* whose originating mapping rule is being deactivated can be made to persist either at the top or at the bottom of the *r-stack*.

The `persist` attribute in the `<references>` element allows a *reference-set* to be persisted beyond the deactivation of its originating mapping rule. The valid values for the `persist` attribute are `true` and `false`. A *reference-set* is not persisted by default, so an absent `persist` attribute has the same effect as setting it to `false`.

```
<references persist="true|false">
...
</references>
```

By default, a persistent *reference-set* persists in the set at the top of the *r-stack*. This persistent set will be destroyed as soon as its containing set is also destroyed. For convenience, the `xfMap` allows a set to be persisted in the set at the bottom of the *r-stack*. The `<references>` element's optional `persist-in` attribute can be used to control this. The valid values for the `persist-in` attribute are `parent` and `base`. Setting the `persist-in` attribute to `base` allows a set to be persisted at the bottom of the *r-stack*. By default, a persistent *reference-set* will be persisted in the set at the top of the stack – this is the same as setting the `persist-in` attribute to `parent`.

```
<references persist="true" persist-in="parent|base">
...
</references>
```

A *reference-set* persisting in another *reference-set* overwrites the references in the host set.

refexpr Expressions

The **refexpr expression** allows access to stored reference values and is represented in the `xfMap` by the `<refexpr>` element.

The general syntax for the element is:

```
<refexpr expr="..." default="..." reference-group="..." r-stack="...">
  <arg> ... optional reference name as expr seq ... </arg>
  <arg> ... optional default value as expr seq ... </arg>
</refexpr>
```

The `expr` attribute denotes the name for a stored *reference*. The optional `default` attribute may contain the value returned by the expression in the case that a *reference* by that *name* is not found in any of the constructed *reference-sets*. This may occur if the *reference* was never stored or if the *reference-set* containing that *reference* was already destroyed.

Both `<arg>` child elements in the `<refexpr>` element are optional.

The `expr` attribute is required but it may be the empty string, in which case the first `<arg>` child element must be present, and the evaluated *expression sequence* for this first argument becomes the *reference's* name to search.

If the `default` attribute is absent or if it is present but its value is the empty string, then the second argument, if present, is evaluated to become the default value for the expression in the case that the *reference* is not found.

The `reference-group` attribute is optional and it specifies the *group-name* for a *reference* in a *reference-set*. If the `reference-group` attribute is absent, then the default group in a *reference-set* is assumed.

The `r-stack` attribute is optional and it specifies the starting place to search for values in the stack of *reference-sets*. The valid values for the `r-stack` attribute are `top`, `previous` and `bottom`, with `top` being the default value.

Example

Consider the following XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<zones>
  <class>
    <code v="HP">Historic Park</code>
    <code v="NE">Natural Environment Park</code>
    <code v="NP">National Park</code>
    <code v="PA">Protected Area</code>
  </class>
```

```

    <zone name="A-1" class="NE"/>
    <zone name="B-3" class="PA"/>
    <zone name="D-H" class="HP"/>
</zones>

```

The following xfMap maps each <zone> element into an FME feature. A class attribute is also added to the feature, but we use the class's long descriptive name rather than class's code as its value:

```

<?xml version="1.0" encoding="UTF-8"?>
<xfMap>
  <reference-map>
    <!-- construct an empty reference-set for other references-sets to persist in. This
         reference-set will only be destroyed when the </zones> end tag is read -->
    <mapping match="zones">
    </mapping>
  </reference-map>

  <reference-content-map>
  <!-- For each <code> element built a reference-set with one reference belonging to
         the "codes" reference group, the name of the reference will be the code id,
         the value in the "v" attribute, and the reference value is the descriptive
         name of the code. We also make this reference-set persist, it will persist in the
         parent reference-set, in this case, this is the reference-set that was constructed
         when the <zones> element got matched. -->
    <mapping match="class/code">
      <references persist="true">
        <reference group-name="codes">
          <name><extract expr="@v"/></name>
          <value><extract expr="."/></value>
        </reference>
      </references>
    </mapping>
  </reference-content-map>

  <feature-map>
    <mapping match="zone">
      <feature-type><literal expr="zone"/></feature-type>
      <attributes>
        <attribute>
          <name><literal expr="name"/></name>
          <value><extract expr="@name"/></value>
        </attribute>
        <attribute>
          <name><literal expr="class"/></name>
          <value>
            <refexpr expr="" reference-group="codes">
              <arg><extract expr="@class"/></arg>
            </refexpr>
          </value>
        </attribute>
      </attributes>
    </mapping>
  </feature-map>
</xfMap>

```

```

+++++
Feature Type: `zone'
Attribute(string): `class' has value `Natural Environment Park'
Attribute(string): `name' has value `A-1'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====

```

```

+++++
Feature Type: `zone'
Attribute(string): `class' has value `Protected Area'
Attribute(string): `name' has value `B-3'

```

```

Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `zone'
Attribute(string): `class' has value `Historic Park'
Attribute(string): `name' has value `D-H'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====

```

Mapping Rules (Optional Elements)

This section describes the optional elements of a mapping rule. All mapping rules, whether they are feature, group, or reference mapping rules may contain these elements.

Signature Element

The **signature** of a mapping rule serves two purposes:

1. *it identifies the mapping rule by name*, and
2. *it declares some named parameters*, which are passed into a mapping rule on activation.

The signature of a mapping rule is defined with the optional <signature> element. The <signature> element has a name attribute that defines the name of the mapping rule. The mapping rule's optional named parameters are defined via the <signature>'s <params> child element; this element contains a sequence of one or more <param> elements that specify the names of the parameters via their name attribute:

```

<mapping match="...">
  <signature name="myMappingRule"/>
  ...
</mapping>

<mapping match="...">
  <signature name="myOtherMappingRule">
    <params>
      <param name="firstParam"/>
      <param name="secondParam"/>
    </params>
  </signature>
</mapping>

```

myOtherMappingRule has two named parameters: firstParam, and secondParam. The value of a named parameters is a *expression sequence* and it may be access via the *parmval expression* element.

Expression Element (parmval Expressions)

The **parmval expression** allows access to the mapping rule's parameters by referring to them by name in its *expression string*. The *parmval expression* is represented in xfMap by the <parmval> element; its expr attribute holds the value of its *expression string*:

```
<parmval expr="the-name-of-a-mapping-rule-parameter"/>
```

The actual value for a parameter in an activated mapping rule R_1 must be supplied by a mapping rule R_0 , where R_0 is the mapping rule that got suspended when R_1 activated.

The expr value for the **parmval expression** may be optionally specified via a named argument. This allows the expr value to be constructed programatically by an expression sequence. The argument must be named 'expr' and it must take the following form:

```
<parmval><arg named="expr">...any expression sequence...</arg></parmval>
```

For example, the following mapping rule sets the feature type with a *parmval expression*:

```
<mapping match='...'>
  <signature name='myRule'>
```

```

        <params>
            <param name="firstParam"/>
            <param name="theFeatureType"/>
        </params>
    </signature>

    <feature-type>
        <parmval expr="theFeatureType"/>
        <literal expr="_"/>
        <parmval>
            <arg name="expr">
                <literal expr="firstParam"/>
            </arg>
        </parmval>
    </feature-type>
</mapping>

```

Use-Mappings Element (supplying the parameters)

The optional <use-mappings> element must be the last element in a mapping rule. This element contains a sequence of one or more <use> elements that identifies a mapping rule by its signature name through the name attribute:

```

<mapping match="...">
    ...
    <use-mappings>
        <use name="myMappingRule1"/>
        <use name="myMappingRule2"/>
    </use-mappings>
</mapping>

```

The above only specifies parameter-less mapping rules. To supply the parameters for a mapping rule, the <use> element may optionally have an <args> element. The <args> element can have one or more <arg> elements that take *expression sequences* as their values:

```

<mapping match="...">
    ...
    <use-mappings>
        <use name="myMappingRule1"/>
            <args>
                <arg> <!-- arg0 --> </arg>
                ...
                <arg> <!-- argN --> </arg>
            </args>
        </use>
    </use-mappings>
</mapping>

```

The myMappingRule1 mapping rule's signature must contain (N+1) named parameters.

Use-Mappings Element (limiting the active-search-set)

The <use-mappings> element of an executing mapping rule changes the default contents of an *active-search-set* (see the Contents of an active-search-set (Default Contents)). It limits the contents of an *active-search-set* to the mapping rules listed under its <use> elements.

For example, if the following is a *feature mapping rule*:

```

<mapping match="...">
    ...
    <use-mappings>
        <use name="mr1"/>
        <use name="mr2"/>
    </use-mappings>
</mapping>

```

Then, when the above *feature mapping rule* is executing, its *feature-search-set* (recall that the *feature-search-set* is the *feature mapping rule's active-search-set*) will only contain the *feature mapping rules* mr1 and mr2.

Consider the following input XML document:

drawing.xml

```
<?xml version="1.0"?>
<drawing>
  <figure>
    <color type="background">
      <component type="red">0.949</component>
      <component type="green">0.357</component>
      <component type="blue">0.283</component>
    </color>
    <color type="foreground">
      <component type="red">0.532</component>
      <component type="green">0.899</component>
      <component type="blue">0.521</component>
    </color>
  </figure>
</drawing>
```

The following xfMap document maps the above <figure> element into an FME feature:

drawing.xmp

```
<?xml version="1.0"?>
<!DOCTYPE xfMap SYSTEM "xfMap.dtd">

<xfMap>
  <feature-map>
    <mapping match="figure">
      <feature-type> <literal expr="figure"/> </feature-type>
    </mapping>
  </feature-map>

  <feature-content-map>
    <mapping match="color">
      <!-- The use-mappings element will limit the feature-search-set
           to contain only the mr_colors mapping rule.

           The value of the type attribute from the color element is
           passed as an argument to the mr_colors mapping rule.-->
      <use-mappings>
        <use name="mr_colors">
          <args>
            <arg> <extract expr="@type"/> </arg>
          </args>
        </use>
      </use-mappings>
    </mapping>

    <mapping match="component">
      <!-- The signature of this mapping rule has 1 named parameter
           called colorType, its value is passed as an argument from
           the mapping rule above. -->
      <signature name="mr_colors">
        <params>
          <param name="colorType"/>
        </params>
      </signature>

      <attributes>
        <attribute>
```

```

                <name>
                    <!-- Access the colorType parameter. -->
                    <paramval expr="colorType"/>
                    <literal expr="."/>
                    <extract expr="@type"/>
                </name>
                <value> <extract expr="."/> </value>
            </attribute>
        </attributes>

        <!-- NOTE: This mapping rule does not have a use-mappings element.
             when this mapping rule is executing the feature-search-set is
             set to its default contents. That is, all of the feature mapping
             rules defined under the feature-content-map -->
    </mapping>
</feature-content-map>
</xfMap>

```

The FME feature created is:

```

+++++
Feature Type: `figure'
Attribute: `background.blue' has value `0.283'
Attribute: `background.green' has value `0.357'
Attribute: `background.red' has value `0.949'
Attribute: `foreground.blue' has value `0.521'
Attribute: `foreground.green' has value `0.899'
Attribute: `foreground.red' has value `0.532'
Attribute: `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====

```

Define Element

The <define> element allows *expression sequences* to be named and referenced in other *expression sequences* through that name. It contains a sequence of one or more <let> elements. Each <let> element defines a **let definition**, and the name of the *let definition* is specified via the <let> element's name attribute while its value is an *expression sequence*:

```

<define>
  <let name="myExprSeq">
    <!-- some expression sequence -->
  </let>
  <let name="myOtherExprSeq">
    <!-- some expression sequence -->
  </let>
  ...
</define>

```

These let expression sequences may be accessed in two different ways: by the *defnval expressions*, or by the <condition> element's Boolean expression % token. The *defnval expression* is described in [defnval Expressions](#). This section describes the % token in the <condition> element's Boolean expressions.

Recall that unlike the mapping rule's match Boolean expression, the Boolean expressions from the <condition> element may contain an extra % token, which is reproduced for reference below.

```

booleanExpr =  attrCondition
              |  andExpr
              |  orExpr
              |  '( ' booleanExpr ' )'

andExpr = booleanExpr 'and' booleanExpr
orExpr = booleanExpr 'or' booleanExpr

```

```
attrCondition = '@'attrName('+ ' | '- ') |
 '@'attrName('= ' | '!=' )'%'? ( '"" | """)attrValue( "" | """)
```

The token % means to interpret the *attrValue* as the name of a *let definition*. The value of the let definition *attrValue*, an *expression sequence*, is then substituted for comparison.

The following example illustrates how the *let definitions* may be accessed through the <condition> element's Boolean expressions.

players.xml

```
<?xml version="1.0"?>
<players>
  <positions>
    <position name="Josephine" type="forward"/>
    <position name="Joan" type="backward"/>
  </positions>
  <player>
    <name>Josephine</name>
    <age>16</age>
  </player>
  <player>
    <name>Joan</name>
    <age>17</age>
  </player>
</players>
```

We would like to map each <player> element into an FME feature. We would also like to add a position attribute that describes the player's position to the FME feature. Notice that all position information precedes the data of the individual players; therefore we'll use *group mapping rules*:

players.xmp

```
<?xml version="1.0"?>
<xfMap>
  <group-map>
    <!-- We create this group so that the groups that are constructed
         out of the position elements can persist here. -->
    <mapping match="players"/>
  </group-map>

  <group-content-map>
    <mapping match="positions/position">
      <!-- A group is constructed for every position element, when
           the position element end-tag is read the group is defined
           to persist in its parent-group (i.e., the group defined
           in the group-map above). -->

      <define>
        <!-- This let definition will be access by the
             condition element below. We want to keep track
             of the name of this player that started this group. -->
        <let name="playerName"> <extract expr="@name"/> </let>
      </define>
      <persist/>
      <apply-attribute-sets>
        <attribute-set>
          <!-- Only features that have an attribute called
               'name' with its value equal to the value
               of the expression sequence denoted by 'playerName'
               will receive the attribute set. -->
          <condition feature="@name=%'playerName'"/>
          <attributes>
            <attribute>
```

```

        <name> <literal expr="position"/> </name>
        <value> <extract expr="@type"/> </value>
      </attribute>
    </attributes>
  </attribute-set>
</apply-attribute-sets>
</mapping>
</group-content-map>
<feature-map>
  <mapping match="player">
    <feature-type> <literal expr="player"/>
    <attributes>
      <attribute>
        <name> <literal expr="name"/> </name>
        <value> <extract expr="./name"/> </value>
      </attribute>
      <attribute>
        <name> <literal expr="age"/> </name>
        <value> <extract expr="./age"/> </value>
      </attribute>
    </attributes>
  </mapping>
</feature-map>
</xfMap>

```

The FME features output when the above *players.xml* and *players.xmp* are fed into the XML Reader are:

```

+++++
Feature Type: `player'
Attribute: `age' has value `16'
Attribute: `name' has value `Josephine'
Attribute: `position' has value `forward'
Attribute: `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `player'
Attribute: `age' has value `17'
Attribute: `name' has value `Joan'
Attribute: `position' has value `backward'
Attribute: `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====

```

Notice that the player FME features contain the position element with their correct attribute values.

More Expression Elements

- defnval Expressions
- strexpr Expressions
- tclexpr Expressions
- counter Expressions
- maprule Expressions
- selexpr Expressions
- matched Expressions
- keyword Expressions
- fmefunc Expressions

- logexpr Expressions
- comparison Expressions

defnval Expressions

The defnval expression allows access to the value of a *let variable* in an *expression sequence*. The expr attribute of a <defnval> element must be equal to the name of a *let variable* defined under the mapping rule's <define> element.

The following *feature mapping rule* shows how to access the value of a *let variable* in an *expression sequence*:

```
<mapping match="...">
  ...
  <define>
    <let name="theFeatureType">
      <!-- some expression sequence -->
    </let>
  </define>
  ...
  <feature-type>
    <defnval expr="theFeatureType"/>
  </feature-type>
  ...
</mapping>
```

strexpr Expressions

This expression provides string processing capabilities on an expression sequence. The strexpr expression has the following general form:

```
<strexpr expr="...">
  <arg> <!-- some expression sequence --> </arg>
  <arg> <!-- some expression sequence --> </arg>
  ...
  <arg> <!-- some expression sequence --> </arg>
</strexpr>
```

The value of this expression depends on the string operation that is specified through the *expression string*, i.e., the expr attribute. This also dictates the number of arguments (i.e., the number of <arg> elements) that the strexpr expression should have.

The following lists the available operations and their arguments.

charAt:

```
<strexpr expr="charAt">
  <arg> <!-- source string --> </arg>
  <arg> <!-- index --> </arg>
  --> </arg>
</strexpr>
```

It returns the character of the *source string* at the specified *index*. The *index* must be in 0 and (the length of the *source string* minus 1).

contains:

```
<strexpr expr="contains">
  <arg> <!-- source string --> </arg>
  <arg> <!-- string to search --> </arg>
  <arg> <!-- value to return if true --> </arg>
  <arg> <!-- value to return if false --> </arg>
</strexpr>
```

Evaluates containment of the *string to search* within the *source string*. It returns the evaluated expression sequence for the third argument when the *source string* contains the *string to search*, otherwise it returns the evaluated expression sequence for the fourth argument.

extract:

```

<strexpr expr="extract">
  <arg> <!-- source string --> </arg>
  <arg> <!-- start --> </arg>
  <arg> <!-- length --> </arg>
</strexpr>

```

Returns the characters that are in the range *start* and (*start* + *length*) of the *source string*.

findAndReplace:

```

<strexpr expr="findAndReplace">
  <arg> <!-- source string --> </arg>
  <arg> <!-- old text --> </arg>
  <arg> <!-- new text --> </arg>
  <arg> <!-- string to return if no replace --> </arg>
</strexpr>

```

Replaces all the occurrences of *old text* with *new text* in the *source string*. The 4th argument is optional, it allows the expression to return an alternate evaluated expression sequence when no *old text* is present in the *source string*. Both *new text* and *old text* are fixed strings. Use *regexReplace* for regular expression support.

first:

```

<strexpr expr="first">
  <arg> <!-- source string --> </arg>
  <arg> <!-- text to search --> </arg>
</strexpr>

```

Returns the index of the first occurrence of the *text to search* in the *source string*, or the empty string if the *text to search* is not found.

last:

```

<strexpr expr="last">
  <arg> <!-- source string --> </arg>
  <arg> <!-- text to search --> </arg>
</strexpr>

```

Returns the index of the last occurrence of the *text to search* in the *source string*, or the empty string if the *text to search* is not found.

leftOf:

```

<strexpr expr="leftOf">
  <arg> <!-- source string --> </arg>
  <arg> <!-- index --> </arg>
</strexpr>

```

Returns the sub-string left of the specified *index* in the *source string*; the empty string is returned if the *index* is out of range.

leftOfString:

```

<strexpr expr="leftOfString">
  <arg> <!-- source string --> </arg>
  <arg> <!-- separator --> </arg>
</strexpr>

```

Returns the sub-string left of the first separator in the source string; the empty string is returned if the separator is absent in the source string.

length:

```

<strexpr expr="length">
  <arg> <!-- source string --> </arg>
</strexpr>

```

It returns the character length of the *source string*.

normalizeWhitespace:

```
<strexpr expr="normalizeWhitespace">
  <arg> <!-- source string --> </arg>
</strexpr>
```

Returns the source string with all consecutive whitespace, tabs, line feeds, and spaces collapsed into a single space.

padLeading:

```
<strexpr expr="padLeading">
  <arg> <!-- source string --> </arg>
  <arg> <!-- target length --> </arg>
  <arg> <!-- pad char --> </arg>
</strexpr>
```

Pads the beginning of the *source string* with the *pad char* character until the resulting string length equals *target length*. Only the first character of *pad char* is taken when *pad char* length is > 1.

padTrailing:

```
<strexpr expr="padTrailing">
  <arg> <!-- source string --> </arg>
  <arg> <!-- target length --> </arg>
  <arg> <!-- pad char --> </arg>
</strexpr>
```

Pads the end of the *source string* with the *pad char* character until the resulting string length equals *target length*. Only the first character of *pad char* is taken when *pad char* length is > 1.

regexMatch:

```
<strexpr expr="regexMatch">
  <arg> <!-- regular expression --> </arg>
  <arg> <!-- source string --> </arg>
</strexpr>
```

Returns either the string 'true' or 'false' depending on whether the regular expression matches the entire source string. The regular expression supports Perl Compatible Regular Expressions (PCRE).

regexReplace:

```
<strexpr expr="regexReplace">
  <arg> <!-- matching regular expression --> </arg>
  <arg> <!-- source string --> </arg>
  <arg> <!-- replacement regular expression --> </arg>
  <arg> <!-- optional return value if no matches --> </arg>
</strexpr>
```

Returns the result of replacing all occurrences of *matching regular expression* with *replacement regular expression* in *source string*. If there are no matches and if the optional 4th argument is absent then the *source string* is returned untouched, otherwise the value for the 4th argument is returned. The regular expressions support Perl Compatible Regular Expressions (PCRE).

rightOf:

```
<strexpr expr="rightOf">
  <arg> <!-- source string --> </arg>
  <arg> <!-- index --> </arg>
</strexpr>
```

Returns the sub-string right of the specified *index* in the *source string*; the empty string is returned if the *index* is out of range

rightOfString:

```

<strexpr expr="rightOfString">
  <arg> <!-- source string --> </arg>
  <arg> <!-- separator --> </arg>
</strexpr>

```

Returns the sub-string right of the first separator in the source string; the empty string is returned if the separator is absent in the source string.

remove:

```

<strexpr expr="remove">
  <arg> <!-- source string --> </arg>
  <arg> <!-- start -->
  <arg> <!-- length --> </arg> </arg>
</strexpr>

```

Removes the characters in the range of *start* to (*start* + *length*) from the *source string*.

toLower:

```

<strexpr expr="toLower">
  <arg> <!-- source string --> </arg>
</strexpr>

```

Returns the source string converted to lowercase.

toUpper:

```

<strexpr expr="toUpper">
  <arg> <!-- source string --> </arg>
</strexpr>

```

Returns the source string converted to uppercase.

trim:

```

<strexpr expr="trim">
  <arg> <!-- source string --> </arg>
  <arg> <!-- char to trim --> </arg>
</strexpr>

```

Trims all of the leading and trailing *char to trim* characters from the *source string*. If value of the *char to trim* argument is `whitespace`, for example, `<arg> <literal expr="whitespace"/> </arg>`, then leading and trailing whitespaces will be removed from the *source string*.

trimLeading:

```

<strexpr expr="trimLeading">
  <arg> <!-- source string --> </arg>
  <arg> <!-- char to trim --> </arg>
</strexpr>

```

Trims all of the leading *char to trim* characters from the *source string*.

trimTrailing:

```

<strexpr expr="trimTrailing">
  <arg> <!-- source string --> </arg>
  <arg> <!-- char to trim --> </arg>
</strexpr>

```

Trims all of the trailing *char to trim* characters from the *source string*.

Example

The example below illustrates several of the *strexpr expression* operations. Please refer to the comments in the *strexpr.xmp* for the details.

strexpr.xml

```
<?xml version="1.0"?>
<strings>
  <id>78</id>
  <date>05,12,1999</date>
</strings>
```

strexpr.xmp

```
<?xml version="1.0"?>
<!DOCTYPE xfMap SYSTEM "xfMap.dtd">

<xfMap>
  <feature-map>
    <mapping match="strings">
      <define>
        <let name="featType">
          <!-- define the 'featType' expression sequence
                to be an id of exactly 15 digits. Pad it with leading
                zeros until the results length is 15 digits. -->
          <strexpr expr="padLeading">
            <arg> <extract expr="./id"/> </arg>
            <arg> <literal expr="15"/> </arg>
            <arg> <literal expr="0"/> </arg>
          </strexpr>
        </let>
      </define>

      <feature-type> <defnval expr="featType"/> </feature-type>

      <attributes>
        <attribute>
          <name> <literal expr="date"/> </name>
          <value>
            <!-- Replace all the commas in the date element content
                    by a dash -->
            <strexpr expr="findAndReplace">
              <arg> <extract expr="./date"> </arg>
              <arg> <literal expr=","/> </arg>
              <arg> <literal expr="-"/> </arg>
            </strexpr>
          </value>
        </attribute>

        <attribute>
          <name> <literal expr="orig-id"/> </name>
          <value>
            <!-- trim off all leading 0's from the featType
                    expression sequence -->
            <strexpr expr="trimLeading">
              <arg> <defnval expr="featType"/> </arg>
              <arg> <literal expr="0"/> </arg>
            </strexpr>
          </value>
        </attribute>

        <attribute>
          <name> <literal expr="featType-length"/> </name>
          <value>
            <!-- the length of the featType expression sequence -->
            <strexpr expr="length">
              <arg> <defnval expr="featType"/> </arg>
            </strexpr>
          </value>
        </attribute>
      </attributes>
    </mapping>
  </feature-map>
</xfMap>
```

```

                                </attribute>
                            </attributes>
                        </mapping>
                    </feature-map>
                </xfMap>

```

FME feature constructed:

```

+++++
Feature Type: `000000000000078'
Attribute: `date' has value `05-12-1999'
Attribute: `featType-length' has value `15'
Attribute: `orig-id' has value `78'
Attribute: `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====

```

tclexpr Expressions

The tclexpr expression provides a limited set of Tcl processing capabilities on the expression sequences. It has the following general form:

```

<tclexpr expr="...">
    <arg> <!-- some expression sequence --> </arg>
    <arg> <!-- some expression sequence --> </arg>
    ...
    <arg> <!-- some expression sequence --> </arg>
</tclexpr>

```

The value of this expression depends on the Tcl command that is specified through the *expression string* (i.e., the expr attribute); this also dictates the number of arguments (i.e., the <arg> elements) that the *tclexpr expression* should have.

The following lists the available Tcl commands and their arguments.

expr:

```

<tclexpr expr="expr">
    <arg> <!-- some expression sequence --> </arg>
    ...
</tclexpr>

```

Concatenates all of the arguments and evaluates the result as a Tcl expression. The number of <arg> elements following the first one depends on which Tcl expression is specified.

concat:

```

<tclexpr expr="concat">
    <arg> <!-- arg0 --> </arg>
    ...
    <arg> <!-- argN --> </arg>
</tclexpr>

```

Returns a concatenated list by treating all of the arguments *arg0* to *argN* as lists.

join:

```

<tclexpr expr="join">
    <arg> <!-- source list --> </arg>
    <arg> <!-- (optional) join string --> </arg>
</tclexpr>

```

Returns a string that is the concatenated elements of the *source list*. An optional *join string* may be specified to separate the concatenated elements. This *join string* defaults to a single space when it is not specified.

lindex:

```

<tclexpr expr="lindex">
  <arg> <!-- source list --> </arg>
  <arg> <!-- index --> </arg>
</tclexpr>

```

Returns the *index* item from the *source list*. The *index* starts at 0, and can be 'end' so that it returns the last item of the *source list*.

linsert:

```

<tclexpr expr="linsert">
  <arg> <!-- source list --> </arg>
  <arg> <!-- index --> </arg>
  <arg> <!-- element0 --> </arg>
  ...
  <arg> <!-- elementN --> </arg>
</tclexpr>

```

Insert the elements *element0* ... *elementN* into the source list starting at the specified index. An index of 0 inserts at the beginning while an index of 'end' inserts at the end of the source list.

list:

```

<tclexpr expr="list">
  <arg> <!-- arg0 --> </arg>
  ...
  <arg> <!-- argN --> </arg>
</tclexpr>

```

Returns a list containing the given arguments *arg0* to *argN*.

llength:

```

<tclexpr expr="llength">
  <arg> <!-- source list --> </arg>
</tclexpr>

```

Return the number of elements in the *source list*.

lrange:

```

<tclexpr expr="lrange">
  <arg> <!-- source list --> </arg>
  <arg> <!-- first --> </arg>
  <arg> <!-- last --> </arg>
</tclexpr>

```

Returns a list consisting of the *source list* elements from indices *first* to *last*. The indices start from 0; the last index can be 'end' to refer to the last element of the *source list*.

lreplace:

```

<tclexpr expr="lreplace">
  <arg> <!-- source list --> </arg>
  <arg> <!-- first --> </arg>
  <arg> <!-- last --> </arg>
  <arg> <!-- element0 --> </arg>
  ...
  <arg> <!-- elementN --> </arg>
</tclexpr>

```

Replaces the elements in the *source list* having the indices *first* through *last* with the given elements *element0* ... *elementN*. If no elements are supplied, then the list elements within the indices are deleted.

lsearch:

```

<tclexpr expr="lsearch">
  <arg> <!-- (optional) search mode --> </arg>
  <arg> <!-- source list --> </arg>
  <arg> <!-- search pattern --> </arg>
</tclexpr>

```

Searches the *source list* for an element that matches the *search pattern*. If it is found, it returns the index of the matching element in the *source list*; otherwise it returns -1. The valid values for the optional *search mode* are: *-exact* (use exact matching), *-glob* (use glob pattern matching), and *-regexp* (use regular expression matching).

lsort:

```

<tclexpr expr="lsort">
  <arg> <!-- (optional) sort options --> </arg>
  <arg> <!-- source list --> </arg>
</tclexpr>

```

Sorts the elements in the *source list*. The valid values for the optional *sort options* are:

- *ascii* (sort by ASCII collation order)
- *dictionary* (sort by dictionary order)
- *integer* (compare elements as integers)
- *real* (compare elements as floating points)
- *increasing* (sort in increasing order)
- *decreasing* (sort in decreasing order)

split:

```

<tclexpr expr="split">
  <arg> <!-- source string --> </arg>
  <arg> <!-- (optional) separators --> </arg>
</tclexpr>

```

Splits the *source string* into a Tcl list. The elements in the string are split if they are separated by any of the characters in *separators*. The *separators* argument is optional; when it is not specified, then the default separator is whitespace.

string:

```

<tclexpr expr="string">
  <arg> <!-- option --> </arg>
  ...
</tclexpr>

```

Performs string operations based on *option*; this value also dictates the number of arguments that follow it.

The valid values for *option* are:

compare

```

<tclexpr expr="string">
  <arg> <literal expr="compare"/> </arg>
  <arg> <!-- string1 --> </arg>
  <arg> <!-- string2 --> </arg>
</tclexpr>

```

Compares the strings *string1* and *string2* lexicographically. Returns -1 if *string1* is less than *string2*, 0 if equal, or 1 if greater.

first

```

<tclexpr expr="string">
  <arg> <literal expr="first"/> </arg>
  <arg> <!-- string1 --> </arg>

```

```
    <arg> <!-- string2 --> </arg>
</tclexpr>
```

Returns the index of the first occurrence of *string1* in *string2*, or -1 if there are no occurrences.

index

```
<tclexpr expr="string">
  <arg> <literal expr="index"/> </arg>
  <arg> <!-- source string --> </arg>
  <arg> <!-- char index --> </arg>
</tclexpr>
```

Returns the character in *source string* that has index *char index*, else the empty string is returned if *char index* is out of range.

last

```
<tclexpr expr="string">
  <arg> <literal expr="last"/> </arg>
  <arg> <!-- string1 --> </arg>
  <arg> <!-- string2 --> </arg>
</tclexpr>
```

Returns the index of the last occurrence of *string1* in *string2*, else -1 if there are no occurrences.

length

```
<tclexpr expr="string">
  <arg> <literal expr="length"/> </arg>
  <arg> <!-- source string --> </arg>
</tclexpr>
```

Returns the length of the *source string*.

match

```
<tclexpr expr="string">
  <arg> <literal expr="match"/> </arg>
  <arg> <!-- pattern --> </arg>
  <arg> <!-- source string --> </arg>
</tclexpr>
```

Returns 1 if the *source string* matches the glob *pattern*, else 0 is returned.

range

```
<tclexpr expr="string">
  <arg> <literal expr="range"/> </arg>
  <arg> <!-- source string --> </arg>
  <arg> <!-- first --> </arg>
  <arg> <!-- last --> </arg>
</tclexpr>
```

Returns the substring of *source string* consisting of the characters from the index *first* through the index *last*. *last* can be the string 'end'.

tolower

```
<tclexpr expr="string">
  <arg> <literal expr="tolower"/> </arg>
  <arg> <!-- source string --> </arg>
</tclexpr>
```

Returns the *source string* converted to lowercase.

toupper

```

<tclexpr expr="string">
  <arg> <literal expr="toupper"/> </arg>
  <arg> <!-- source string --> </arg>
</tclexpr>

```

Returns the *source string* converted to uppercase.

trim

```

<tclexpr expr="string">
  <arg> <literal expr="trim"/> </arg>
  <arg> <!-- source string --> </arg>
  <arg> <!-- (optional) chars to trim --> </arg>
</tclexpr>

```

Returns the *source string* with the leading and trailing characters from the set *chars to trim* removed. The *chars to trim* argument is optional; when it is not specified, it defaults to the whitespace characters.

trimleft

```

<tclexpr expr="string">
  <arg> <literal expr="trimleft"/> </arg>
  <arg> <!-- source string --> </arg>
  <arg> <!-- (optional) chars to trim --> </arg>
</tclexpr>

```

Returns the *source string* with the leading characters from the set *chars to trim* removed. The *chars to trim* argument is optional; when it is not specified, it defaults to the whitespace characters.

trimright

```

<tclexpr expr="string">
  <arg> <literal expr="trimright"/> </arg>
  <arg> <!-- source string --> </arg>
  <arg> <!-- (optional) chars to trim --> </arg>
</tclexpr>

```

Returns the *source string* with the trailing characters from the set *chars to trim* removed. The *chars to trim* argument is optional; when it is not specified, it defaults to the whitespace characters.

wordend

```

<tclexpr expr="string">
  <arg> <literal expr="wordend"/> </arg>
  <arg> <!-- source string --> </arg>
  <arg> <!-- index --> </arg>
</tclexpr>

```

Returns the index after a word for which *index* falls in the *source string*. A word is assumed to be delimited by whitespace.

wordstart

```

<tclexpr expr="string">
  <arg> <literal expr="wordstart"/> </arg>
  <arg> <!-- source string --> </arg>
  <arg> <!-- index --> </arg>
</tclexpr>

```

Returns the index before a word for which *index* falls in the *source string*. A word is assumed to be delimited by whitespace.

Example

The example below illustrates several of the *tclexpr expression* operations. Please refer to the comments in the *tclexpr.xmp* for details.

tclexpr.xml

```

<?xml version="1.0"?>
<strings>
  <str1>faerie</str1>
  <str2>queene</str2>
  <str3>the</str3>
</strings>

```

tcexpr.xmp

```

<?xml version="1.0" encoding="UTF-8"?>
<xfMap>
  <feature-map>
    <mapping match="strings">
      <define>
        <let name="str1"> <extract expr="./str1"/> </let>
        <let name="str2"> <extract expr="./str2"/> </let>
        <let name="str3"> <extract expr="./str3"/> </let>
        <let name="theList">
          <literal expr=""/>
          <tclexpr expr="list">
            <arg> <defnval expr="str3"/> </arg>
            <arg> <defnval expr="str1"/> </arg>
            <arg> <defnval expr="str2"/> </arg>
          </tclexpr>
          <literal expr=""/>
        </let>
      </define>

      <feature-type> <defnval expr="theList"/> </feature-type>

      <attributes>
        <attribute>
          <name> <literal expr="list-length"/> </name>
          <value>
            <tclexpr expr="length">
              <arg> <defnval expr="theList"/> </arg>
            </tclexpr>
          </value>
        </attribute>

        <attribute>
          <name> <literal expr="sorted-list"/> </name>
          <value>
            <tclexpr expr="lsort">
              <arg> <defnval expr="theList"/> </arg>
            </tclexpr>
          </value>
        </attribute>

        <attribute>
          <name> <literal expr="eval-expr(1+3+4+3+1)"/> </name>
          <value>
            <tclexpr expr="expr">
              <arg> <literal expr="1+3+4+3+1"/> </arg>
            </tclexpr>
          </value>
        </attribute>

        <attribute>
          <name> <literal expr="upcase"/> </name>
          <value>
            <tclexpr expr="string">
              <arg> <literal expr="toupper"/> </arg>
              <arg> <defnval expr="theList"/> </arg>
            </tclexpr>
          </value>
        </attribute>
      </attributes>
    </mapping>
  </feature-map>
</xfMap>

```

```

        </tcl:expr>
      </value>
    </attribute>
  </attributes>
</mapping>
</feature-map>
</xfMap>

```

FME feature constructed:

```

+++++
Feature Type: `the faerie queene'
Attribute: `eval-expr(1+3+4+3+1)' has value `12'
Attribute: `list-length' has value `3'
Attribute: `sorted-list' has value `faerie queene the'
Attribute: `upcase' has value `THE FAERIE QUEENE'
Attribute: `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====

```

counter Expressions

The counter expression provides counting capabilities on the expression sequences. It has the following general form:

```
<counter expr="..." start-value="..." modulo="..." scope="..." peek="..." />
```

Or

```

<counter>
  <arg> <!-- expr --> </arg>
  <arg> <!-- start-value --> </arg>
  <arg> <!-- modulo --> </arg>
  <arg> <!-- scope --> </arg>
  <arg> <!-- peek --> </arg>
</counter>

```

A *counter expression* counts integral values. Upon evaluation the *counter expression* will return its current integral value and then update its counter by 1. The *expression string* (i.e., the *expr* attribute) names the counter.

All properties are optional (and default to the empty string for *expr*, zero for *modulo* and *start-value*, and local for *scope*). The *start-value* specifies the initial value for the *counter expression* while *modulo* allows clock-like arithmetic when the counter value is updated. A modulo of zero implies no *modulo* specification. A counter is by default bound to the lifetime of a mapping rule activation. To bound the counter beyond this lifetime the *scope* attribute can be used. The valid values for the *scope* property are *local*, which is the default, *parent* which uses the scope of the parent node, and *xfMap*. Specifying *xfMap* for the *scope* bounds the counter to the *xfMap*. To access the value of a counter without updating its value the optional *peek* attribute, whose default value is *false*, should be set to *true*.

The ability to specify properties of the *counter* as arguments allows these properties to be determined dynamically based on the document being read. If an argument evaluates to the empty string, then its value is ignored (and the default is used, or the previously established value in a counter with non-*local* scope).

Example

The example below illustrates several of the *counter expression* operations.

bin.xml

```

<?xml version="1.0"?>
<bin>
  <item>I'm some sort of item inside this bin.</item>
  <item>what type of item may I be?</item>
  <item>I wouldn't know.</item>
  <item>why should I?</item>
  <item>Ask the bin.</item>
</bin>

```

counter.xmp

```
<?xml version="1.0"?>
<!DOCTYPE xfMap SYSTEM "xfMap.dtd">
<xfMap>
  <group-map>
    <mapping match="bin">
      <apply-attribute-sets>
        <attribute-set>
          <attribute>
            <name> <literal expr="item-order-in-bin"/> </name>
            <value><counter expr="my-counter-name"/> </value>
          </attribute>
          <attribute>
            <name> <literal expr="count-modulo-2"/> </name>
            <value><counter expr="mod-2-count modulo="2"/> </value>
          </attribute>
          <attribute>
            <name> <literal expr="some-other-counter"/> </name>
            <value>
              <counter expr="some-counter" start-v
            </value>
          </attribute>
        </attribute-set>
      </apply-attribute-sets>
    </mapping>
  </group-map>
  <feature-map>
    <mapping match="item">
      <feature-type> <literal expr="item"/> </feature-type>
      <attributes>
        <attribute>
          <name> <literal expr="value"/> </name>
          <value><extract expr="."/> </value>
        </attribute>
      </attributes>
    </mapping>
  </feature-map>
</xfMap>
```

FME features constructed:

```
+++++
Feature Type: `item'
Attribute(string): `count-modulo-2' has value `0'
Attribute(string): `item-order-in-bin' has value `0'
Attribute(string): `some-other-count' has value `5'
Attribute(string): `value' has value `I'm some sort of item inside this bin.'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `item'
Attribute(string): `count-modulo-2' has value `1'
Attribute(string): `item-order-in-bin' has value `1'
Attribute(string): `some-other-count' has value `6'
Attribute(string): `value' has value `what type of item may I be?'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `item'
Attribute(string): `count-modulo-2' has value `0'
Attribute(string): `item-order-in-bin' has value `2'
```

```

Attribute(string): `some-other-count' has value `7'
Attribute(string): `value' has value `I wouldn't know.'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `item'
Attribute(string): `count-modulo-2' has value `1'
Attribute(string): `item-order-in-bin' has value `3'
Attribute(string): `some-other-count' has value `5'
Attribute(string): `value' has value `why should I?'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `item'
Attribute(string): `count-modulo-2' has value `0'
Attribute(string): `item-order-in-bin' has value `4'
Attribute(string): `some-other-count' has value `6'
Attribute(string): `value' has value `Ask the bin.'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====

```

maprule Expressions

The *maprule* expression provides information about mapping rules. It is represented in the xfMap document with the `<maprule>` element, and it has the following general form:

```
<maprule expr="..."/>
```

The *expression string* (i.e., the `expr` attribute) specifies the type of mapping rule information.

The following lists the valid values the *expression string* may take:

activate-count:

```
<maprule expr="activate-count"/>
```

Returns the number of times a mapping rule has been activated.

Example

The example below illustrates several of the *counter expression* operations.

bin.xml

```

<?xml version="1.0"?>
<bin>
  <item>I'm some sort of item inside this bin.</item>
  <item>what type of item may I be?</item>
  <item>I wouldn't know.</item>
  <item>why should I?</item>
  <item>Ask the bin.</item>
</bin>

```

maprule.xmp

```

<?xml version="1.0"?>
<!DOCTYPE xfMap SYSTEM "xfMap.dtd">
<xfMap>
  <feature-map>
    <mapping match="item">
      <feature-type> <literal expr="item"/> </feature-type>
      <attributes>
        <attribute>
          <name> <literal expr="value"/> </name>

```

```

        <value><extract expr="."/> </value>
      </attribute>
    </attribute>
    <name> <literal expr="mapping-rule-activate-count"/> </name>
    <value><maprule expr="activate-count"/> </value>
  </attribute>
</attributes>
</mapping>
</feature-map>
</xfMap>

```

FME features constructed:

```

+++++
Feature Type: `item'
Attribute(string): `mapping-rule-activate-count' has value `1'
Attribute(string): `value' has value `I'm some sort of item inside this bin.'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `item'
Attribute(string): `mapping-rule-activate-count' has value `2'
Attribute(string): `value' has value `what type of item may I be?'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `item'
Attribute(string): `mapping-rule-activate-count' has value `3'
Attribute(string): `value' has value `I wouldn't know.'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `item'
Attribute(string): `mapping-rule-activate-count' has value `4'
Attribute(string): `value' has value `why should I?'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `item'
Attribute(string): `mapping-rule-activate-count' has value `5'
Attribute(string): `value' has value `Ask the bin.'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====

```

selexpr Expressions

The *selexpr* expression returns the first, last, or the *i*th non-empty evaluated expression sequence from its argument list. It is represented in the xfMap document with the <selexpr> element, and it has the following general form:

```

<selexpr expr="...">
  <arg> <!-- some expression sequence --> </arg>
  <arg> <!-- some expression sequence --> </arg>
  ...
  <arg> <!-- some expression sequence --> </arg>
</selexpr>

```

The *expression string* (i.e., the *expr* attribute) specifies the first, last, or *i*th evaluated non-empty expression sequence to be returned. The valid values for the *expression string* are first, last, or a positive integer between 1 and *k*, where *k* is the number of arguments.

Example

items.xml

```
<?xml version="1.0"?>
<items>
  <item>
    <primary-id>p9384</primary-id>
    <alternate-id></alternate-id>
  </item>
  <item>
    <primary-id></primary-id>
    <alternate-id>a2046</alternate-id>
  </item>
  <item>
    <primary-id></primary-id>
    <alternate-id></alternate-id>
  </item>
</items>
```

selexpr.xmp

```
<?xml version="1.0"?>
<xfMap>
  <feature-map>
    <mapping match="item">
      <feature-type>
        <selexpr expr="first"/>
        <arg> <extract expr="./primary-id"/> </arg>
        <arg> <extract expr="./alternate-id"/> </arg>
        <arg> <literal expr="no-id"/> </arg>
      </selexpr>
    </feature-type>
  </mapping>
</feature-map>
</xfMap>
```

FME features constructed:

```
+++++
Feature Type: `p9384'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `a2046'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `no-id'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
```

matched Expressions

The *matched* expression returns the mapping rule's matched element's local-name, namespace-prefix, namespace-uri, QName, or sequence number. This expression is useful to retrieve the name of the matched element in the case that a wildcard was used in the **match expression**, or to determine a unique identifier for an element. It is represented in the xfMap document with the <matched> element, and it has the following general form:

```
<matched expr="..." ancestor="..." />
```

The *expression string* (i.e., the expr attribute) specifies whether the local-name, the namespace-prefix, the namespace-uri, QName, or the sequence number of the matched element is to be returned. The valid values for the *expres-*

sion string are local-name, prefix, uri, qname, and sequence. The sequence number is a period-separated list of numbers which identify the path from the root of the xml tree to the child element where the matched expression is evaluated.

The `ancestor` attribute is optional. Its value indicates which ancestor element the `expr` attribute applies to. The valid values for the `ancestor` attribute are `self`, `parent`, `grandparent`, or a non-negative number, with `self`, `parent`, and `grandparent` being equivalent to 0, 1, and 2, respectively. The default value for the `ancestor` attribute is `self`.

Example

three_players.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<players xmlns="http://schemas.sports.com/players"
  xmlns:pl2="http://schemas.sports.com/2/players"
  xmlns:pl3="http://schemas.sports.com/5/players">

  <Laura> <age>24</age> </Laura>
  <pl2:Sharen> <age>27</age> </pl2:Sharen>
  <Claudia> <age>28</age> </Claudia>

</players>
```

three_players.xmp

```
<?xml version="1.0" encoding="UTF-8"?>
<xfMap xmlns:pl="http://schemas.sports.com/players">
  <feature-map>
    <mapping match="pl:players/*">
      <feature-type> <literal expr="player"/> </feature-type>
      <attributes>
        <attribute>
          <name> <literal expr="local-name"/> </name>
          <value> <matched expr="local-name"/> </value>
        </attribute>
        <attribute>
          <name> <literal expr="ns-prefix"/> </name>
          <value> <matched expr="prefix"/> </value>
        </attribute>
        <attribute>
          <name> <literal expr="ns-uri"/> </name>
          <value> <matched expr="uri"/> </value>
        </attribute>
        <attribute>
          <name> <literal expr="QName"/> </name>
          <value> <matched expr="qname"/> </value>
        </attribute>
        <attribute>
          <name> <literal expr="Sequence Number"/> </name>
          <value> <matched expr="sequence"/> </value>
        </attribute>
        <attribute>
          <name> <literal expr="Parent-QName"/> </name>
          <value> <matched expr="qname" ancestor="parent"/> </value>
        </attribute>
      </attributes>
    </mapping>
  </feature-map>
</xfMap>
```

FME features constructed:

```
+++++
Feature Type: `player'
Attribute(string): `QName' has value `Laura'
Attribute(string): `Sequence Number' has value `1.3'
Attribute(string): `local-name' has value `Laura'
```

```

Attribute(string): `ns-prefix' has value ``
Attribute(string): `ns-uri' has value `http://schemas.sports.com/players'
Attribute(string): `Parent-QName' has value `players'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `player'
Attribute(string): `QName' has value `p12:Sharen'
Attribute(string): `local-name' has value `Sharen'
Attribute(string): `ns-prefix' has value `p12'
Attribute(string): `ns-uri' has value `http://schemas.sports.com/2/players'
Attribute(string): `Parent-QName' has value `players'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
+++++
Feature Type: `player'
Attribute(string): `QName' has value `Claudia'
Attribute(string): `local-name' has value `Claudia'
Attribute(string): `ns-prefix' has value ``
Attribute(string): `ns-uri' has value `http://schemas.sports.com/players'
Attribute(string): `Parent-QName' has value `players'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====

```

keyword Expressions

The *keyword* expression allows access to values that are declared in the FME mapping file. It is represented in the xFMap document with the <keyword> element, and it has the following general form:

```

<keyword expr="..." default="..." keyword-group="..." op="compare|contains">
  <arg> <!-- keyword --> </arg>
  <arg> <!-- optional default value --> </arg>
</keyword>

```

The *expression string* (i.e., the **expr** attribute) denotes the name of the keyword, which would have been specified in the FME mapping file through the **XFMAP_KEYWORD** keyword, or via a file through the **XFMAP_KEYWORD_FILE** keyword. The optional **default** attribute specifies the default value in the case the specified keyword was not defined.

The **expr** attribute may be an empty string, in which case, the first argument, the first <arg> element, must be present. The evaluated expression sequence for this first argument becomes the keyword to search.

If the **default** attribute is absent or if it is the empty string, then the second argument if it is present will be evaluated to become the default value in the case the keyword was not defined.

The keyword-group attribute is only applicable if keywords were specified via a file through the **XFMAP_KEYWORD_FILE**. Each keyword in a file may be optionally partitioned into groups. The value for this attribute indicates the group name.

The **op** attribute indicates the way in which a keyword is retrieved. The attribute defaults to **compare**, which retrieves a keyword value if a keyword with the exact name is found. If the **op** attribute is set to **contains** then a value is retrieved when the specified name is contained in any of the stored keywords.

Example

Assuming the FME mapping file has the following XML **XFMAP_KEYWORD**'s defined:

```

XML_XFMAP_KEYWORD key0 value0
XML_XFMAP_KEYWORD key1 "my other value"

```

keywords.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<items>
  <item>
    <primary-id>key0</primary-id>
    <alternate-id>he5390</alternate-id>
  </item>
  <item>
    <primary-id>key1</primary-id>
    <alternate-id>a2046</alternate-id>
  </item>
  <item>
    <primary-id>key566</primary-id>
    <alternate-id>ad249</alternate-id>
  </item>
</items>

```

keywords.xmp

```

<?xml version="1.0" encoding="UTF-8"?>
<xfMap>
  <feature-map>
    <mapping match="item">
      <feature-type>
        <keyword expr="">
          <arg> <extract expr="./primary-id"/> </arg>
          <arg> <extract expr="./alternate-id"/> </arg>
        </keyword>
      </feature-type>
    </mapping>
  </feature-map>
</xfMap>

```

FME features constructed:

```

+++++
Feature Type: `value0'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
xml-feat
+++++
Feature Type: `my other value'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====
xml-feat
+++++
Feature Type: `ad249'
Attribute(string): `xml_type' has value `xml_no_geom'
Geometry Type: Unknown (0)
=====

```

fmefunc Expressions

The *fmefunc* expression allows FME mapping file functions to be called. It is represented in the xfm document with the <fmefunc> element, and it has the following general form:

```

<fmefunc expr="...FME function specification...">
  <arg> <!-- some expression sequence --> </arg>
  <arg> <!-- some expression sequence --> </arg>
  ...
  <arg> <!-- some expression sequence --> </arg>
</selexpr>

```

The *expression string* (i.e., the `expr` attribute) specifies an FME function. FME functions are named with an “at” sign, @, as their first character. Both type of FME functions, i.e., feature and attribute functions, may be specified.

If the string in the `expr` attribute does not start with an @ sign, then its literal value is returned, otherwise the FME function is evaluated for its return value, bare in mind that FME feature functions do not return a value so an empty string is returned if this is the case.

The *fmefunc* expression may have 0 or more optional arguments that are expression sequences. Each expression sequence in an `<arg>` element should evaluate to an FME function specification, meaning that its first character must be the @, otherwise, the evaluated expression is simply concatenated as part of the final result for the *fmefunc* expression.

The evaluated value for an *fmefunc* expression is the concatenation of the results for all specified FME function specifications.

Each *fmefunc* expression works upon its own “scratch” FME feature allowing the possibility for algorithmic computations because the state of this temporary feature is not reset between the evaluation of the expression’s arguments and each argument is evaluated in order. Algorithmic calculations are achieved by “stacking up” a series of feature and or attribute FME functions.

Example

items.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<items>
  <item>
    <primary-id>key0</primary-id>
    <alternate-id>he5390</alternate-id>
    <geoLat>312129.20N</geoLat>
    <geoLong>0854453.20W</geoLong>
  </item>
</items>
```

items.xmp

```
<?xml version="1.0" encoding="UTF-8"?>
<xfMap>
<feature-map>
  <mapping match="item">
    <feature-type><literal expr="item"/></feature-type>
    <attributes>
      <attribute>
        <name><literal expr="geoLat"/></name>
        <value><extract expr="./geoLat"/></value>
      </attribute>
      <attribute>
        <name><literal expr="geoLong"/></name>
        <value><extract expr="./geoLong"/></value>
      </attribute>
    </attributes>

    <geometry activate="xml-point">
      <data name="data-string">
        <fmefunc expr="">
          <note> @SupplyAttributes(geoLat, LAT, geoLong, LON) </note>
          <arg>
            <literal expr="@SupplyAttributes("/>
            <literal expr="geoLat,"/>
            <extract expr="./geoLat"/>
            <literal expr=",geoLong,"/>

            <extract expr="./geoLong"/>
            <literal expr=")"/>
          </arg>
        </arg>
      </data>
    </geometry>
  </mapping>
</feature-map>
</xfMap>
```

```

<literal expr="@Angle(ATTRIBUTES,DDDMMSS.SSO,DECIMAL_DEGREES,geoLat,geoLong)"/>
  </arg>
  <note> x,y </note>
  <arg><literal expr="@value(geoLong)"/></arg>
  <arg><literal expr=","/></arg>
  <arg><literal expr="@value(geoLat)"/></arg>
</fmefunc>
</data>
</geometry>
</mapping>
</feature-map>
</xfMap>

```

FME features constructed:

```

+++++
Feature Type: `item'
Attribute(string): `fme_geometry' has value `fme_point'
Attribute(string): `geoLat' has value `312129.20N'
Attribute(string): `geoLong' has value `0854453.20W'
Attribute(string): `xml_type' has value `xml_point'
Geometry Type: Point (1)
Number of Coordinates: 1 -- Coordinate Dimension: 2 -- Coordinate System: `
(-85.74810833333333,31.35811111111111)
=====
=====

```

logexpr Expressions

The *logexpr* expression allows for easy logging. While it can also evaluate to a string (usually the message being logged), its primary purpose is to generate a log entry. It is represented in the xfMap document with the <logexpr> element, and it has the following general form:

```

<logexpr expr='the expression to be logged'
  return='the expression to return'
  severity='[inform|warn|error]'
  limit='maximum number of log messages'
  suppress-limit-warning='[true|false]''>
  <arg> <!-- The expression to be logged --> </arg>
  <arg> <!-- some expression to return --> </arg>
</logexpr>

```

The *expression string* (i.e., the *expr* attribute) specifies a string to enter in the log. The *return* attribute specifies a value to evaluate to, and the *severity* attribute determines the type of log message entered into the log. All these attributes are optional, and both *expr* and *return* attributes can be specified by the two arguments. If any argument is specified, then both *expr* and *return* will be ignored. It is not possible to mix-and-match these attributes and arguments.

If no return value is specified (in either argument or attribute), then the expression evaluates to the message entered in the log. In cases where the *logexpr* is being used solely in order to enter a log message, the return attribute or argument must be specified to be the empty string.

The *severity* attribute is optional, and defaults to 'inform' if it is not specified.

The *limit* attribute is optional. This attribute determines the maximum number of log messages to display. After this number of messages have been logged, additional log messages resulting from the *logexpr* expression are suppressed. Notification of this event is provided in the form of a single log message stating that further log messages will be suppressed. If the *limit* attribute is left unspecified, then no limit is set.

If the *suppress-limit-warning* attribute is specified and set to true, then no notification will be provided when the *logexpr* expression has logged its maximum number of log messages as provided in the *limit* attribute.

Example

items.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<items>
  <item1>
    <key>key 1</key>
  </item1>
  <item2>
    <key>2</key>
  </item2>
  <item3>
    <key>5.002</key>
  </item3>
</items>

```

items.xmp

```

<?xml version="1.0" encoding="UTF-8"?>
<xfmap>
  <feature-map>
    <mapping match="items">
      <feature-type><literal expr="keys"/></feature-type>
    </mapping>
  </feature-map>

  <feature-content-map>
    <mapping match="item1">
      <attributes>
        <attribute>
          <!-- Here we log 'found item 1' but evaluates to the empty string.
               This shows its use as a 'side effect' of logging, rather than
               for the string value it will evaluate to -->
          <name><literal expr="item-1"/><logexpr expr="found item 1" return=""
severity="inform"/></name>
          <value><extract expr="./key"/></value>
        </attribute>
      </attributes>
    </mapping>

    <mapping match="item2">
      <attributes>
        <attribute>
          <!-- Notice that the log message becomes part of the name -->
          <name><literal expr="item"/><logexpr expr="two found"
severity="warn"/></name>
          <value><extract expr="./key"/></value>
        </attribute>
      </attributes>
    </mapping>

    <mapping match="item3">
      <attributes>
        <attribute>
          <name><literal expr="item3"/></name>
          <value><extract expr="./key"/>
          <!-- notice that the logexpr expr="three found" is completely ignored here -->
          <logexpr expr="three found" severity="inform">
            <arg><literal expr="item 3's key was found with value
"/><extract expr="./key"/></arg>
            <arg><literal expr=" : key5 expected"/></arg>
          </logexpr>
        </value>
      </attribute>
    </attributes>
  </feature-content-map>
</xfmap>

```

Messages Logged

```
=====  
...|INFORM|found item 1  
...|WARN  |two found  
...|INFORM|item 3's key was found with value 5.002  
=====
```

Features Constructed

```
+++++  
...|INFORM|Feature Type: `keys'  
...|INFORM|Attribute(string): `item-1found item 1' has value `key 1'  
...|INFORM|Attribute(string): `item3' has value `5.002'  
...|INFORM|Attribute(string): `itemtwo found' has value `2'  
...|INFORM|Attribute(string): `xml_type' has value `xml_no_geom'  
...|INFORM|Geometry Type: Unknown (0)  
=====
```

comparison Expressions

The *comparison* expression implements a simple conditional choice for expression sequences. It has the following general form:

```
<comparison [expr=["|=|!="]] lhs="..." rhs="..." success="..." failure="...">  
  <arg name="lhs">...</arg>  
  <arg name="rhs">...</arg>  
  <arg name="success">...</arg>  
  <arg name="failure">...</arg>  
</comparison>
```

Notice that unlike other expressions, the arguments are identified by name rather than by position. The names are not optional, and the order of the arguments does not matter. All arguments/attributes are optional. The *expr* attribute defaults to the string '=', but all other attributes and arguments default to the empty string "".

The meaning of the *comparison* expression is to perform a simple conditional choice. The arguments/attributes 'lhs' and 'rhs' are abbreviations for 'left-hand side' and 'right-hand side'. These two expressions are evaluated and compared. The comparison type is determined by the 'expr' attribute. If the *expr* attribute is '=' and the comparison is true (that is, if the left hand side is equal to the right hand side), then the *success* argument is evaluated and returned, otherwise the *failure* argument is evaluated and returned. If the *expr* attribute is '!=', for 'not equal', then the sense of the comparison is reversed; i.e. if the comparison is true, then *failure* is evaluated, otherwise *success* is evaluated.

Note: This is not a *flow-of-control* conditional expression, but rather simply chooses between expression arguments to evaluate.

The example below demonstrates a case in which news articles are tagged with their source language, 'fr' for 'French' and 'en' for English. Suppose that we want to extract all the author's names, one per feature. But since the french word for 'name' is 'nom' (and 'histoire' for 'story'), we want to selectively extract the author's name based on the stories source language. We do this by assigning an attribute to a feature, and setting that attribute's value as a comparison expression that chooses based on the 'lang' attribute.

Example

stories.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<entries>  
  <entry lang="en">  
    <name>Mark</name>  
    <story> ... </story>  
  </entry>  
  <entry lang="fr">  
    <nom>Jean-Sebastian</nom>  
    <histoire> ... </histoire>
```

```

</entry>
</entries>

```

stories.xmp

```

<?xml version="1.0" encoding="UTF-8"?>
<xfmap>
  <feature-map>
    <mapping match="entry">
      <feature-type><literal expr="author"/></feature-type>
      <attributes>
        <attribute>
          <name><literal expr="author"/></name>
          <!-- Here we check if the 'lang' attribute is equal to the string 'Name' and if
it is, we extract the Name element. Otherwise, we extract the Nom element -->
          <value>
            <comparison lhs="en">
              <arg name="rhs"><extract expr="@lang"/></arg>
              <arg name="success"><extract expr="./Name"/></arg>
              <arg name="failure"><extract expr="./Nom"/></arg>
            </comparison>
          </value>
        </attribute>
      </attributes>
    </mapping>
  </feature-map>
</xfmap>

```

```

=====
          Features Constructed
=====
| INFORM | ++++++
| INFORM | Feature Type: `entry'
| INFORM | Attribute(string): `author' has value `Mark'
| INFORM | Attribute(string): `xml_type' has value `xml_no_geom'
| INFORM | Geometry Type: Unknown (0)
| INFORM | =====
| INFORM | ++++++
| INFORM | Feature Type: `entry'
| INFORM | Attribute(string): `author' has value `Jean-Sebastian'
| INFORM | Attribute(string): `xml_type' has value `xml_no_geom'
| INFORM | Geometry Type: Unknown (0)
| INFORM | =====

```

FME Schema Features

Since the XML Reader is an FME plug-in reader, it must be able to return FME schema features, either to FME or to a third-party application through FME Objects. An xfMap document may contain an optional <schema-type> element that instructs the XML Reader on how the FME schema features should be constructed. On default, if the <schema-type> element is not present, the schema features are constructed by scanning all of the FME features returned by the XML Reader. Currently, the XML Reader can construct FME schema features by three different methods which are specified through the <schema-type> element's child element.

If the <schema-type> element is present, then it must contain one of the following child elements:

1. **<scan> element** - this is the XML Reader's default method for constructing schema features. If the <schema-type> element is not present in an xfMap document, then this method is assumed.
2. **<generate> element** - this element contains two attributes, the xfMap and document attributes, that specify the XML document with the schema feature information and the xfMap document which maps that information into FME schema features.
3. **<inline> element** - this element can contain zero or more <schema-feature> child elements. A <schema-feature> element represents an FME schema feature explicitly; it contains a type attribute that specifies a schema

feature's feature type, and zero or more <schema-attribute> elements that specify the schema feature's attributes and attribute types.

The following sections describe each element.

Scanning For FME Schema Features

The XML Reader defaults to scanning the FME features constructed through the xfMap when it is requested for schema features. This can be explicitly stated in the xfMap by including a <scan> child element in the optional <schema-type> element.

The following xfMap sample explicitly instructs the XML Reader to scan the FME features it constructs for the generation of the schema features (however, it is not necessary, since this is the default mode).

```
<?xml version="1.0"?>
<!DOCTYPE xfMap SYSTEM "xfMap.dtd">
<xfMap>

    <schema-type>
        <scan/>
    </schema-type>
    ...
</xfMap>
```

If possible, the <generate> or <inline> element should be used instead. This avoids the double read the XML Reader would perform on an XML input dataset when the FME uses it for translation: once for returning schema features, and once for returning the actual FME data features.

Generating FME Schema Features

The XML Reader can switch the given xfMap and XML input document when it is requested for schema features. This can be done through the <schema-type> element's <generate> element.

The <generate> element contains two required attributes: the xfMap and the document attributes. They specify the xfMap and XML document to used when XML Reader is requested for schema features. In other words, it is possible for the XML Reader to ignore the given xfMap and input XML document by using the <generate> element.

Consider the following input XML document. (It is the same *points1.xml* document used earlier and reproduced here for convenience.)

points1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<points>
  <point name="myPoint" num="0">
    <color>
      <red>0.324</red>
      <green>0.233</green>
      <blue>0.596</blue>
    </color>
    <location x="10.0" y="0.0"/>
  </point>
  <point name="myPoint" num="1">
    <color>
      <red>0.874</red>
      <green>0.948</green>
      <blue>0.554</blue>
    </color>
    <location x="5.0" y="5.0"/>
  </point>
</points>
```

The following xfMap document, *generate_points1.xmp*, maps the elements in the *points1.xml* document into FME features:

generate_points1.xmp

```

<?xml version="1.0"?>
<xfMap>
  <note>
    This xfMap document maps elements from the points1.xml document.
  </note>

  <schema-type>
    <generate xfMap="generate_points1_schemas.xmp"
      document="generate_points1_schemas.xml"/>
  </schema-type>

  <feature-map>
    <mapping match="point">
      <feature-type> <literal expr="point"/> </feature-type>
      <attributes>
        <attribute>
          <name> <literal expr="number"/> </name>
          <value> <extract expr="@num"/> </value>
        </attribute>
        <attribute>
          <name> <literal expr="color"/> </name>
          <value>
            <extract expr="./color/red"/>
            <literal expr=","/>
            <extract expr="./color/green"/>
            <literal expr=","/>
            <extract expr="./color/blue"/>
          </value>
        </attribute>
      </attributes>

      <geometry activate="xml-point">
        <data name="data-string">
          <extract expr="./location[@x]"/>
          <literal expr=","/>
          <extract expr="./location[@y]"/>
        </data>
      </geometry>
    </mapping>
  </feature-map>
</xfMap>

```

When the *points1.xml* and *generate_points1.xmp* are fed into the XML Reader and FME data features are requested, then the following features are output:

```

+++++
Feature Type: `point'
Attribute(string): `color' has value `0.324,0.233,0.596'
Attribute(string): `fme_geometry' has value `fme_point'
Attribute(string): `number' has value `0'
Attribute(string): `xml_type' has value `xml_point'
Geometry Type: Point (1)
Number of Coordinates: 1 -- Coordinate Dimension: 2 -- Coordinate System: ``
(10,0)
=====
+++++
Feature Type: `point'
Attribute(string): `color' has value `0.874,0.948,0.554'
Attribute(string): `fme_geometry' has value `fme_point'
Attribute(string): `number' has value `1'
Attribute(string): `xml_type' has value `xml_point'
Geometry Type: Point (1)
Number of Coordinates: 1 -- Coordinate Dimension: 2 -- Coordinate System: ``
(5,5)
=====

```

Notice that the *generate_points1.xmp* xfMap contains a <schema-type> element. Its child element is the <generate> element whose attributes, xfMap and document attributes, indicate the xfMap (*generate_points1_schemas.xmp*) and XML document (*generate_points1_schemas.xml*) to be used when the XML Reader is requested for schema features.

The *generate_points1_schemas.xml* document contains one <schema-feature> element; this element describes the FME schema feature that we want for the FME data features that are constructed from the *points1.xml* document through the *generate_points1.xmp* xfMap.

generate_points1_schemas.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<schemas>
  <schema-feature type="point">
    <attribute name="fme_geometry{0}" type="xml_point"/>
    <attribute name="color" type="xml_char(20)"/>
    <attribute name="number" type="xml_int16"/>
    <attribute name="xml_type" type="xml_char(12)"/>
  </schema-feature>
</schemas>
```

The usage of the <generate> element requires the user to have knowledge of how schema features are represented in FME. Every FME schema feature must have an *fme_geometry{}* list attribute that lists the possible geometry types of a schema feature. For the XML Reader, the possible values for the *fme_geometry{}* list attribute are: *xml_no_geom*, *xml_text*, *xml_point*, *xml_line*, *xml_area*, and *xml_text*. If a theme or feature type is capable of containing several geometries (for example, points, lines and areas), then the *fme_geometry{}* list attribute should contain 3 members:

```
fme_geometry{0} = "xml_point"
fme_geometry{1} = "xml_line"
fme_geometry{2} = "xml_area"
```

The type of user attribute, that is, the value of an attribute, must be one of the predefined FME types: *xml_char(#)*, *xml_int16*, *xml_int32*, *xml_real32*, *xml_real64*, *xml_decimal(##)*, or *xml_boolean*.

The following xfMap *generate_points1_schemas.xmp* maps the <schema-feature> element into an FME feature.

```
<?xml version="1.0"?>
<xfMap>
  <note>
    This xfMap document maps elements from the
    generate_points1_schema.xml document.
  </note>
  <feature-map>
    <mapping match="schema-feature">
      <feature-type> <extract expr="@type"/> </feature-type>
    </mapping>
  </feature-map>
  <feature-content-map>
    <mapping match="attribute">
      <attributes>
        <attribute type="sequenced">
          <name> <extract expr="@name"/> </name>
          <value> <extract expr="@type"/> </value>
        </attribute>
      </attributes>
    </mapping>
  </feature-content-map>
</xfMap>
```

Note: Nothing dictates that the xfMap *generate_points1_schemas.xmp* and the *generate_points1_schemas.xml* document should take the form of the example above. What is required when the <generate> element is used is that an FME feature constructed with the XML document (specified in the document attribute) with the xfMap (specified in the xfMap attribute) by the XML Reader conforms to the definition of an FME schema feature. The FME

schema features constructed must contain a `fme_geometry{ }` list attribute, and the attributes defined for the features must have values that equal one of the FME types: `xml_char(#)`, `xml_int16`, `xml_int32`, `xml_real32`, `xml_real64`, `xml_decimal(#, #)`, or `xml_boolean`.

The document attribute in the `<generate>` element may take the special value, `#dataset`, which instructs the XML Reader to keep given input XML document as the document used for generating the FME schema features.

Example:

```
<schema-type>
  <generate xfMap="schemas.xmp" document="#dataset"/>
</schema-type>
```

Inline FME Schema Features

The `xfMap` has a mechanism to specify schema features in its content. This is done through the `<schema-type>` element's `<inline>` element.

The `<inline>` element can contain zero or more `<schema-feature>` child elements. A `<schema-feature>` element represents an FME schema feature explicitly; it contains a `type` attribute that specifies a schema feature's feature type, and zero or more `<schema-attribute>` elements that specify the schema feature's attributes and attribute types.

The `<schema-attribute>` element has two required attributes, they are the name and type attributes. The value for the type attribute must be an FME attribute type: `xml_char(#)`, `xml_int16`, `xml_int32`, `xml_real32`, `xml_real64`, `xml_decimal(#, #)`, or `xml_boolean`.

The following example illustrates the usage of the `<inline>` element. When the XML Reader is requested for FME schema features and the `xfMap` below is used, then a schema feature of feature type `states` having 9 user-defined attributes – `AREA`, `CODE`, `NAME`, `POP1990`, `POP90_SQMI`, `P_URBAN90`, `P_ING_LANG`, `P_EMPL_SEC`, and `HSE_UNIT90` – and no geometry is returned.

```
<?xml version="1.0" encoding="UTF-8"?>
<xfMap>
  <schema-type>
    <inline>
      <schema-feature type="states">
        <schema-attribute name="AREA" type="xml_decimal(16,3)"/>
        <schema-attribute name="CODE" type="xml_char(4)"/>
        <schema-attribute name="NAME" type="xml_char(25)"/>
        <schema-attribute name="POP1990" type="xml_decimal(11,0)"/>
        <schema-attribute name="POP90_SQMI" type="xml_decimal(20,6)"/>
        <schema-attribute name="P_URBAN90" type="xml_decimal(20,6)"/>
        <schema-attribute name="P_ING_LANG" type="xml_decimal(20,6)"/>
        <schema-attribute name="P_EMPL_SEC" type="xml_decimal(20,6)"/>
        <schema-attribute name="HSE_UNIT90" type="xml_decimal(11,0)"/>
        <schema-attribute name="fme_geometry{0}" type="xml_no_geom"/>
      </schema-feature>
    </inline>
  </schema-type>
  ...
</xfMap>
```