

FME Feature Store Reader/Writer

This chapter describes how the Feature Manipulation Engine (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` chapter 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 earlier versions of Workbench (build 1378 and earlier).

A spatial index may also be created and saved with the feature store which the FFS reader makes use of 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.

Schema information may or may not be stored explicitly in FFS files, depending on the options given when it was written.

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	Directory or File
Feature Type	File base name (exceptions are possible)
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	no
elliptical arc	yes	surface	no
ellipses	yes	text	yes
line	yes	z values	yes
none	yes		

Reader Overview

The FFS reader first scans the directory it is given for .ffs files defined in the mapping file or listed in the `IDs` statement. The FFS reader then extracts features from the files one at a time, and passes them on to the rest of the FME for further processing. If the FFS files have associated spatial indexes, then a spatial query can be used to limit the features returned.

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**Required/Optional:** *Required*

The value for this directive is the directory containing the FFS files to be read or a single FFS file. 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.

Example:

```
FFS_DATASET /usr/data/data/92i080.ffe
```

Workbench Parameter: [<WorkbenchParameter>](#)

DEF**Required/Optional:** *Optional*

This specification is used to define FFS files read. The syntax of the `DEF` directive is:

```
<ReaderKeyword>_DEF <baseName>
```

Note that this directive is not used when the dataset was a file.

Example:

The example below defines a `roads` FFS file for input during a translation:

```
FFS_IDS roads
```

Workbench Parameter: [<WorkbenchParameter>](#)

IDs**Required/Optional:** *Optional*

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.

Example:

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: [<WorkbenchParameter>](#)

PASSPHRASE

Required/Optional: *Required only when input is encrypted.*

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. The syntax of the `PASSPHRASE` directive is:

```
<ReaderKeyword>_PASSPHRASE <character string>
```

This must only be used if the FFS files being read were encrypted when they were created.

Workbench Parameter: [<WorkbenchParameter>](#)

SEARCH_ENVELOPE

Required/Optional: *Optional*

This specification is used to restrict the returned features to those that intersect the defined envelope. The syntax of the `SEARCH_ENVELOPE` directive is:

```
<ReaderKeyword>_SEARCH_ENVELOPE <xmin> <ymin> <xmax> <ymax>
```

This can only be used if the FFS files being read were created with spatial indexes.

Workbench Parameter: [<WorkbenchParameter>](#)

SEARCH_ENVELOPE_COORDINATE_SYSTEM

Required/Optional: *Optional*

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.

The syntax of the `SEARCH_ENVELOPE_COORDINATE_SYSTEM` directive is:

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

Workbench Parameter: [Search Envelope Coordinate System](#)

SEARCH_CLOSEST_POINT

Required/Optional: *Optional*

This specification is used to restrict the returned features to the closest feature in each FFS file to some search point. The syntax of the `SEARCH_CLOSEST` directive is:

```
<ReaderKeyword>_SEARCH_CLOSEST_POINT <x> <y> <maxdist>
```

This 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 be-

fore it will be returned. Distances are calculated to the boundaries of area-based features.

Workbench Parameter: [<WorkbenchParameter>](#)

ENFORCE_SECONDARY_FILE_NAMES

Required/Optional: *Optional*

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 named like spillover files. This directive is used to differentiate between the two situations.

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. The default for this directive is YES.

The syntax of the `ENFORCE_SECONDARY_FILE_NAMES` directive is:

```
<ReaderKeyword>_ENFORCE_SECONDARY_FILE_NAMES YES
```

Writer Overview

The FFS writer dumps the memory representation of each FME feature to a disk file, optionally creating a spatial index for them. It can separate the features written into a separate file for each feature type or it can merge all features into a single FFS file. 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

Required/Optional: *Required*

The value for this directive is the destination directory for the FFS files. A typical mapping file fragment specifying an output FFS dataset looks like:

```
<WriterKeyword>_DATASET /usr/data/ffs/new
```

The dataset may also be an actual FFS file. In such a case, the named file is written and there must not be any `DEF` lines present.

Example:

```
<WriterKeyword>_DATASET /usr/data/data/new.ffs
```

Workbench Parameter: [<WorkbenchParameter>](#)

DEF

Required/Optional: *Optional*

When used to write multiple files, the FFS writer uses `_DEF` lines to define files to which features will be written. The definition lists only the feature type of those features to be written to the file.

Example:

This fragment defines an output file for all features whose feature type was `roads`:

```
<WriterKeyword>_DEF roads
```

Workbench Parameter: [<WorkbenchParameter>](#)

INDEXED

Required/Optional: *Optional*

This flag indicates whether or not a spatial index should be created and saved with each output FFS file. The spatial index has the same base name as the FFS file, but will have a `.fsi` extension. Spatial indexes are needed if the FFS files are later used as the source for spatial queries by the FFS reader.

Example:

This fragment specifies that a spatial index should be created:

```
<WriterKeyword>_INDEXED yes
```

Workbench Parameter: [<WorkbenchParameter>](#)

STRICT_SCHEMA

Required/Optional: *Optional*

This flag indicates whether or not features should have all user attributes not listed on the `DEF` line removed before they are saved. Valid values are `yes` and `no`. If `yes`, the unlisted attributes are stripped, forcing the features to strictly conform to the schema specified on the `DEF` line. The default is `no`.

Strictly adhering to a schema may greatly reduce file size in some cases by removing unnecessary attributes.

Example:

This fragment specifies that all features strictly adhere to the defined schema:

```
<WriterKeyword>_STRICT_SCHEMA yes
```

Workbench Parameter: [<WorkbenchParameter>](#)

MAX_FILE_SIZE

Required/Optional: *Optional*

This directive 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. Normally this is not used.

Example:

```
<WriterKeyword>_MAX_FILE_SIZE 1000000
```

[Workbench Parameter: <WorkbenchParameter>](#)

PASSPHRASE

Required/Optional: *Optional*

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. The syntax of the `PASSPHRASE` directive is:

Example:

```
<WriterKeyword>_PASSPHRASE <character string>
```

If this is not used when writing the output dataset, it is not necessary to specify it when reading it in again.

[Workbench Parameter: <WorkbenchParameter>](#)

COMPRESSION

Required/Optional: *Optional*

The user may specify a `COMPRESSION_LEVEL` value between 0 and 9. A lower compression level will result in faster operation for both reading and writing while a higher compression level will result in smaller file sizes.

Default Value: 6

Example:

```
<WriterKeyword>_COMPRESSION_LEVEL 0
```

[Workbench Parameter: <WorkbenchParameter>](#)

BYTE_ORDER

Required/Optional: *Optional*

The directive `BYTE_ORDER` indicates if 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.) 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.

Default Value: `NATIVE`

Example:

```
<WriterKeyword>_BYTE_ORDER LITTLE_ENDIAN
```

[Workbench Parameter: <WorkbenchParameter>](#)

SCAN_FOR_SCHEMA

Required/Optional: *Optional*

The directive `SCAN_FOR_SCHEMA` indicates the `DEF` lines should be used to determine the schema or if the schema should be scanned directly from the input. Scanning will ignore schema definitions on the `DEF` lines and instead write out the schema definitions gleaned from scanning all features output to that writer. The valid values are `YES` and `NO`.

Default Value: *NO*

Example:

```
<WriterKeyword>_SCAN_FOR_SCHEMA NO
```

OUTPUT_NO_SCHEMA

Required/Optional: *Optional*

This is used to indicate whether or not schema information should be stored in the output file. Often schema information is needed from FFS files when they are later used (such as when generating a workspace), and the user may want to decide if the FFS files they create already have that schema information handy or not. If the schema information is stored in the FFS file, it will use this without the cost of reading in and scanning the entire file for a schema. The user may, however, desire to have no schema information present in the FFS file (forcing a schema scan later) if there is no correct schema information handy at the time of writing, or if scanning when reading would otherwise provide a more desirable schema. The valid values are `YES` and `NO`. `YES` will not output any schema information into the FFS file.

Default Value: *NO*

Example:

```
<WriterKeyword>_OUTPUT_NO_SCHEMA YES
```

[Workbench Parameter: <WorkbenchParameter>](#)

Feature Representation

In addition to the generic FME feature attributes that FME Workbench adds to all features (see *About Feature Attributes* on page 7), 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.

The FFS writer does employ one special attribute. If it is being used to write multiple FFS files, and a feature being output has an attribute called `ffs_feature_type`, then just before the feature is output into the FFS, the value for the attribute is used as the feature type and this attribute is removed from the feature. This is used to create fea-

ture stores where the FFS file name is not the same as the feature type of the features it holds.

