

# Welcome to the FME Quick Translator

---

Designed for quick translation, the FME® Quick Translator makes it easy for you to perform fast, easy conversions between hundreds of formats.

## Getting started

- [FME Workbench or FME Quick Translator?](#)

## Resources

- [FME Overview](#)
- [FMEpedia: Community Answers and Knowledge Base](#)

## FME Desktop Help

This help file contains information on performing translations and transformations using the FME Quick Translator.

For other FME Components such as FME Workbench and FME Data Inspector, start the component and click the Help menu.

Coordinate System Help: Look under *Coordinate Systems* in this Help file, or press the F1 key from the Coordinate System Gallery.

*Especially for advanced operations, you may have to look in different help files to get the information you need, or search FMEpedia for solutions.*



## **FME Components**

### **Primary Components**

FME is actually a number of different software applications that comprise a number of spatial data handling components. All applications listed here are included with every edition of FME. Browse your FME installation directory to access the components.

#### ***FME Workbench***

FME Workbench has an intuitive point-and-click graphic interface to enable translations to be graphically described as a flow of data. FME Workbench is the primary tool for data translations in FME. It is quite feasible that you will only ever use Workbench and the FME Universal Viewer.

#### ***FME Universal Viewer***

The FME Universal Viewer allows quick viewing of data in any FME-supported format. It is used primarily for data validation and quality assurance by allowing you to preview data before translation, or review data after translation.

#### ***FME Data Inspector (FME technology preview)***

The FME Data Inspector is a Safe Software technology preview, intended to eventually replace the FME Universal Viewer. Although it maintains many of the same features as the FME Universal Viewer, the Data Inspector is cross-platform, it uses the latest display technology, and it supports 3D viewing.

For FME 2011, users will have the option of using both.

#### ***FME Quick Translator/FME Universal Translator***

The FME Quick Translator is the redesigned FME Universal Translator. The component has been retooled to highlight FME's quick translation capability.

#### ***FME Command-Line Engine***

The FME Command-Line Engine enables translation requests to be submitted at the command-line level.

### **Developer Components**

These additional development components are all included as part of the standard FME package.

#### ***FME Objects***

FME Objects is a software library for working with spatial data. Application developers use FME Objects to add spatial data reading and writing support into their stand-alone applications.

#### ***FME Application Extenders***

FME Application Extenders are components by which FME technology is used or embedded into other GIS applications. These are then known as FME Enabled Applications. Application Extenders enable a GIS product to view datasets not native to that application.

#### ***FME Plug-In SDK***

The FME Plug-In SDK allows developers to add their own formats or functionality to the FME translation core.

#### ***FME Integration SDK***

The FME Integration SDK allows developers to create FME-enabled applications for users who already have FME installed on the same PC.

## Getting Started

### Looking for more information?

These help files contain information on performing translations using the FME Quick Translator.

FME Quick Translator, FME Workbench, FME Universal Viewer, and FME Data Inspector all have separate help files.

However, because there is some overlap for advanced functionality in FME Workbench and FME Quick Translator, you may have to look in separate help files or search our FMEpedia site to get the information you need.

### **Help File Locations**

Your FME installation directory contains a help folder, which contains all the help files associated with FME. You can also find these files, including updates, on our website.

### **Documentation**

To view all documentation, visit the [Documentation page](#) on FMEpedia.

[FME Fundamentals Reference](#): This manual contains the fundamental operating and configuration principles of FME.

[FME Functions and Factories Reference](#): This is a detailed guide to the various functions and factories supported in FME. This is an advanced developer resource.

### **FMEpedia**

[www.FMEpedia.com](http://www.FMEpedia.com), an information commons provided by Safe Software Inc. for the FME User Community.

### **FME Workbench or Quick Translator?**

For simple translations that do not involve any customization, you can get quick results by using the *FME Quick Translator*.

Advanced FME users can execute custom mapping files through this interface.

If you want to add customizations or detailed transformations, including those that require multiple input and output formats, *FME Workbench* is the best tool. *FME Workbench* is also the easiest way to apply the processing facilities of FME to your data as it is translated.

### **Quick Translation Steps**

Follow these steps to perform a quick automated translation:

1. Drag a source file and drop it in the FME window.
2. Select your destination format.
3. Browse to a destination folder and enter a filename.
4. Click OK.

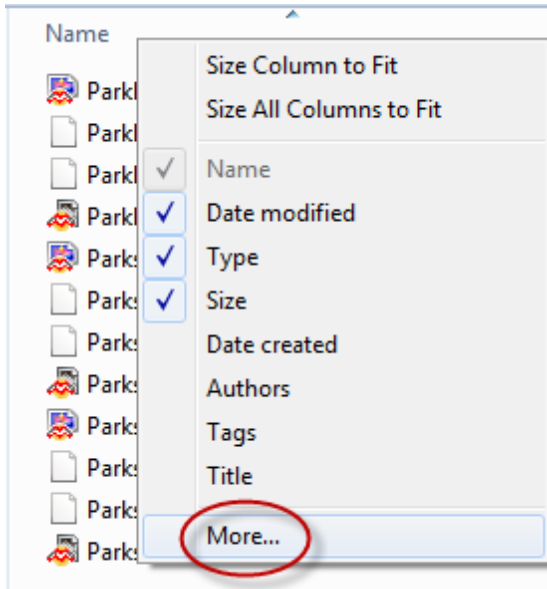
FME will start the translation and the output will be saved to your destination folder.

## File Management

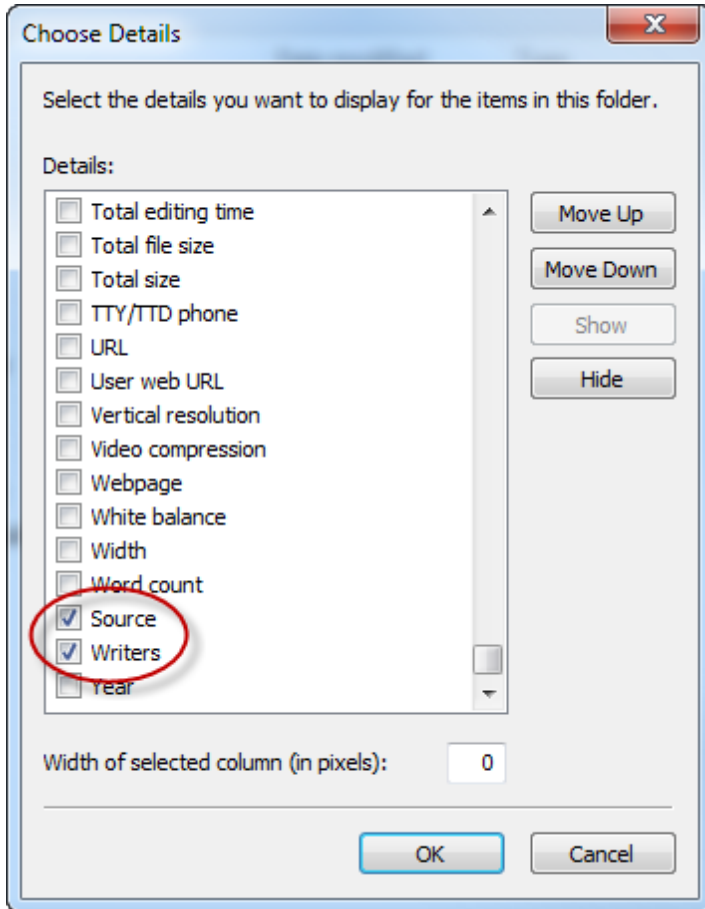
Through Windows Explorer, you can view summary information about FME-generated files without having to start FME and open the file. For mapping files, this includes the Reader, Writer, and a number of Functions and Factories. Float the cursor over a filename to see the information in a pop-up window.

You can also view the source and destination format in two additional columns in Windows Explorer:

1. Open Windows Explorer and go to the directory that contains your FME files.
2. Right-click an existing column heading to display the command menu and select **More**.



3. Scroll down and check the boxes labeled **Source** and **Writers**.



4. Click **OK** to display the columns in Windows Explorer.

**Note:** You will have to repeat these steps for each directory that you want to display the Source and Writers columns.



## Setting the Temporary Directory

When FME runs a large, multi-dataset translation, it often requires a lot of temporary disk space. This is particularly true when running a Dataset Fanout, because there is no guarantee that the features will arrive at the fanout in a single dataset group. Therefore, FME has to write out all of the datasets to temporary storage, and then fan them out afterwards. So the amount of available disk space is important, but on a performance issue you might be more concerned about the speed of all this disk activity.

Many of the FME temporary files are created when caching data for larger datasets or for display in FME Universal Viewer or Data Inspector. Using a faster hard drive can make a significant difference to the FME translation if disk cache I/O speeds are improved. An example would be if you have an SSD (Solid-State Drive) on your computer. These are typically quite a bit faster than traditional HDD drives, so if you can point your `FME_TEMP` to an SSD then you'll see a bit of a performance increase when working with larger datasets.

Where possible, set your temporary directory to point to the fastest disk you have available.

*The **FMEpedia** article setting a different temporary directory **tells you how to set the `FME_TEMP` environment variable.** (In Windows 7, look under Control Panel > System > Advanced > Environment Variables).*

## Usage Notes

Don't set your temporary directory on the same disk that the operating system uses; FME might be slowed down by the operating system writing to the same disk at the same time.

Try to set the temporary directory to a disk that has a large amount of free space – it won't improve the speed, but it might prevent a large translation from failing due to a lack of disk space.

## Java VM Loading

***Note: These settings are for FME Plug-ins written in Java.***

These environment variables allow you to specify memory available to Java Plugins:

- `FME_JVM_MIN_HEAP_SIZE`: customize the initial heap size for initializing Java VM.
- `FME_JVM_MAX_HEAP_SIZE`: customize the maximum heap size for initializing Java VM.

By default, the initial heap size is 1024K and the maximum is 16384K. JVM will take the default values if these two variables are not set. The values of these two variables must be multiples of 1024K (for example, 4M and 64M or 4096K and 32768K).

## Shared Resources

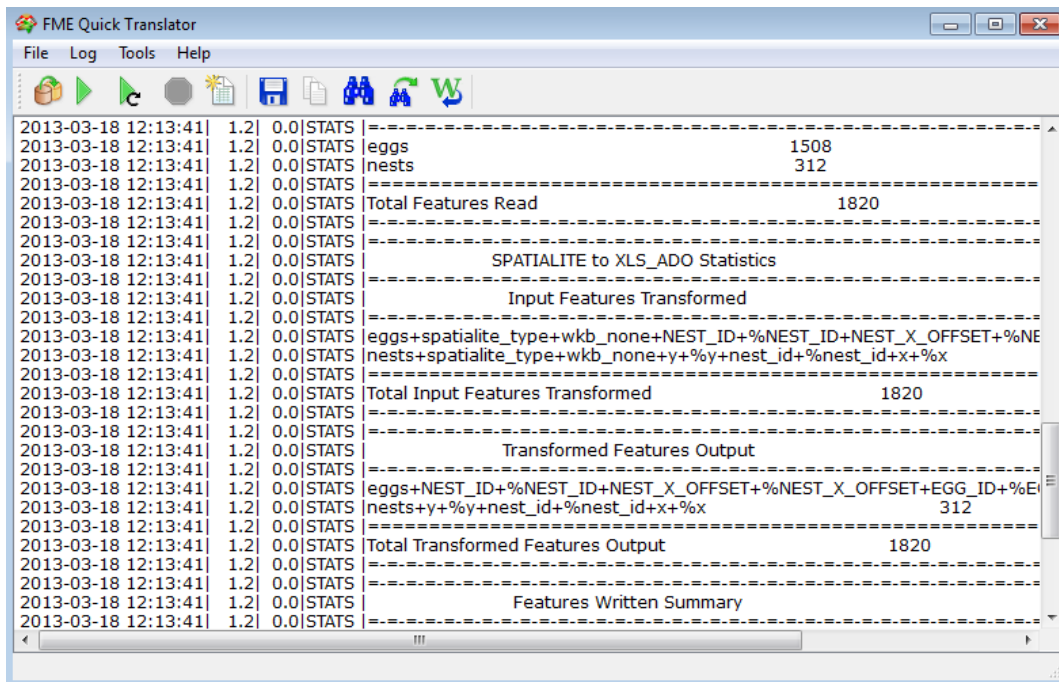
A shared resource is any FME file that has been made available for other users to use. These files are:

- [Workspaces](#) (\*.fmw)
- [Mapping Files](#) (\*.fme)
- [Custom Transformers](#) (\*.fmx)
- [Custom Formats](#) (\*.fds)
- [Custom Coordinate Systems](#) (anything really, though \*.fme is best)
- [Transformer Categories](#) (\*.fmxmlist)

Using **Tools > FME Options > Default Paths**, an FME user can define specific locations where shared resources can be found. After doing so resources in these directories will become automatically available to that user; that is, custom formats appear in the formats gallery, custom transformers in the transformer gallery, and so forth.

## User Interface

FME Quick Translator has an easy-to-use interface. This example shows the log window after a quick format-to-format translation.



## Menus

### File Menu

**Translate:** Display the Set Translation Parameters dialog.

**Run/Rerun:** Run a custom mapping file that is not in the registry, or rerun the last file using the same parameters.

**Stop:** Stop a translation.

**Exit:** Shut down FME.

### Log Menu

**Copy:** Copy text from the log view, select all text in the window, or save the contents to a text file. See page 16.

**Select All:** Select the text in the log window.

**Save As:** Save the log file to a text file.

**Find/Find Next:** Search for text in the log.

**Word Wrap:** Allows you to wrap words to the next line at the vertical edge of the log window.

### Tools Menu

**Generate:** Generate a mapping file for further editing.

**Create Batch:** Save a batch script of a translation to run later, or on another platform.

**Save Temporary Files:** During a translation, the FME GUI stores temporary files in the default temporary Windows directory (usually C:\temp). Usually these files are removed when FME is shut down. However, you can choose to keep temporary GUI files for information purposes by selecting Save Temporary Files from the Settings menu. Note that temporary FME translation files may be stored in a different directory as defined by the FME\_TEMP environment variable.

**Purge Temporary Files:** Purges any temporary files that FME may have created during translations. Note, however, that this command removes all FME temporary files from C:\TEMP as well as the directory referenced by FME\_TEMP. See [Temporary Directory Determination](#).

**Browse Coordinate Systems:** Display the Coordinate System Gallery.

**Browse Readers and Writers:** View a gallery of FME-supported formats.

**FME Options:** [Set options for FME components](#).

### Help Menu

**FME Quick Translator Help:** This help system.

**FME Readers and Writers Reference:** Displays a technical document containing information on all FME-supported formats.

**Contact Safe Support:** Displays Safe Software's *Technical Support Request* web page.

**FMEpedia Knowledge Base:** Access the FMEpedia knowledge base.

**Training Resources:** Find out about FME training options.

**Check for Updates:** Checks for a newer version of the FME software, and provides a link to click if a newer version is available.

**About FME Quick Translator:** Displays information about the edition and version of FME software installed on your computer.

## Log Window

The FME log view displays statistics and processing information that includes the following:

- FME version
- reader being used
- writer being used
- logging information
- warning messages

Information messages are displayed until the translation is complete. When the translation is complete, you can search for text, copy selected contents of the window directly to another application, or save the contents to a file.

Change log file preferences from the **Log** menu.

***Note:** The text in this window contains important information on the translation. If you ever get results you did not expect in your output data, check the contents of the log.*

## Drag and Drop Support

FME supports complete drag and drop functionality:

- To run a mapping file, you can select an .fme file from your Windows desktop or from Windows Explorer, and drag it to the log view. FME will automatically run the file.
- Select one or more input files from your Windows desktop or from Windows Explorer, and drag them onto the FME window. The Translate dialog will appear. All you have to do is enter destination information.
- Select a file from your Windows desktop or from Windows Explorer, and drag it onto the **Generate** toolbar button. The Generate Mapping file dialog will appear. All you have to do is enter destination information. See [Creating a Custom Mapping File](#).

## Reader and Writer Gallery

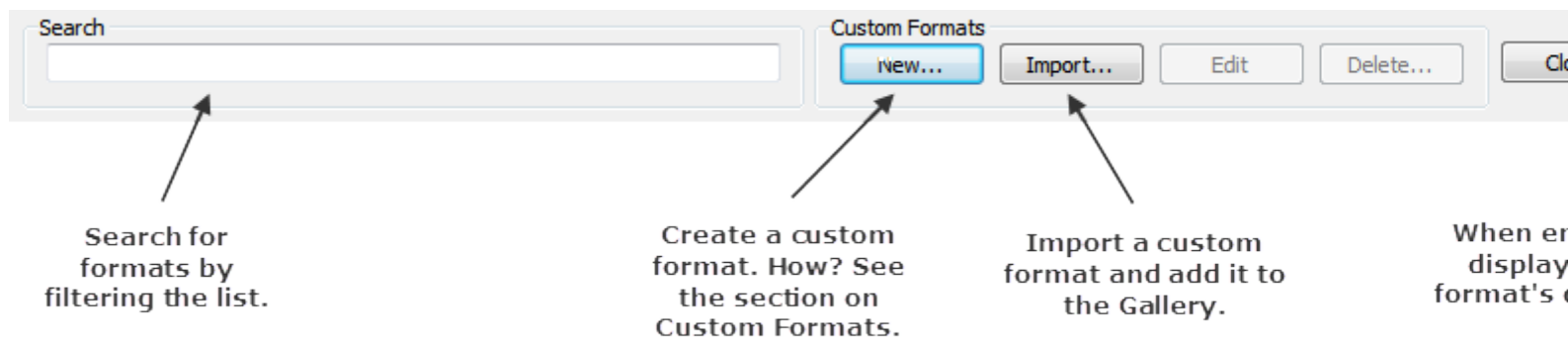
To display the Formats Gallery, from the **Tools** menu select **Browse Readers and Writers**.

The Formats Gallery displays all formats supported by FME. It includes the following information:

- **Description:** Format name.
- **Short Name:** This is the common abbreviation for the format name.
- **Extensions:** File name extensions associated with the formats.
- **Type:** Directory-based or file-based format.
- **Read:** Indicates whether FME reads the format.
- **Write:** Indicates whether FME writes the format.
- **Coordinate System:** Indicates whether a custom coordinate system is associated with the format.
- **Licensed:** Some formats require that you obtain special licensing, or specific versions of FME. This column indicates whether your FME is licensed to read/write the format.

*You can't edit the settings for any of the columns listed above – double clicking on a checkmark will simply select the format and close the window.*

- **Search** and **Custom Formats**



## Setting Format Parameter Defaults

You change the defaults for frequently used settings in most format parameter dialogs. For example, you might want to permanently set a default file, server names and user accounts for databases.

Many parameter boxes have a Defaults button, with the following options:

- Save as My Defaults: Edit a field, then choose this option to save the parameter.
- Reset to FME Defaults: Changes the field back to the standard (FME) defaults.
- Reset to My Defaults: If you manually edit this field, you can reset it back to your own defaults. Note, however, that you cannot restore your own defaults after resetting to FME defaults.

## Using the New Defaults

- Start a new workspace and specify the format for which you changed defaults.
- Click the Parameters button to open the Settings box. The new default settings will appear.

## Restoring Original Defaults

You can restore original FME defaults to all formats by choosing Reset to FME Defaults.

Warning! Restoring defaults will remove all custom defaults that you have entered.



## FME Options

The following options apply to FME Quick Translator.



Runtime



Coordinate Systems

For other options, please see FME Workbench help topics.

## Runtime

Select **Tools > FME Options** and click the Runtime icon.



Runtime

### **Translation Priority**

You can set the CPU priority for running translations. In most cases, the default **Normal** setting is adequate. If you regularly run large translations, you can keep the default setting, or change it to **Low**, so the translations won't dominate your CPU (and so you can do other work in the foreground while the translation is running). If you have other CPU-intensive tasks running concurrently, you may want to set the priority to **High** to make sure the translation gets its share of the CPU.

### **Track Usage Statistics**

If you select this option, FME will transmit information about your version of Windows and how you use FME. The information collection process is completely anonymous, and your results will be automatically combined with other users' results. The resulting statistics will help us identify trends and usage patterns (for example, which formats and processing facilities are utilized more than others), which in turn will help us focus our development efforts for future versions of FME.

We will not collect your name, address or any other personally identifiable information.

### **Log File Defaults**

- **Save log to file:** Select this option to save the translation log in the default workspace directory. (When you choose **Save to file** in the log pane, the default location will be the location you choose under Default Workspace Directory, below. The default filename will be **translation\_log.txt**.)
- **Append to log file:** Select this option to append log results to the previously generated log instead of overwriting the file.
- **Log timestamp and debugging information:** Select either of these options to add these additional details to your log file output.

### **Log Filter**

Select the type(s) of messages you want to view in the log file. For example, you might want the log to display only Warnings and Errors.

## Coordinate Systems

Select Tools > FME Options and click the Coordinate Systems icon.



Coordinate Systems

### **How to determine which grid shift files are installed**

A number of grid shift files are included when you install FME. For details, see Included Grid Shift Files.

### **How to maintain and edit grid shift files**

This option allows you to maintain and edit grid files. You can manage the grid files by selecting a file and clicking the Edit button (or, simply double-click the file). The dialog that appears lists one or more grid shift files:

- **Add button:** Opens a file browser so you can select a new file and add it to the list.
- **Remove button:** Removes a selected file.

***Note:** We recommend that you make sure that the grid files you need are in the list, and that you remove any files that you do not need. See [NAD27/NAD83 Datum Shifts in U.S. and Canada on FMEpedia](#) for information on why it is important to configure FME to use the correct files.*

- **Fallback Datum:** Select the datum to use if the selected grid file does not cover the area of your input data. Fallback datums are listed in individual .gdc files, which you can open with any text editor.
- **Move Up and Move Down buttons:** Note that these buttons will rearrange the list, but they do not determine the order of precedence of the grid files.

Click OK to apply changes, and Cancel to discard changes.

### **WARNING!**

*If you add a file, you are not copying the file – you are only pointing to the file's location. Thus, if you delete a file from its original location, the entry you create here will point to a nonexistent file. Safe Software recommends that you copy each grid file to FME's Reproject\GridData subdirectory before adding any files to your configuration.*

### **How to add a grid shift file to FME**

Follow these steps to install a grid file so that FME will recognize the file:

1. Copy the new file to FME's Reproject\GridData subdirectory.
2. Select Tools > FME Options and click the Coordinate Systems icon.
3. Select the applicable datum shift and click the Edit button.
4. A dialog displays the files already recognized by FME for the applicable datum shift. To include new files, click the Add button, browse to the applicable directory and find the file in Reproject\GridData subdirectory.
5. Select the file and click OK.

**More Information**

For additional information on grid shift files, see the Coordinate Systems section of this help file, or view these specific topics:

[How Manages Grid Files](#)


Included Grid Shift Files

## Running a Translation

### Setting Up a Basic Translation

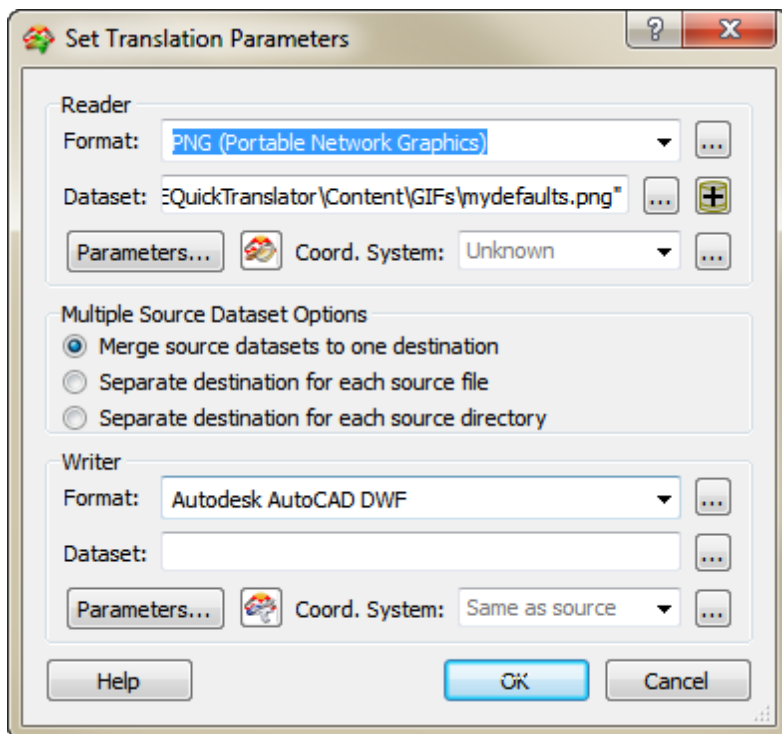
Use an automated translation whenever you want to move data from source to destination quickly and with minimal effort. In most cases, an automated translation will produce exactly the results you require.

You can initiate a translation in any one of the following ways:

- Drag the source file and drop it in the FME Quick Translator window. This is the easiest method because the source information is automatically defined.
- Select **File > Translate**.
- Click the Translate tool .
- Press **Ctrl+T** from the main window.

*Tip: Select Tools > Browse Readers and Writers for a quick overview of FME-supported formats, and their file extensions.*

In each case, the Set Translation Parameters dialog box appears.



### What's Next?

You have the option of further [defining your source information](#) by specifying reader parameters, changing the coordinate system and viewing your input dataset.

## Specifying Source Information

Open the [Set Translation Parameters dialog](#).

The easiest way to specify your source information is to drag a file or directory folder onto the FME Quick Translator main window. If you drag a single file onto the window, FME will know the format from the file extension. Alternatively, you can [Merge Datasets](#) or choose from the formats list.

*If you decide to choose another dataset in a different format, you will have to manually select the new format.*

### Reader Parameters


Reader Parameter boxes allow you to specify additional information relating to some source datasets. When you select one of these formats, the Parameters button will become available. When you click the button, you'll see a dialog that contains options specific to the format you selected.

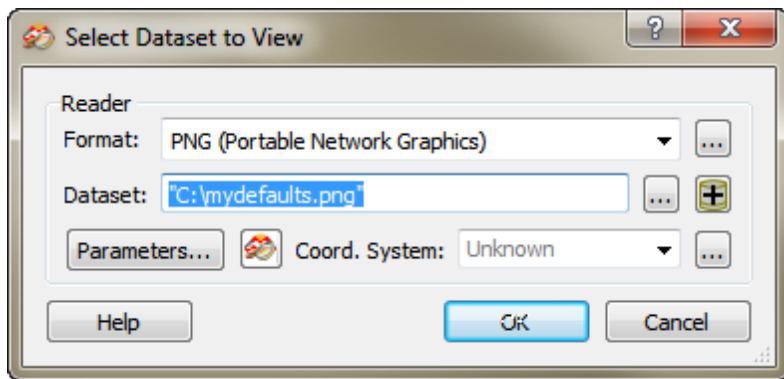
If the format doesn't support any special settings, then the button will remain dimmed (and unavailable).

*You don't have to change the settings if your source format supports this feature – FME will simply use the default settings.*

**Help for Parameters:** When you open a settings box, press the F1 key for detailed help.

### Viewing the Source Data

To view source data, click the Viewer button . This starts the FME Universal Viewer, and displays a [confirmation dialog](#) containing your source data information. Click **OK** to view the dataset.



*You can also select a different data source to view; however the input filename will not change in the Set Translation Parameters dialog.*

You can also start the Universal Viewer by selecting **Viewer** from the **Tools** menu, by pressing **Alt+V**, or from the Mapping File Generation window.

### Coordinate System

This field will usually display **Unknown**, which means that FME will simply use default values. In most cases, the default value is all you'll need to perform the translation.

For more information see:

[Converting Coordinate Systems](#)


## Adding Custom Coordinate Systems

### ***What's Next?***

When you've finished specifying the source information, you can specify the [destination information](#).

### **Choosing from the Formats List**


When you choose a source or destination format from the [pull-down menu](#), the ten most recently used formats are displayed. You can also enter a prefix or character string that will display a list of matching selections.

If you want to view all formats supported by FME, click the Browse button  to display the [Formats Gallery](#).



## Merging Similar Datasets

You can specify multiple datasets of the same format and with the same schema (data model). These datasets will be merged together when you run the translation.

1. Click  in the Set Translation Parameters dialog to display the Advanced Dataset Manager.
2. Click **Add Files** to browse for and select individual files.
3. Click **Add Directories** to browse for and select specific directories.

All files in the specified format in the directories will be included.

*Note: You can check the **Subdirectories** box to include all subfolders below that directory. For formats such as ArcInfo Coverages, the dataset consists of an entire directory.*

4. Check the **Identical Schema** box if you know that all files have the same schema.  
If you check the box, then FME will not have to perform an initial scan of all the files to determine their schemas. Instead, FME will read the first file as representative of the data model.
5. To remove a directory from the list, select it and then click **Remove**.
6. Click **OK**. The new datasets will append to the original dataset name. When you run the translation, it will merge the specified datasets. The log window will display detailed information during the translation.

*Note: The **batch translation** radio buttons will now be enabled. You can choose to merge files to one destination for a regular translation, or choose separate output destinations for each file. If you choose separate output destinations, you will be [running a batch translation](#).*

## Defining the Writer

In the destination area of the Set Translation Parameters dialog, select a writer format and enter a dataset name.

If you enter an existing filename, the existing file will be overwritten. FME will not prompt you before overwriting a file.

## Writer Parameters

Writer Parameter boxes allow you to specify additional information relating to some destination datasets. When you select one of these formats, the Parameters button will become available. When you click the button, you'll see a dialog that contains options specific to the format you selected.

If the format doesn't support any special settings, then the button will remain dimmed (and unavailable).

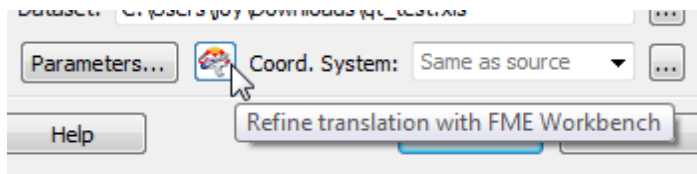
**Note:** *You don't have to change the settings if your destination format supports this feature – FME will simply use the default settings.*

**Help for Output Settings:** When you open a settings box, press the **F1** key for detailed help.

## Refining the Translation in FME Workbench

After you have defined the reader and writer, you can choose to continue setting up your translation in FME Workbench. This means that you'll be continuing the translation using Workbench's user interface; you'll get the same results, but you'll produce them in a graphical environment.

Click the Refine Translation with FME Workbench button:



FME Workbench will start, and New Workspace dialog will be already populated.

If you decide to switch your translation environment to FME Workbench (for example, you might want to consider adding transformers), you can go back and click Cancel in the Set Translation Parameters dialog, and exit FME Quick Translator.

## Coordinate System

This field will usually display **Same as source**, which means that FME will use the same coordinate system as the source file used. In most cases, the default value is all you'll need to perform the translation.

For more information see:

[Converting Coordinate Systems](#)

[Adding Custom Coordinate Systems](#)

## What's Next?

When you've finished specifying the destination information, you can create a custom mapping file.

## Working With Mapping Files

Mapping files control how FME converts data from the selected source format to the selected destination format. FME will read the file from the location specified in the mapping file specification you create, and then use that information to perform the translation. In most cases, customized mapping files contain further translation and transformation operations for FME to perform.

*Tip: To immediately start a translation, double-click the mapping file.*

You can do a quick translation by selecting **File > Translate**, which generates the mapping file and performs the translation in one operation. Using a custom mapping file allows you to generate the mapping file and translate your files in separate operations.

### **What's Next?**

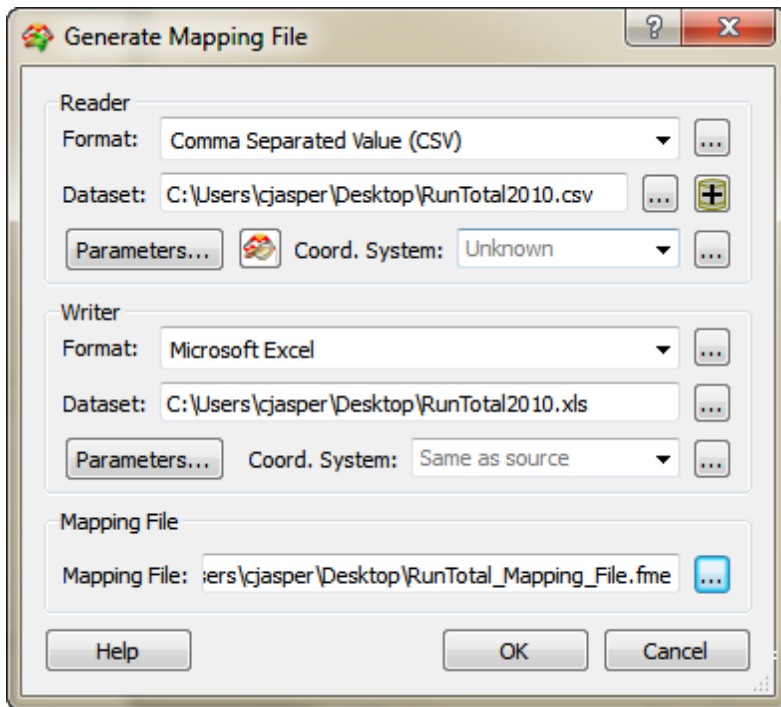
[Creating a Custom Mapping File](#)

## Creating a Custom Mapping File

The first step in the development of a custom mapping file is to use FME's mapping generation capability to create a mapping file.

1. Select **Tools > Generate**, or press **Ctrl+G**.

The Generate Mapping File dialog appears, for example:



2. Enter the reader and writer parameters you want to use for the mapping file.
3. **Parameters:** When you select certain source formats, the **Parameters** button becomes available. These parameters are similar to the input and output settings available from the Set Translation Parameters dialog. When you click this button, you'll see a dialog (which is different for each format), that allows you to adjust settings specific to that particular format. For detailed help in these dialogs, press the **F1** key.
4. Select the folder where you want to save the mapping file, and enter a unique name for it.
5. Click **OK**.

FME will generate the basic mapping file with the source and destination information that you specified. You can then use a text editor to customize this mapping file.

## What's Next?

### [Running a Custom Mapping File](#)

## Running a Custom Mapping File

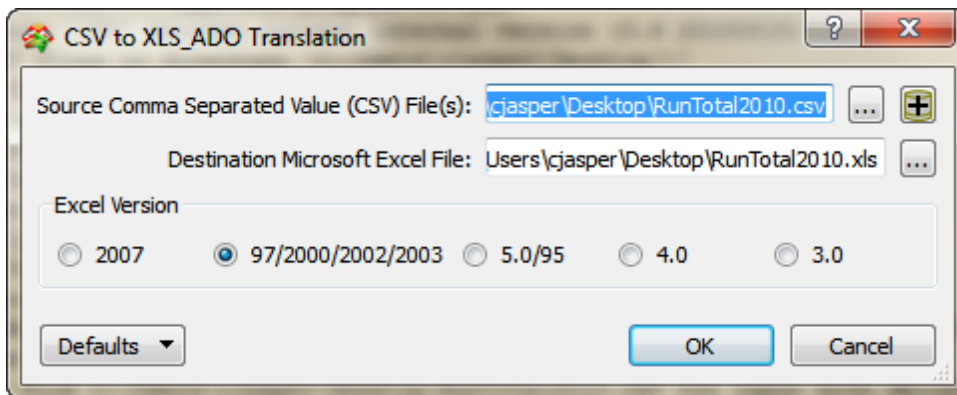
To use a custom mapping file:

1. Select **File > Run**, or press Ctrl+R.
2. Enter the name of the mapping file you want to run.

**Tip:** Remember that you can also drag .fme files directly to the FME window. Depending on the requirements of the mapping file, another dialog box will appear requesting additional parameters, or the file will start to run immediately.

3. Click the **Open** button in the dialog box.

You're prompted if the mapping file requires additional parameters. FME will run the file and display statistics in the FME log view.



If FME detects syntax errors or inconsistencies in your mapping file, they will be displayed in the log view. You can then use any text editor to fix the errors, and then run the file again.

**Tip:** If you rerun the same mapping file, FME will remember the values you used last time, and will provide them to you as defaults.

**Shortcut:** Use the **F5** key to immediately rerun the mapping file or workspace.

## What's Next?

Running a Translation

## **Running a Translation**

Start the translation by either double-clicking the mapping file you selected in [Creating\\_a\\_Custom\\_Mapping\\_File.htm](#), or by clicking **OK** in the Set Translation Parameters dialog. The FME will begin the translation, and you'll see information in the log window.

The amount of time the translation takes depends on different factors that may include the processing power of the FME host machine, the size of your dataset, and the amount of processing that FME has to do.

### [Stopping a Translation](#)

In most cases, however, a translation will take a few minutes at most, and the Log view will display TRANSLATION SUCCESSFUL upon completion. You can scroll back through the log to check specific statistics.

### [Viewing the Log File](#)

#### ***What If a Translation Fails?***

If you see a message that says TRANSLATION FAILED, the FME log will often be the best source to find out what went wrong. Usually it will be something simple, like a missing file. Sometimes, however, you will have to perform some troubleshooting on either your source data or on the mapping file. If you have edited your mapping file, the problem could be as simple as a missing backslash.

#### **More Information:**

A good source of information and help is [FME Talk](#). You can also search the [mailing list archive](#) for similar circumstances.


#### ***What Next?***

You have completed the steps involved in performing a translation using a default mapping file. The output from the translation is the output filename you specified in the beginning.

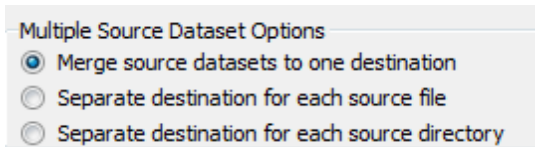
### [Viewing the Output File](#)

## Running a Batch Translation

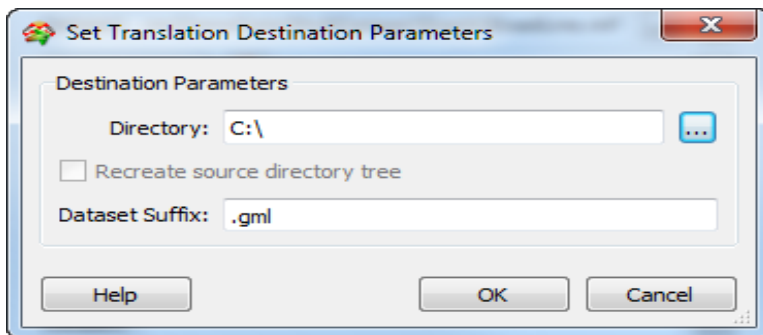
The FME Quick Translator interface supports batch execution and batch script creation. For example, you can translate a large number of input files, and produce separate outputs for each. You can choose to run the batch translation immediately, or save a batch script to be run later, potentially on another platform.

1. Select File > Translate to open the Set Translation Parameters dialog.
2. Specify the Reader format.
3. Click the  button to [specify multiple datasets](#).

This will enable the Multiple Source Dataset Options:



4. Click *Separate destination for each source file*.
5. Select a Writer format.
6. Click the Browse button beside the Dataset text box.
7. Specify the directory for the output files and (optionally) a suffix to append to the dataset (for example, a file-type suffix).



8. Click OK to close the dialog box.
9. Click OK in the Set Translation Parameters dialog to start the translation.

## Saving a Batch Script to Run Later

1. Select Tools > Create Batch.

This displays the Set Batch Script Parameters dialog with an additional field to accommodate saving a batch script.

2. Follow the steps above to specify the translation parameters.

At the bottom of the dialog, you will see a Batch Script area, with a filename field.

3. Type or browse for the location of the batch translation file.

Note that batch files consist of both a .tcl file and a .bat file. Both files are needed to run the batch translation.



4. Click OK.

The batch script will be saved.

### ***Running the Scripts Outside of FME***


Go to the Windows DOS prompt and either type the name of the .bat file or type `fme tclfilename.tcl`.



## **Viewing the Output File**

You can view the output file with the FME Universal Viewer. If the Viewer isn't already running, start it and open the output dataset file.

## Stopping a Translation


When you're running a translation, the Stop tool  becomes available. Click the tool to immediately stop the translation.

*Note: Sometimes stopping a translation causes FME to create temporary files that can take up unnecessary disk space. It's a good idea to select Tools > Purge Temporary Files (or press Alt + C) after you stop a translation.*

### **What happens after you stop a translation?**


When you stop a translation, the log view will display **Translation Stopped**. The output file is left incomplete and is not usable. If you run the file again, the entire translation will restart, overwriting any partial results from a previously aborted translation.

## Viewing the Log File


You may find it useful to save a text file of statistics or other processing information from a translation run. You can save the displayed text by clicking  or choosing **Save As** from the Log menu.

Enter a filename in the dialog box that appears, and click **Save**.

To copy and paste text, select the text that you want to copy and then perform one of the following actions:

- Click .
- Select **Log > Copy**.
- Press **Ctrl+C**.
- Right-click the selected text to display a context menu.

Then paste the text into a file with a text editor, and save that file.


To search for a keyword or text string in the log, click , or press **Ctrl+F**. To search for the same text again, click , or press the **F3** button.

## Coordinate Systems and Datums

### Converting Coordinate Systems

For systems that know their coordinate system (such as MapInfo and DLG), this field will display *Read from Source* and FME will read the coordinate system from the source dataset. For most other input sources, the field will display *Unknown* (which simply means that FME will use default values). In most cases, the default value is all you'll need to perform the translation.

You can always choose to override the defaults and choose a new coordinate system.

1. Click the Browse button  beside the Coordinate System text field.
2. When the Coordinate System Gallery appears, in the field next to "contains", enter part of a filename, any character string, or even a file suffix, and the list will display results starting from any matching name.

You can also select a new coordinate system and view properties on selected coordinate systems. To do so, select a coordinate system from the list and click Properties.

To add a coordinate system to the Set Translation Parameters dialog, select it from the list, and then click OK. FME will use the selected coordinate system for the translation.

### How FME Identifies Coordinate Systems

For more information on mapping file directives and coordinate systems, see [How FME Identifies Coordinate Systems](#).

### More Help With Coordinate Systems

For more information on coordinate systems and FME, open the Coordinate System Gallery. Press the F1 key or click the question mark button in the top-right corner of the dialog.

## How FME Identifies Coordinate Systems

These mapping file directives are used to identify the source and destination coordinate system:

```
<READER_KEYWORD>_COORDINATE_SYSTEM <coordinateSystemName>  
<WRITER_KEYWORD>_COORDINATE_SYSTEM <coordinateSystemName>
```

If a coordinate system is specified in both the source format and the mapping file, the coordinate system in the mapping file is the one that will be used. The coordinate system specified in the source format is not used, and a warning is logged. If a source coordinate system is not specified in the mapping file and the format or system does not store coordinate system information, then the coordinate system is not set for the features that are read.

If a destination coordinate system is set and the feature has been tagged with a coordinate system, then a coordinate system conversion is performed to put the feature into the destination system. This happens after features leave the factory pipeline, but before they enter the transformation process.

If the destination coordinate system was not set, then the features are written out in their original coordinate system.

If a destination coordinate system is set, but the source coordinate system was not specified in the mapping file or stored in the source format, then no conversion is performed. The features are simply tagged with the output system name before being written to the output dataset.

## How FME Manages Grid Shift Files

Grid shift files are used when reprojecting between coordinate systems that have different datums.

FME supports conversions between coordinate systems using different datums. Many datum transformations are not mathematically definable and require the use of grid of shifts. If you attempt to make a datum transformation of this kind without the appropriate grid shift file in place, FME will abort the translation.

The FME installation includes a configuration file for managing the use and location of grid shift files.

These files are found in the Reproject directory in the FME installation directory. For example, the file [Nad27ToNad83.gdc](#) includes file paths to both the Canadian and US grid shift files for the NAD27 <-> NAD83 transformations.

Included Grid Shift Files

Adding Grid Shift Files

Coordinate Systems

## **Working with Mapping Files**

The translation of features from a source system to a destination system is completely controlled by rules specified in the mapping file. This ASCII text file consists of a series of translation specifications.

FME Workbench workspaces are mapping files with special comments in them that Workbench uses to recreate as a graphical representation.

The mapping file consists of free-form ASCII text organized into a series of lines. To expedite maintenance, a mapping file may contain both line and block comments which can be nested, may contain text expansion macros, and may include other text files. Each non-comment line in the correlation file starts with a keyword recognized by some component of the translation engine.

A syntax highlighting editor such as UltraEdit, Textpad, or VIM can greatly improve productivity when working with mapping files. FME ships with syntax files for these editors, as well as templates for the FME functions and factories that greatly speed their entry.

## Writer Definitions

Most writers in FME require the schema for the output data to be defined before translation begins. This is done through DEF lines, which take this syntax:

```
<writerKeyword>_DEF <FeatureType>  
<AttributeName> <AttributeType>  
<AttributeName> <AttributeType>  
.  
.  
.
```

For example, for the ESRI Shape format, the feature type defines the file name for the output Shapefile. An ESRI Shapefile named ROADS.shp with attributes LENGTH and NUMLANES can be defined like this:

```
SHAPE_DEF ROADS \  
LENGTH char(30) \  
NUMLANES char(30)
```

The details of what feature type names, attribute names, and types are allowed depends on the individual format. Consult each format's writer documentation for more details.



## Dynamic Feature Type Definitions (Advanced FME)

When writing out a large number of feature types, it can be tedious to create definitions for every single feature type you may want to write out. In addition, the output feature types may depend on the input data, and may not be known before translation begins. In this situation, FME can create feature types dynamically, so that DEF lines don't need to be written in a mapping file. Many, but not all, FME writers support dynamic feature type creation.

In order to use this feature, a single DEF line must be given in order to provide a template for the dynamic feature types. Each feature type that is dynamically created will be a copy of the template feature type. The `fme_template_feature_type` attribute is used to define these dynamic feature types. When a feature is written with a feature type not defined on any DEF line, the writer checks if the `fme_template_feature_type` attribute is defined on that feature. If it is, the value of that attribute is used as the feature type name for the template that this feature should use.

In the example shown below, we are writing to the GML2 writer, and use the `CreationFactory` to simulate reading a large number of feature types. Normally we would have to have a `GML2_DEF` section for every feature type from 1 to 100, but in this case we have only a single definition (Template), which is used as a template for every dynamic feature type that will be created. Every one of the 100 layers that will be created in the GML2 file will have a schema identical to that of the Template definition, with a different feature type.

```
READER_TYPE NULL
NULL_DATASET null
WRITER_TYPE GML2
GML2_DATASET "c:/out.gml"
GML2_SCHEMA_MODE CREATE
# -----
# Create 100 features, with the count attribute increasing for each one.
FACTORY_DEF * CreationFactory \
  2D_GEOMETRY 0 0 \
  CREATE_AT_END no \
  NUMBER_TO_CREATE 100 \
  OUTPUT FEATURE_TYPE 2DCREATOR_CREATED \
  count @Count(counter,0)
# -----
# Set each feature's feature type to the count attribute, and set
# the fme_template_feature_type.
FACTORY_DEF * TeeFactory \
  FACTORY_NAME "2DCREATOR_CREATED -> Template Correlator" \
  INPUT FEATURE_TYPE * \
  OUTPUT FEATURE_TYPE * \
  @Transform(FME_GENERIC,GML2) \
  @FeatureType(&count) \
  @SupplyAttributes(fme_template_feature_type,Template)
# -----
NULL *
GML2 *
# -----
GML2_DEF Template \
  gml2_type gml2_point \
  attr1 gml2_char(10) \
  attr2 gml2_char(10)
```

## Line Continuation

Long mapping file lines may logically continue to the next line by using a backslash (\) as the last character. For example, the following two lines:

```
IGDS 45 igds_color 8 \  
    igds_style 3
```

are equivalent to this line:

```
IGDS 45 igds_color 8 igds_style 3
```

There is no limit to the number of consecutive lines that may be joined together using continuation characters.

Tip: Breaking long lines into several smaller parts makes your mapping files easier to read and maintain. The continued parts of lines are usually indented to aid readability.

## Quoted Text

Parameters or string literals containing spaces or tabs must be enclosed in quotation marks. The quotation marks are stripped off by FME when the mapping file is processed and the string they contain is treated as a single token.

The example below assigns the text *Hello There* to the `text.textString` attribute of a SAIF object.

```
SAIF Text::TRIM text.textString "Hello there"
```

If a parameter or string literal contains a quotation mark, the quotation mark is preceded by a backslash, as shown in the example below:

```
SAIF Text::TRIM text.textString "Some \" label"
```

## Escaping Commas

Function parameters that contain commas must have the commas escaped; otherwise, they will be considered parameter separators. This is done by placing a backslash before the comma. The example below passes the string "Hi, There" to @Log as the message it should output.

```
@Log("Hi\", there")
```

## Block Comments

Sometimes a larger discussion of the mapping file contents is desired. To accomplish this without the inconvenience of starting each line of the comment with a special character, a block comment may be used. Block comments start with slash-asterisk (/\*) at the beginning of the line, and end with an asterisk-slash (\*/) at the end of the line.

The following examples illustrate the use of block comments in a mapping file:

```
/* This is a one-line block comment */  
/*  
This is a multi-line comment.  
Any text in here will be ignored  
/* This is a nested comment */  
*/
```

Tip: Because block comments may be nested, they are often useful for disabling large sections of a mapping file during testing.

## MACRO Directives

To avoid duplicating portions of correlation lines repeatedly throughout a mapping file and to allow creating symbolic constants to avoid hardcoding numbers, the mapping file provides a simple textual substitution mechanism. Macros are defined by lines starting with the keyword `MACRO` which must be in uppercase letters. The word following the `MACRO` keyword is the name of the macro, which will be replaced by the contents of the rest of the line when it is expanded on other lines. A macro expansion is requested by enclosing the macro name with `$(` and `)`. The lines below:

```
MACRO blue 8
IGDS 32 igds_color $(blue) igds_style 3
```

will be expanded to:

```
IGDS 32 igds_color 8 igds_style 3
```

**Tip:** Using macros as symbolic constants for color numbers and line styles eases the maintenance of mapping files.

Macro definitions may themselves refer to other macros which will be expanded recursively at the time the outermost macro is expanded. Macros may be redefined at any time in the file. A macro definition remains in effect from the point of definition until the end of the file, or until it is redefined. Macro definitions may, of course, be spread over several lines using the continuation character (`\`) to join consecutive physical lines. A macro may not, directly or indirectly, refer to itself.

FME flags any reference to an undefined macro and stops the translation if one is encountered.

The next example illustrates a more complex use of macros to define a large, commonly used correlation block. Note that the first macro itself references another macro to set the color of the text. Once the two macros are defined, they are then used in the specification of the translation from IGDS to SAIF.

```
MACRO IGDSTextSpec \
  igds_color $(blue) \
  igds_type igds_text \
  igds_style 0 igds_class 0 igds_weight 1\
  igds_text_size %size \
  igds_text %string \
  igds_font %font \
  igds_rotation %orientation
MACRO SAIFTextSpec \
  textOrSymbol.characterHeight %size \
  textOrSymbol.text %string \
  textOrSymbol.fontName %font \
  textOrSymbol.orientation @SAIFAngle(%orientation)
#-----
# Feature KB14250000 -- Text (Hydrographic)
IGDS 44 igds_color 2 $(IGDSTextSpec)
SAIF OtherText::TRIM \
  textType hydrographic $(SAIFTextSpec)
```

## **Substituting Strings in Mapping Files**

Occasionally, a character can cause a conflict in the mapping file parser. If this occurs, you can substitute safe character strings in many formats, functions and factories, including:

- Geodatabase writer (Attribute Definitions > subtypes, Attribute Definitions > domains)
- @Lookup
- PythonFactory
- @SearchList
- @SupplyAttributes
- @Tcl2

### **Character Substitutions**

The encoding method is inspired by XML character entities.

*Note: Substitutions are made only in attribute type definitions.*

| <b>Symbol</b> | <b>Replaced with</b> |
|---------------|----------------------|
| <             | <lt>                 |
| >             | <gt>                 |
| "             | <quote>              |
| &             | <amp>                |
| \             | <backslash>          |
| /             | <solidus>            |
| '             | <apos>               |
| \$            | <dollar>             |
| @             | <at>                 |
|               | <space>              |
| ,             | <comma>              |
| (             | <openparen>          |
| )             | <closeparen>         |
| [             | <openbracket>        |
| ]             | <closebracket>       |
| {             | <opencurly>          |
| }             | <closecurly>         |
| ;             | <semicolon>          |
| \r            | <cr>                 |
| \n            | <lf>                 |
| \t            | <tab>                |
| \a            | <bell>               |
| \b            | <backspace>          |
| \v            | <verttab>            |
| \f            | <formfeed>           |



## International Characters

| International Character Support | Replaced with  |
|---------------------------------|--|
| <specialCharacter>              | <uABCD><br>(where ABCD is the 4-digit hex value for the equivalent UTF-16 character) |

The encoding described above takes care of characters that may cause problems with mapping file parsing, but it does interfere with some "special" characters from some encodings (where *special* means characters which are not a part of the 7-bit ASCII set.) The problems appear in systems where one or more bytes of a multibyte character are special to traditional encoding.

To counter this, FME supports Unicode-based character encoding. Encoding of the special characters is now done as a special tag, <uXXXX>, where XXXX is the hex representation for the UTF-16 character.

This very simple encoding scheme is based on one observation (assumption): any characters which will cause problems inside of the encoding will involve one or more bytes which has the high bit set. Thus, in order to traditionally encode any string containing a character where a high bit is set, we have to convert the strings to a form where the encoding system is neutralized, where we can somehow represent that character with a string of bytes containing only 7-bit values (i.e. 7-bit clean values).

The encoding process basically applies the following heuristic:

If the string contains any bytes with the high bit set:

- Convert the string to UTF-16
- Convert the string 16-bit UTF-16 characters to a 7-bit traditionally encoded string

The outcome from this is an encoded string which is 7-bit clean (i.e., has no high bits set), and which can be decoded to get exactly the original characters for the unencoded result.

When encountering the character sequence <u anywhere within an encoded string, the decoder will be able to reverse the above process after a "traditional" decoding, to get back to the original UTF-8 or system-default encoded string.

If any 16-bit value is too large for a 7-bit character, replace it with \uXXXX, where XXXX is the hex representation for the 16-bit value.

### **Example**

According to the encoding method, the attribute type

```
coded_domain(TextDom:char(50):a:"the letter a":b:"the letter b")
```

would be represented as

```
coded_domain(TextDom:char<openparen>50<closeparen>:a:  
<quote>the<space>letter<space>a<quote>:b:<quote>the<space>  
letter<space>b<quote>)
```

within a mapping file. Notice that the encoding is only applied to the body of the coded\_domain type, i.e. coded\_domain(<body>). Similarly, it would only be applied to the body of range\_domain(<body>), subtype(<body>), and subtype\_codes(<body>) which are Geodatabase attribute types and to enum(<body>) and set(<body>) which are both MySQL attribute types.

## **Predefined Macros**

Most mapping files contain external references to the source and destination datasets for the translation. Some mapping files include external references to other mapping files, for example, mapinfoMacros.fmi. Advanced mapping files may reference external Tcl scripts used to perform sophisticated operations. In all of these cases, you can simplify the portability of the mapping file by using one of the macros automatically defined by FME.

### ***FME\_ASC\_xxx***

This set of macros each expands to the corresponding ASCII character. The xxx in FME\_ASC\_xxx is a character set index number, and the macro value is the character at that index in the ASCII character set.

For example, FME\_ASC\_2 expands to the ASCII character whose index value is 2.

## **FME\_BASE**

This macro expands to yes if the FME that is running is licensed only for FME Base Edition capabilities. If the FME is licensed at the Professional level or greater, then this macro has the value no.

This macro could be used to selectively include factories by using (as a single, continuous line):

```
INCLUDE base_factories_$(FME_BASE).fmi
```

in a mapping file, and then having base\_factories\_yes.fmi and base\_factories\_no.fmi in the mapping file directory.

***Note: The deprecated FME\_DESKTOP macro has the same value. The FME\_DESKTOP macro will still function in older mapping files.***

***FME\_BUILD\_DATE***

This macro expands to the date that the FME executable processing the mapping file was built, in YYYYMMDD format.

***FME\_BUILD\_NUM***

This macro expands to the build number of the FME executable processing the mapping file.

***FME\_HOME***

This macro expands to the directory where the FME executable resides. On UNIX it uses the slash (/) character as directory separator, and on Windows it uses the backslash (\). It includes a trailing slash, independent of the platform.



***FME\_HOME\_DOS***

This macro expands to the directory where the FME executable resides. It always uses the backslash directory separator, independent of the platform, and does not have a trailing separator (either slash or backslash).

***FME\_HOME\_UNIX***

This macro expands to the directory where the FME executable resides. It always uses the slash directory separator, independent of the platform, and does not have a trailing separator (either slash or backslash).

## ***FME\_MF\_DIR***

This macro expands to the directory where the mapping file resides. On UNIX it uses the slash (/) character as directory separator, and on Windows it uses the backslash (\) character. It includes a trailing slash, independent of the platform.

***Note: In older versions of FME, FME\_MF\_DIR was called FME\_CF\_DIR. Although FME still accepts FME\_CF\_DIR in place of FME\_MF\_DIR to accommodate backwards compatibility, its use is not recommended.***

***FME\_MF\_DIR\_DOS***

This macro expands to the directory where the mapping file resides. It always uses the backslash separator, independent of the platform, and does not have a trailing separator (either slash or backslash).

***FME\_MF\_DIR\_DOS\_MASTER***

This macro is like FME\_MF\_DIR\_MASTER, but all directory separators are the backslash, and it does not include a trailing separator.

***FME\_MF\_DIR\_MASTER***

Similar to FME\_MF\_DIR, this macro expands to the directory of the outermost original mapping file except if the mapping file names ends with .tmp. Unlike FME\_MF\_DIR, it does not change value within any INCLUDED mapping file fragments.

***FME\_MF\_DIR\_UNIX***

This macro expands to the directory where the mapping file resides. It always uses the slash separator, independent of the platform, and does not have a trailing separator (either slash or backslash).

***FME\_MF\_DIR\_UNIX\_MASTER***

This macro is like FME\_MF\_DIR\_MASTER, but all directory separators are the forward slash, and it does not include a trailing separator.



***FME\_MF\_NAME***

This macro expands to the name of the mapping file including the extension, but not including the directory specification.

***FME\_MF\_NAME\_MASTER***

Similar to FME\_MF\_NAME, this macro expands to the name of the outermost original mapping file except if the mapping file names ends with .tmp. Unlike FME\_MF\_DIR, it does not change value within any INCLUDED mapping file fragments.

***FME\_PRODUCT\_NAME***

This macro expands to the version name of the FME product that is processing the mapping file.

For example: FME 2007

## File Locations

It is recommended that any files referenced by your mapping file be located in the same directory as mapping file or in a directory that can be expressed relative to the mapping file directory. This allows you to identify the location of these files using the FME\_MF\_DIR macro (or one of its variants) in the place of using hard coded directory specifications. By avoiding hard coded directories, you can make your mapping files inherently portable across installations. For example:

```
DEFAULT_MACRO SourceDataset "${FME_MF_DIR}/../maps/082e.dgn"
```

Include files provided as part of the FME can be referenced using the FME\_HOME macro as follows:

```
INCLUDE "${FME_HOME}/metafile/mapinfoMacros.fmi"
```

**Note: Although FME treats both slash and backslash as a legal directory separators, independent of the platform, slash is the preferred separator because it avoids ambiguity with the escape character.**

Where Tcl is involved, all paths should be enclosed by braces. For example:

```
INCLUDE [source ${FME_MF_DIR}/gen.tcl]; \  
genLines ${FME_MF_DIR} ]
```

In most situations, the FME\_MF\_DIR and FME\_HOME macros will suffice. However, if you encounter a platform-specific issues where the standard macros do not give the desired result, the \_UNIX and \_DOS variants can be used in their place. As an example, consider the following line that duplicates a file using the DOS copy command:

```
@System("copy \"${FME_MF_DIR_DOS}\\MSFORMS.DBF\" \  
\"${DestDirectory}\"")
```

For this command to be interpreted correctly, it is necessary to use the backslash separator.

The following simple mapping file example can be used to troubleshoot the use of these macros on any platform:

```
#-----  
# Specify a Null Reader and Writer  
READER_TYPE NULL  
WRITER_TYPE NULL  
NULL_DATASET NULL  
#-----  
# Print the values of all predefined macros  
TCL puts {FME_MF_DIR= ${FME_MF_DIR}}  
TCL puts {FME_MF_DIR_UNIX= ${FME_MF_DIR_UNIX}}  
TCL puts {FME_MF_DIR_DOS= ${FME_MF_DIR_DOS}}  
TCL puts {FME_HOME= ${FME_HOME}}  
TCL puts {FME_HOME_UNIX= ${FME_HOME_UNIX}}  
TCL puts {FME_HOME_DOS= ${FME_HOME_DOS}}  
TCL puts {FME_MF_NAME= ${FME_MF_NAME}}  
TCL puts {FME_BUILD_NUM= ${FME_BUILD_NUM}}  
TCL puts {FME_BUILD_DATE= ${FME_BUILD_DATE}}  
TCL puts {FME_PRODUCT_NAME= ${FME_PRODUCT_NAME}}
```

**Note: All of the examples shown in this topic should appear as a single continuous line.**

### ***DEFAULT\_MACRO Directive***

Default macros are used to supply a value to a macro name **only** if the macro was undefined. If the macro was already defined, the DEFAULT\_MACRO line is equivalent to that of the MACRO directive.

```
DEFAULT_MACRO CELL_LIB "DEFAULT.LIB"
```

## **INCLUDE Directive**

To allow for modularity and reuse of portions of mapping files, a facility exists to allow mapping files to *include* other mapping files. The net effect is that the entire contents of the included file are processed as though they were pasted into the mapping file containing them. Breaking mapping files into smaller pieces allows reuse of mapping file portions, which makes maintenance easier.

Tip: GUI directives should only appear in the main mapping file. GUI directives that appear in files included by the main mapping file, either directly or indirectly, will be ignored.

The INCLUDE keyword is followed by the name of the file to include. If the file name given is not an absolute path, then the file name is assumed to be relative to the location of the containing file. For example, the control fragment below:

```
INCLUDE colors.fmi
```

causes the contents of the colors.fmi file to be read before the rest of the file is processed. The colors.fmi file must be in the same directory as the mapping file that included it. Of course, absolute path names may also be specified in an INCLUDE statement.

***Note: Mapping files are named with an .fme extension. Included files are usually named with an .fmi extension.***

INCLUDE files are often used to define macros that are used by many mapping files. In this way, a mapping file change may be made in just one place and affect all mapping files that include the changed file.

Tip: The file name to be included may be composed of macros. For example, INCLUDE \$(\_COLOR\_SETTINGS).

Tip: There is no limit on the nesting of INCLUDE files; however, circular includes are not allowed.

The INCLUDE directive is also used for Dynamic Mapping File Creation.

## Dynamic Mapping File Creation

To further assist in keeping mapping files modular and maintainable, mapping file fragments can be generated programmatically. This is accomplished through two extensions to the INCLUDE syntax.

If the INCLUDE target begins with a pipe (|) character, then FME assumes that the INCLUDE target is a program which should be executed. The standard output of the program is then included into the mapping file. This allows arbitrary shell scripts to be executed to produce FME mapping file code. In this way, portions of the mapping file can be created from tables which are more easily maintained than large repetitive blocks of mapping file code.

If the INCLUDE target is enclosed in square brackets ([ ]), then FME assumes the INCLUDE target is a Tool Command Language (Tcl) script that should be executed. Any output written by the script using the puts Tcl command is then included in the mapping file. Because the Tcl interpreter is part of FME, this approach is more portable than using the pipe method of dynamic mapping file creation. It is also more efficient, because no new process needs to be started to do the generation. For more information on Tcl, see @Tcl2 in the *FME Functions and Factories* manual. FME uses Tcl version 8.5.2.

Tip: FME renames the Tcl puts command to puts\_real. If the script needs to output to other files, then puts\_real should be called.

In both methods of dynamic mapping file creation, the program or script is executed with the mapping file directory as the current working directory. This allows any configuration tables used to drive the script or program to be located in the same directory as the mapping file and, therefore, be easily found.

### Dynamic Mapping File Creation Example 1

The awk script resides in the file create.awk which contains this line:

```
{print "DWG_DEF", $1, "autocad_color", $2, "autocad_linetype", $3}
```

A table listing the names of AutoCAD layers, their colors, and their linetypes is found in a file called layers.lst and contains these three lines:

```
ROADS 13 CONTINUOUS  
RIVERS 4 DASHED  
RAILWAYS 7 CONTINUOUS
```

This INCLUDE line:

```
INCLUDE |awk -f create.awk < layers.lst
```

causes these lines to be generated into the mapping file:

```
DWG_DEF ROADS autocad_color 13 autocad_linetype CONTINUOUS  
DWG_DEF RIVERS autocad_color 4 autocad_linetype DASHED  
DWG_DEF RAILWAYS autocad_color 7 autocad_linetype CONTINUOUS
```

This example shows how the text processing language awk is used to generate mapping file fragments to be included in a mapping file. In this example, an awk script located in the same directory as the mapping file is executed and its results are included in the mapping file.

### Dynamic Mapping File Creation Example 2

In this example, a Tcl script located in the same directory as the mapping file is executed and its results are included in the mapping file. The Tcl script resides in the file create.tcl which contains these lines:

```
proc makeDefs {filename} {  
    set file [open $filename r]
```

```
while {[gets $file line] >= 0} {  
  set layer [lindex $line 0]  
  set col   [lindex $line 1]  
  set lt    [lindex $line 1]  
  puts "DWG_DEF $layer autocad_color $col  autocad_linetype $lt"  
}  
close $file  
}
```

A table listing names of AutoCAD layers, their colors, and linetypes is found in a file called layers.lst and contains these three lines:

```
ROADS 13 CONTINUOUS  
RIVERS 4 DASHED  
RAILWAYS 7 CONTINUOUS
```

This INCLUDE line:

```
INCLUDE [source create.tcl ; makeDefs layers.lst]
```

will cause these lines to be generated and parsed into the mapping file:

```
DWG_DEF ROADS autocad_color 13 autocad_linetype CONTINUOUS  
DWG_DEF RIVERS autocad_color 4 autocad_linetype DASHED  
DWG_DEF RAILWAYS autocad_color 7 autocad_linetype CONTINUOUS
```

These techniques can be extended to more sophisticated input files and output requirements. For example, transfer specifications and \_DEF lines can be generated this way from comma-separated value files which are easily edited using spreadsheet tools.

This example shows how the same problem is solved using the Tcl scripting language. This has the advantage of being slightly more efficient and portable since no external program is started to process the table.



## **Environment Variable Expansion**

Sometimes it is useful to pick up the value of an environment variable within a mapping file. This allows session settings to be part of a user's environment rather than requiring them to be specified each time a mapping file is run. An environment variable expansion is requested by enclosing the environment variable name with `${` and `}`.

Tip: Each operating system defines its own mechanism for setting environment variables. On UNIX, they are most often set in `.login` or `.cshrc` scripts that run when a user logs on. On Windows, they may be set in an `AUTOEXEC.BAT` file or by using the Control Panel.

Assuming that the environment variable `FME_CONTROL_DIR` has been set to

```
/usr/control_files
```

then this line:

```
INCLUDE ${FME_CONTROL_DIR}/mappings.fme
```

will be expanded to:

```
INCLUDE /usr/control_files/mappings.fme
```

Environment variable values may themselves refer to other environment variables, which are expanded recursively when the outermost environment variable is substituted. An environment variable may not, directly or indirectly, refer to itself.

FME flags any reference to an undefined environment variable, and stops the translation if one is encountered.

## Static Function Evaluation

In certain situations it is convenient to execute an FME attribute function while the mapping file is parsed. The value returned from the function is then used as if it was part of the original mapping file. In this way, certain settings are determined dynamically rather than by prompting the user.

An FME function may be invoked at mapping file parse time by enclosing it with `[$[ and ]]`. The function may take any type of arguments, including macros which are expanded prior to its evaluation.

Only functions returning a value are legal in this context.

For example, if the mapping file contains the output shape file line:

```
SHAPE road creationTime $[@TimeStamp("^\Y^m^d^H^M^S")]
```

then all road objects created will have the same value for their timestamp attribute. The timestamp is the parse time of the mapping file. However, if the mapping file contained the line:

```
SHAPE road creationTime @TimeStamp("^\Y^m^d^H^M^S")
```

all road objects created would have different time stamps, reflecting the time this object was created.

Other FME functions that are often useful in this context are `@Lookup`, `@TCL`, and `@Evaluate`.

### Example: Reprojecting Coordinates

The following example shows how to use static function evaluation to reproject coordinates from LL83 to UTM, and specify the bounding coordinates for the search envelope in the destination coordinate system (UTM).

```
READER_TYPE NULL
NULL_DATASET null
WRITER_TYPE NULL

# Dump out the mapping file after it is parsed -- this way we can see
# if our procedures worked.

FME_DEBUG MAPPING_FILE

# Set up some macros for testing purposes

MACRO _SDE3MINX 490000
MACRO _SDE3MINY 5000000
MACRO _SDE3MAXX 510000
MACRO _SDE3MAXY 5300000

# Define two TCL procedures, which will reproject a coordinate and
# return either the X or the Y. To do this, we make use of the
# TCL "FME_Execute" to do @Reproject on attributes of a feature.
# We "fake out" attributes with the X and Y we want by setting them
# in the "FME_Attributes" array. Then we just return the reprojected
# value.

TCL proc pointReprojectX {sourceCsys destCsys sourceX sourceY} { \
global FME_Attributes; \
set FME_Attributes(x) $sourceX; \
set FME_Attributes(y) $sourceY; \
FME_Execute Reproject $sourceCsys $destCsys x y; \
return $FME_Attributes(x); \
```

```

}

TCL proc pointReprojectY {sourceCsys destCsys sourceX sourceY} { \
global FME_Attributes; \
set FME_Attributes(x) $sourceX; \
set FME_Attributes(y) $sourceY; \
FME_Execute Reproject $sourceCsys $destCsys x y; \
return $FME_Attributes(y); \
}

# Now, use "static function evaluation" (${ }) to actually call the TC
# procedures defined above, which will do the conversion and return the
# correct values.

SDE30_SEARCH_ENVELOPE ${@TCL("pointReprojectX UTM27-15 LL83 $_SDE3MINX $_SDE3MINY")} \
  ${@TCL("pointReprojectY UTM27-15 LL83 $_SDE3MINX $_SDE3MINY")} \
  ${@TCL("pointReprojectX UTM27-15 LL83 $_SDE3MAXX $_SDE3MAXY")} \
  ${@TCL("pointReprojectY UTM27-15 LL83 $_SDE3MAXX $_SDE3MAXY")}

# This is just used to give the mapping file something to do...

FACTORY_DEF * CreationFactory \
OUTPUT FEATURE_TYPE junk @Log()

```

**Note: The example illustrates a technique, but does not perform any translation.**

## **Command-Line Interface**

While FME's graphical user interface is well suited for most uses, FME also has a powerful and flexible command-line interface that enables it to be used as a component of a batch processing environment.

## **Starting FME from the Command Line**

Type *fme* on the command line to start the FME command-line utility.

## Using Command-Line Options

The *fme* command supports the following options:

| Syntax  | Reference Section   |
|---|---|
| fme COMMAND_FILE <commandFile>                                      | <a href="#">Executing Batch Files (Multiple Runs Per Session)</a> |
| fme Generate ...  | <a href="#">Generating Mapping Files</a>                          |
| fme GENTRANS  | <a href="#">Performing Generic Translations</a>                   |
| fme LIST_TRANSFORMERS [VERBOSE]<br>fme LIST_UNLICENSED_TRANSFORMERS | <a href="#">Listing Transformers</a>                              |
| fme <mappingFile>   | <a href="#">Running Custom Mapping Files</a>                      |
| fme PARAMETER_FILE  | <a href="#">Generating Mapping Files</a>                          |
| fme PROTECT <sourceFile> <destFile>                                 | <a href="#">Reading Command-Line Parameters from a File</a>       |
| fme <TclFile> ...   | <a href="#">Reading Command-Line Parameters from a File</a>       |

## Executing Batch Files (Multiple Runs Per Session)

### **fme COMMAND\_FILE <commandFile>**

This option is used to execute an FME batch file, as follows:

```
fme COMMAND_FILE <commandFile>
```

Normally, FME starts up and shuts down for each translation. In cases where users want to process a large number of files, the time taken to start and stop can become significant. In order to operate more efficiently in such situations, FME supports a mode where it runs several times in a single session without the resources required to restart.

To use FME in this way, a command file must be created that consists of multiple FME command lines. These command lines are executed in the order they are specified when FME is run.

A command file is invoked using the following syntax:

```
fme COMMAND_FILE <cmdFile>
```

The command file must contain one command per line. Line continuation characters (\) can be used to split commands over several physical lines in the file. The commands that can be processed in this way are: mapping file generation, running a mapping file, and even executing another batch file. For example, the command file might contain lines like:

```
c:\dx2dgn.fme --SourceDataset c:\in1.dxf --DestDataset c:\out1.dgn  
c:\dx2dgn.fme --SourceDataset c:\in2.dxf --DestDataset c:\out2.dgn
```

Command lines that generate mapping files could also be present in the command file. Notice that the word "fme" is not present as the first word of each command, as it would be if this were a DOS Batch (.BAT) file.

Mapping files run in this way should contain LOG\_FILENAME lines so that statistics on FME performance are gathered. Alternately, a LOG\_FILENAME could be specified at the end of each command line:

```
c:\dx2dgn.fme --SourceDataset c:\in1.dxf --DestDataset c:\out1.dgn LOG_FILENAME c:\fme.log  
c:\dx2dgn.fme --SourceDataset c:\in2.dxf --DestDataset c:\out2.dgn LOG_FILENAME c:\fme.log
```

An example batch run might look like this:

```
fme COMMAND_FILE c:\fmebatch.cmd
```

### **CONTINUE\_ON\_FAIL**

The CONTINUE\_ON\_FAIL optional keyword controls whether or not to continue running the translations in the command file. If its value is **no**, all the translations will fail if one translation fails.

```
fme COMMAND_FILE fmebatch.cmd CONTINUE_ON_FAIL no
```

If the keyword is not used, then all the translations will run, and a summary of the failures is output at the end.

## **Return Codes**

When FME is run as a command-line utility, it produces informational messages on its standard error output stream. When the mapping file generation or translation has completed, FME will exit with a return code of 0 if there was no error, and with -1 if there was. In general, a calling script should test for zero as a successful return code and non-zero as a failure.

Note that it is possible for FME to exit with a zero return code, indicating success even if no output was produced. A variety of situations result in this condition; for example, if a query was issued to a spatial database that returned no features, the translation would succeed even though there was no output. Software calling FME to perform translations may wish to do additional checks on the output data to ensure that something was produced.



## Generating Mapping Files

### **fme Generate**

This option is used to generate a mapping file without performing a translation. It is equivalent to using File > Generate from the FME Universal Translator user interface, and is used as follows:

```
fme Generate ...
```

**Note:** You can generate mapping files in this way; however, Safe Software recommends that you use Workbench to configure translations.

Here is some example syntax:

```
fme Generate <readerType> <writerType> <sourceDataset> <mappingFile> \  
[+DATASET <additionalSourceDataset>]* \  
[+MERGE_SCHEMAS (yes|no)] \  
[+FME_DEBUG DUMP_SCHEMA] \  
[--APPEND_LINE <text>]* \  
[--<macroName> <value>]*
```

**Note that pre-FME-2013, the line `--APPEND_LINE <text>]*` was `--APPEND_LINE <text>]`**

**Tip:** It is usually easier to customize a mapping file generated by FME or Workbench than to build a mapping file from scratch. The generated mapping file may be edited and customized for the particular translation. See *Running Custom Mapping Files*.

The following example, which should appear as one continuous line,

```
fme Generate SHAPE MIF /usr/shapedata/92b034 /tmp/s2m.fme
```

would generate a mapping file in /tmp/s2m.fme which could be used to translate all the Shape files in the /usr/shapedata/92b034 directory to MapInfo MID/MIF.

Additional source datasets of the same source format can be combined in the initial generate command to create a mapping file that will merge them all when run. The +DATASET directive is used to specify an additional dataset, and this can be repeated multiple times. In such a situation, the +MERGE\_SCHEMAS directive controls whether or not all the specified datasets are examined for schema information. If MERGE\_SCHEMAS is set to no, then only the <sourceDataset> has its schema extracted. This is used when it is known that all the input datasets have the same schema. If it is set to yes, then each of the source datasets specified have their schema extracted, and the union of these schemas is used to generate the mapping file. The default for MERGE\_SCHEMAS is no.

The +FME\_DEBUG DUMP\_SCHEMA can be added to the generate command line to have the schema information from the source dataset logged to standard output when the generate is done. This can be used to determine why a generated mapping file does not seem to have the appropriate schema information in errant circumstances.

When the --APPEND\_LINE option is used, the specified text is added at the end of the mapping file on a line by itself.

## **Performing Generic Translations**

This method of using the FME translation engine allows translations to be performed without requiring a workspace or mapping file to provide the translation rules. When used in this mode, the FME will make a "best guess" for a translation between two formats.

## GENTRANS

FME's GENTRANS mode is a simple and easy way to perform a quick translation between two formats.

GENTRANS is useful when you need to perform quick translations between different formats, and don't need to perform any special operations that might require either Workbench or a mapping file. GENTRANS performs translations that are very similar to the Universal Translator's Generate/Translate functionality.

### Command-Line Syntax

```
Fme GENTRANS [OPTIONS] <param-file>
```

```
Fme GENTRANS [OPTIONS] <param-file> <source-dataset> <dest-dataset>
```

```
Fme GENTRANS [OPTIONS] < source -format> <source-dataset> <dest-format> <dest-dataset>
```

```
Fme GENTRANS [OPTIONS] < source -format> <source-dataset> <source-directives> <dest-format> <dest-dataset> <dest-directives>
```

```
Fme GENTRANS [OPTIONS] <session-directives> < source -format> <source-dataset> <source-directives> <dest-format> <dest-dataset> <dest-directives>
```

### Description

The minimum amount of information that FME GENTRANS requires in order to function is:

- **Source/Destination Formats:** The "Short Name" as listed in the Formats Gallery.
- **Source/Destination Datasets:** Where the data should be read from, and written to.

In addition to the required configuration information, GENTRANS also allows directives to be passed directly to the FME translation session, as well as to the individual readers and writers.

The above configuration information can be passed to GENTRANS either via the command line, or via a parameter file.

As shown in the synopsis, there are several different command line forms allowed.

***Note: The Generic Translator is implemented using FME Objects – using some advanced functionality requires an understanding of FME Objects.***

The components of the command line are:

|                      |  |
|----------------------|--|
| <param-file>         | The location of the parameter file   |
| <source-dataset>     | The location of an existing dataset.   |
| <dest-dataset>       | The location where the data will be written to.<br><b>Note:</b> If the dataset exists, it will be overwritten. |
| <source-format>      | The short name of the source format.   |
| <dest-format>        | The short name of the destination format.  |
| <session-directives> | Any directives to pass to the FME session.<br><b>Note:</b> See FME Objects documentation at                    |

|                     |  |
|---------------------|--|
|                     | <a href="http://www.safe.com/download/index.php#obj_documents">www.safe.com/download/index.php#obj_documents</a>   |
| <source-directives> | Any directives to pass to the FME reader. Note: See FME Objects documentation <a href="http://www.safe.com/download/index.php#obj_documents">www.safe.com/download/index.php#obj_documents</a> . |
| <dest-directives>   | Any directives to pass to the FME writer. Note: See FME Objects documentation <a href="http://www.safe.com/download/index.php#obj_documents">www.safe.com/download/index.php#obj_documents</a> . |

**Note:** Directives are specified using a comma-separated list of strings. Where possible, it is recommended that directives not be specified on the command line, but are instead put in a parameter file.

### Options

The following options are currently supported. These options must be specified right after GENTRANS on the command line.

|                            |  |
|----------------------------|--|
| [LOG_STANDARDOUT <YES NO>] | If YES, the session will log to standard output as the translation executes.   |
| [LOG_TIMINGS <YES NO>]     | If YES, the standard FME timings will be logged.   |
| [LOG_FILENAME <filepath>]  | If specified, an FME translation log will be created at the specified location. If not specified, no log file will be created.   |
| [GUI]                      | If specified, an FME dialog will pop up, allowing the source and destination formats and data to be specified. The selected formats and data will override any formats or data specified on the command-line or in a parameter file.     |
| [GENERATE <param-file>]    | If specified, the translation will not run, and a parameter file will be generated instead.  |
| [GENERATE_CMDLINE]         | If specified, the translation will not run, and the command-line arguments will be printed to standard output. When used with the GUI option, this option allows users to determine which command-line options to use for a translation. |
| [PIPELINE <pipeline-file>] | Specifies a file containing an FME factory pipeline. Features read from the source will be processed using this pipeline before being written to the destination.  |

## Wildcard Support

Instead of exactly specifying source datasets, it is possible to use the "\*" character to specify several files at once. This is especially useful for merging several files.

For example,

```
GENTRANS SHAPE "*.shp" FFS out.ffe
```

*Note: When using wildcards, the source dataset must be enclosed in quotation marks to prevent the shell's file globbing mechanism from taking over.*

## Parameter File Format

The GENTRANS parameter file uses a simple format that is very similar to older Window's INI files. Each parameter file is broken into sections. Within each section, there are multiple key-value pairs.

### Syntax

The syntax for the parameter file is very simple. Sections begin with the specification of a section header, and run until either another section header, or the end of the file, is reached.

Each section header occurs on its own line and are of the form:

```
[ SectionName ]
```

Within each section, there are multiple key-value pairs of the form:

```
key = value
```

### Sections

The following sections are currently supported:

#### **SESSION**

Specifies session-specific configuration options:

GENTRANS-specific keys:

- GENTRANS\_PIPELINE: The name of a pipeline file.

Any other key-value pairs are assumed to be session directives and will be used to initialize the session.

#### **SOURCE**

Specifies source-specific configuration options:

GENTRANS-specific keys:

- GENTRANS\_FORMAT: The short name of the format
- GENTRANS\_DATASET: The dataset that will be read

Any other key-value pairs are assumed to be reader directives and will be passed to the reader.

#### **DESTINATION**

Specifies destination-specific configuration options:

GENTRANS-specific keys:

- GENTRANS\_FORMAT: The short name of the format
- GENTRANS\_DATASET: The dataset that will be written

Any other key-value pairs are assumed to be writer directives and will be passed to the reader.

## CONSTRAINTS

Constrain the reading to a certain subset of the dataset.

**Note: Using constraints is an advanced functionality, and assumes familiarity with FME's FME Objects API.**

This section's key-value pairs are used to construct a constraints feature that will be used with the reader's setConstraints method.

Either a bounding box, or a set of coordinates is required to construct the feature. To specify a list of coordinates, use the following keys: GENTRANS\_X\_COORDS, GENTRANS\_Y\_COORDS, GENTRANS\_Z\_COORDS. Each of these keys take a comma-separated list of floating point numbers as a value.

**Note: If a 2 dimensional constraint is desired, the GENTRANS\_Z\_COORDS key does not need to be specified.**

For example,

```
GENTRANS_X_COORDS = 0.0, 1.5, 1.5, 0.0, 0.0  
GENTRANS_Y_COORDS = 0.0, 0.0, 2.5, 2.5, 0.0
```

A bounding box is specified with the GENTRANS\_BOUNDING\_BOX key name. The value must have the following format:

```
<minx>, <maxx>, <miny>, <maxy>
```

where minx,maxx, miny, and maxy are floating point numbers.

For example,

```
GENTRANS_BOUNDING_BOX = 0.0, 2.5, 0.0, 1.5
```

In addition to the geometric constraints, a search type must also be specified using the FME\_SEARCH\_TYPE key. Values for the search type can be found in the FME Objects documentation.

For example,

```
FME_SEARCH_TYPE = FME_ENVELOPE_INTERSECTS
```

Certain search types require additional key-value pairs to be specified as parameters.

Any additional key-value pairs in this section will be passed as attributes on the constraints feature. See the FME Objects documentation for more details.

### Parameter File Generation

GENTRANS offers a GENERATE mode that provides a quick method of creating parameter files. This mode is especially useful for formats that require several directives.

To generate a parameter file, take any valid GENTRANS command-line, and add the GENERATE <parameter file> option. The translation will not run, and a new parameter file will be generated (even if another one was used).

Usage of the GUI option further increases the usefulness of this feature by allowing a user to specify format options in a dialog box.

For example,

```
GENTRANS GUI GENERATE filename.param SHAPE foo.shp MIF foo.mif
```

## Examples

### Using Coordinate System Reprojection

```
FME GENTRANS LOG_FILENAME gentrans.log ACAD drainage_design_P001.dwg COORDSYS,BritishNatGrid ACAD drainage_design2_P001.dxf  
COORDSYS,BritishNatGrid
```

### Translating with a Parameter File

#### Generating a parameter file:

```
FME GENTRANS GUI GENERATE key.param ACAD drainage_design_P001.dwg ACAD drainage_design2_P001.dxf
```

#### Using a parameter file:

```
FME GENTRANS LOG_FILENAME logfile.txt parameter_file.txt
```

### Generating a GENTRANS Command from a GUI

This example opens a GUI which outputs a GENTRANS command line. To run it, prefix it with FME GENTRANS.

```
FME GENTRANS GUI GENERATE_CMDLINE
```

### Generating a GENTRANS Command from a Parameter File

```
FME GENTRANS GENERATE_CMDLINE key.param
```

### Translating with Reader and Writer Directives

```
FME Gentrans ACAD "S:\key-systems\drainage_design_P001.dwg"  
"RUNTIME_MACROS,\  
"METAFILE,acadScan,_EXPAND_BLOCKS,no,_EXPAND_VISIBLE,no,_BULGES_AS_ARCS,no,_STORE_BULGE_INFO,yes,_READ_PAPER_SPACE,no,ACAD_IN_  
READ_GROUPS,no,_IGNORE_UCS,no,_MERGE_SCHEMAS,YES\","META_MACROS,\  
"Source_EXPAND_BLOCKS,no,Source_EXPAND_VISIBLE,no,Source_BULGES_AS_ARCS,no,Source_STORE_BULGE_INFO,yes,Source_READ_PAPER_  
SPACE,no,SourceACAD_IN_READ_GROUPS,no,Source_IGNORE_UCS,no\  
","METAFILE,acadScan,COORDSYS,BritishNatGrid,IDLIST," ACAD "S:\key-systems\drainage_design_P001.dxf" "RUNTIME_MACROS,\  
"_ATTRKIND,external_attributes,_REL,Release2000,_TMPL,S:\key-systems\drainage_design_P001.dwg\","META_MACROS,\  
"Dest_ATTRKIND,external_attributes,Dest_REL,Release2000,Dest_TMPL,S:\key-systems\drainage_design_P001.dwg\","METAFILE,ACAD,  
COORDSYS,BritishNatGrid"
```

### Using GENTRANS for Batch Processing

```
for /R x: %%F in (*.shp) do fme gentrans LOG_FILENAME "y:\%%~pF%%~nF.log" PIPELINE ground2grid.fmi SHAPE "%%F" SHAPE "y:\%%~pF"
```



## Listing Transformers

The following parameters, which are used almost exclusively for internal testing purposes, list licensed and unlicensed transformers. If the VERBOSE option is specified, all functions and factories used by each transformer will also be listed.

```
fme LIST_TRANSFORMERS [VERBOSE]  
fme LIST_UNLICENSED_TRANSFORMERS
```

## Running Custom Mapping Files

### ***fme*** <mappingFile>

This option, the most common use of the FME command-line utility, is used to translate data from one format to another. It is equivalent to using File > Run from the graphical user interface and is used as follows:

```
fme <mappingFile> [--<macroName> <value>]* [[-]<overridekeyword> <value>]* [+<addkeyword> <value>]*
```

When FME is invoked in this way, all parameters pertaining to the translation session are extracted from the mapping file specified. The additional optional arguments are used to override or supplement the contents of the mapping file.

For example,

```
fme roadgen.fme
```

fme <mappingFile>

This option, the most common use of the FME command-line utility, is used to translate data from one format to another. (See "Running Custom Mapping Files" on page 90.) It is equivalent to using File > Run from the graphical user interface and is invoked as follows:

```
fme <mappingFile> ...
```

## **Defining Macros**

Frequently, a mapping file intentionally references a macro that it does not define. It is assumed that when the mapping file is used, such macros will be given values on the command line.

Macros are defined on the command line by preceding the macro name with two dashes. The value that follows the macro name is assigned to it.

***Note: Macros specified on the command line provide only an initial value for the macro. If the macro is defined anywhere in the mapping file, the mapping file's definition overrides that given on the command line.***

In the following example, the *roadgen.fme* file contains the line:

```
SAIF_DATASET $(saifFile)
```

However, the *saifFile* macro is not defined anywhere in this mapping file. A value for the macro must be provided on the command line:

```
fme roadgen.fme --saifFile /usr/data/92j013.zip
```

### **Overriding Mapping File Settings**

Any keyword settings in the mapping file may be overridden on the command line simply by listing the keyword, optionally preceded by a dash, followed by its new value.

Any values for the keyword that are already in the mapping file will be overridden by the new value.

The following example sets the `READER_TYPE` keyword value to `SAIF` and the `WRITER_TYPE` keyword value to `SHAPE`. The values provided for `READER_TYPE` and `WRITER_TYPE` in the mapping file are ignored.

```
fme roadgen.fme -READER_TYPE SAIF -WRITER_TYPE SHAPE
```

Note that since the dashes are optional when keyword values are being overridden, the previous example is equivalent to:

```
fme roadgen.fme READER_TYPE SAIF WRITER_TYPE SHAPE
```

### **Extending Mapping File Settings**

The keyword settings in a mapping file may be extended on the command line by preceding the keyword with a plus sign (+), and listing the additional values for the keyword.

The result is the same as if the keyword and values were placed at the end of the mapping file. This is only useful for keywords accumulating their values and may be specified more than once in the mapping file. It is most commonly used to add to the `_IDs` being read by the reader module during an FME session.

The following example adds the Lanes ID to the other IDs that were requested for translation in the `roadgen.fme` mapping file. If this mapping file originally included a line stating `SAIF_IDs Roads Railroads` and the command line below was used, the SAIF Reader would process the features in the Roads, Railroads, and Lanes collections.

```
fme roadgen.fme +SAIF_IDs Lanes
```

However, if the plus sign (+) was not used and the command line below was given instead, then the `SAIF_IDs` on the command line would override those in the mapping file, and only the Lanes collection would be processed.

```
fme roadgen.fme -SAIF_IDs Lanes
```

## Reading Command-Line Parameters from a File

***fme* PARAMETER\_FILE ...**

This option reads command-line parameters from a file. Although the number of macros or keywords that can be specified on the FME command line is unlimited, some operating systems place a limit on overall command-line length.

This option serves as a workaround when such a limit is reached – it tells FME to read the command-line parameters from a file as follows:

```
fme PARAMETER_FILE ...
```

***Note: Two additional FME command-line options can be used when FME is run in server mode. These options are documented in the FME Server Administrator's Guide (available at [www.safe.com/support/onlinelearning/documentation.php](http://www.safe.com/support/onlinelearning/documentation.php)).***

## Password Protecting a Mapping File

This option is used to permanently protect a mapping file. You do not require a password: once you protect a file using this option, you cannot remove the protection.

```
fme.exe PROTECT <sourceFile> <destFile>
```

### *Using a password*

If you want to protect a file but allow access with a password, you can follow the steps in the example below:

1. Create an .fmi file (for example, password.fmi) that includes the following information:

```
MACRO password secretPass  
MACRO username secretUser
```

2. In your main mapping file, include the following statement:

```
INCLUDE password.fmi
```

3. From the command line, protect password.fmi:

```
fme PROTECT password.fmi passwordp.fmi
```

4. In the main mapping file, change the INCLUDE line to:

```
INCLUDE passwordp.fmi
```



## Executing Tcl Programs

### *fme* <TclFile>

FME can also be requested to execute Tcl (version 8.5.2) scripts, which in turn can run FME translations and perform shell commands through its command-line interface. A Tcl program is executed using the following syntax:

```
fme <TclProgramName> [<argument>]*
```

where TclProgramName must have a .tcl extension. Any additional arguments will be passed to the Tcl program in the argv list, and argc will be set to the number of additional arguments. For example, if test.tcl contains these lines:

```
puts "argc is: $argc"
puts "argv is: $argv"
foreach i $argv {
  puts $i
  fme $i.fme --SourceDataset $i.e00
}
```

the FME command-line utility is then invoked as follows:

```
fme test.tc train railway
```

And as a result, FME executes the following commands:

```
fme train.fme --SourceDataset train.e00
fme railway.fme --SourceDataset railway.e00
```

## Tcl Functions

In a Tcl script, FME provides the following Tcl functions:

### ***FME\_CommonPrefix***

Returns the longest common prefix shared by all input string arguments.

Syntax:

```
FME_CommonPrefix <arg1> <arg2> . . .
```

- All arguments are strings. Any number of strings may be passed in.
- Returns a string.

Example:

```
FME_CommonPrefix "c:/data1/bob.txt" "c:/data1/subdir2/joe.txt" returns "c:/data1/"
```

### ***FME\_RecursiveGlob***

Expands all input arguments into a single list of filenames, according to the pattern matching rules described in Reader Datasets.

Syntax:

```
FME_RecursiveGlob <arg1> <arg2> . . .
```

- All arguments are strings. Any number of strings may be passed in.
- Returns the files as a list of strings.

Example:

```
FME_RecursiveGlob "c:/files/*.shp" "c:/morefiles/file.txt" returns  
"c:/files/bob.shp" "c:/files/joe.shp" "c:/files/john.shp" "c:/morefiles/file.txt"
```

### ***FME\_TempFilename***

Generates a temporary filename in FME temporary directory. The filename is guaranteed to be a new file. The returned filename will have no extension.

Note that FME will create an empty file with the given name; you must delete it when you are done.

Syntax:

```
FME_TempFilename
```

Example:

```
FME_TempFilename
```

```
returns "c:/Temp/FME_a02724"
```

## FME User Interface Directives

FME can be used to create graphical user interfaces that greatly simplify the task of operational data translation by allowing the user to control translation parameters at run-time. This section describes the statements used to configure the FME user interface.

All FME user interface statements start with the GUI (Graphical User Interface) keyword. FME uses all the GUI statements in a mapping file to create a single dialog box that is displayed to the user. Translation begins once the user clicks OK from this dialog.

The syntax and usage of the remaining FME GUI statements are described later in this section:

- [Coordinate System Name Parameters](#)
- [Directory Parameters](#)
- [Dividers](#)
- [File Name Parameters](#)
- [Floating Point Parameters](#)
- [Informational Messages](#)
- [Integer Parameters](#)
- [Multiple-Selection List Parameters](#)
- [Password Parameters](#)
- [Single-Selection List Parameters](#)
- [Text Parameters](#)

Each GUI statement in the mapping file corresponds to one dialog element. The sequence in which the GUI statements appear within the mapping file determines the sequence in which the corresponding dialog elements are ordered on the translation dialog.

Specialized text boxes are available for restricting input to numeric values and for hiding text as it is entered. Specialized text boxes with browse buttons are available to assist users with selection of files, directories, and coordinate systems.

***Note: GUI statements that appear in external files referenced using the INCLUDE directive are not used by FME.***

All GUI statements require the user to supply a value. If a parameter is truly optional, that is, a blank value is acceptable, then the GUI statement can be flagged as such by placing the keyword OPTIONAL as the second word on the line. For example:

```
GUI OPTIONAL FLOAT Tolerance Surface Tolerance (Optional):
```

The DEFAULT\_MACRO directive can be used to supply default values to dialog elements.

For example, the following lines:

```
GUI TITLE Line Generalization Parameters
DEFAULT_MACRO SourceDir C:\Data
GUI DIRNAME SourceDir Source directory:
```

```
DEFAULT_MACRO InputFile line.dat
GUI FILENAME Input Line_Files(*.dat)|*.dat Input file:
```

```
DEFAULT_MACRO MinVertex 3
GUI INTEGER MinVertex Minimum number of vertices:
```

```
DEFAULT_MACRO MaxLengthThreshold 5.5
GUI FLOAT MaxLengthThreshold Maximum length threshold:
```

```
DEFAULT_MACRO Algorithm Douglas
GUI CHOICE Algorithm Douglas%Deveau Algorithm:
```

```
DEFAULT_MACRO CoordinateSystem LL
GUI COORDSYS CoordinateSystem Destination Coordinate System:
```

```
DEFAULT_MACRO Classes class 1 class 5
GUI LISTBOX classes class1%class2%class3%class4%class5 Classes:
```

cause FME to display a dialog that contains these elements. The title of the dialog box typically describes the type of processing performed by the mapping file and is specified using the following syntax:

```
GUI TITLE <dialogTitle>
```

The GUI TITLE statement differs from all the other GUI statements because its position in the file relative to the other GUI statements has no relevance. In other words, dialogTitle will be used as the title for the translation dialog whether the GUI TITLE statement appears as the first line in the file, the last line in the file, or anywhere in-between.

Tip: When FME runs a mapping file that has been run before, in the same session or in a previous session, the last values specified for the dialog elements take precedence over those supplied in the mapping file using the DEFAULT\_MACRO directive.

## Coordinate System Name Parameters

A parameter whose value must be chosen from the list of allowed coordinate system names is requested using the following syntax:

```
GUI COORDSYS <macroName> <label>
```

This causes the FME user interface to interact with the user to get a coordinate system name and use it to define the macro macroName. For example,

```
DEFAULT_MACRO CoordinateSystem LL  
GUI COORDSYS CoordinateSystem Destination Coordinate System:
```

requests a text box labelled Destination Coordinate System with a default value of LL. A browse button is used to display the FME Coordinate System Gallery.

After the user dismisses the Coordinate System Gallery dialog, the macro CoordinateSystem is set to their choice, which is constrained to the list of valid coordinate system names.

## Directory Parameters

Directory paths are requested using the following syntax:

```
GUI DIRNAME <macroName> <label>
```

This causes the FME user interface to interact with the user to get a directory path and use it to define the macro named by macroName. For example:

```
DEFAULT_MACRO SourceDir C:\Data  
GUI DIRNAME SourceDir Source directory:
```

requests a text box with a field labelled Source Directory, that has a value of C:\Data.

A browse button is used to display the directory browse window.

After the directory selection dialog is dismissed, the macro SourceDir is set to the directory that was input.

Tip: Use Settings > Classic File Dialogs to choose the style of directory selection dialog boxes you want to see. Choosing Classic File Dialogs (which is the default setting) can be faster in some network environments.

## Dividers

When a dialog has a large number of entries, it is useful to separate the parameters into groups. A horizontal line will be placed in the dialog when a divider is requested:

GUI DIVIDER

---

## File Name Parameters

File names are requested using the following syntax:

```
GUI FILENAME <macroName> <filter> <label>
```

This causes the FME user interface to interact with the user to get a file name and use it to define the macro named by macroName.

The filter is a standard MS-Windows filter, which follows the syntax:

```
<description>|<filter>[|<description>|<filter>]*
```

For example:

```
DEFAULT_MACRO InputFile line.dat  
GUI FILENAME InputFile Line_Files(*.dat)|*.dat Input file:
```

requests a dialog box with a field labelled Input File that has a default value of line.dat.

A browse button displays the file browse dialog.

After the file selection dialog is dismissed, the macro InputFile is set to the file name that was input.



## Floating Point Parameters

Floating point numbers are requested using the following syntax:

```
GUI FLOAT <macroName> <label>
```

For example:

```
DEFAULT_MACRO MaxLengthThreshold 5.5  
GUI FLOAT MaxLengthThreshold Maximum length threshold:
```

requests a text box labelled Maximum Length Threshold that accepts only floating point values. After the user dismisses the dialog box, the macro MaxLengthThreshold is set to the floating point value that was input.

There is no ability to restrict the range of the floating point number.

## Informational Messages

Messages may be added to the dialog to provide additional information to the end user. Messages are specified using the following syntax:

```
GUI MESSAGE <message text>
```

For example:

```
GUI MESSAGE Enter Information for the Second Autocad Input Dataset
```

## Integer Parameters

Integers are requested using the following syntax:

```
GUI INTEGER <macroName> <label>
```

For example:

```
DEFAULT_MACRO MinVertex 3  
GUI INTEGER MinVertex Minimum number of vertices:
```

requests a text box labelled Minimum number of vertices that accepts only integers.

After the user dismisses the dialog box, the macro MinVertex is set to the integer value that was input.

There is no ability to restrict the range of the integer.

## Multiple Selection List Parameters

Multiple text or numeric parameters whose values must be chosen from a list of values are requested using either of the following syntaxes:

```
GUI LISTBOX <macroName> <listItem>[%<listItem>]* <label>  
GUI CLASSICLISTBOX <macroName> <listItem>[%<listItem>]* <label>
```

For example:

```
DEFAULT_MACRO Types type1 type5  
GUI LISTBOX Types type1%type2%type3%type4%type5 Types:
```

requests a text box labelled Types with a default value of type1 type5. A browse button displays the multiple-selection list box dialog.

After you dismiss the multiple-selection list box dialog, the macro Types is set to the selected items, separated by spaces.

In older FME versions, multiple-selection list boxes were displayed on the main dialog. This effect can still be achieved by using CLASSICLISTBOX instead of LISTBOX.

For example:

```
DEFAULT_MACRO Classes class2 class3  
GUI CLASSICLISTBOX Classes class1%class2%class3%class4%class5 Classes:
```

requests a list box that displays a list of selectable items.

After a user dismisses the dialog box, the macro Classes is set to the selected items, separated by spaces.

**Tip:** If you choose not to select anything from a multiple-selection list box, it will result in an empty parameter.

## Password Parameters

Secure text parameters, such as passwords that must be obfuscated as they are typed, are requested using the following syntax:

```
GUI PASSWORD <macroName> <label>
```

For example:

```
GUI PASSWORD passw Password:
```

requests a text box labelled Password that accepts text characters. As each character is entered, the PASSWORD field displays an asterisk (\*) to keep the contents of the field from displaying. After the user dismisses the dialog box, the macro *passw* is set to their input.

## Single-Selection List Parameters

Text or numeric parameters whose value must be chosen from a list of values are requested using the following syntax:

```
GUI CHOICE <macroName> <value>[%<value>]* <label>  
GUI INT_OR_CHOICE <macroName> <value>[%<value>]* <label>  
GUI FLOAT_OR_CHOICE <macroName> <value>[%<value>]* <label>  
GUI STRING_OR_CHOICE <macroName> <value>[%<value>]* <label>
```

For example:

```
DEFAULT_MACRO Algorithm Douglas  
GUI CHOICE Algorithm Douglas%Deveau Algorithm:
```

requests a text box labelled Algorithm with a pulldown list box and the entries Douglas and Deveau. After the user dismisses the dialog, the macro Algorithm is set to the value of their choice, which is constrained to a set of allowed values. Note that if an OPTIONAL CHOICE is requested, then the pulldown list will automatically have a blank entry added to it.

```
<TYPE>_OR_CHOICE
```

allows a value of <TYPE> to be entered from the keyboard.

## Text Parameters

Text parameters are requested using the following syntax:

```
GUI TEXT <macroName> <label>
```

For example:

```
GUI TEXT mapid Mapsheet Identifier
```

requests a text box labelled Mapsheet Identifier that accepts only text parameters.

After the user dismisses the dialog box, the macro mapid is set to the text that was input.

## Reader and Writer Selection

The two most important FME keywords specify the reader and writer modules to be used during an FME session. The syntax of these keywords are:

```
READER_TYPE <readerType>  
WRITER_TYPE <writerType>
```

Any valid reader or writer module may be named following these keywords. When the FME session is started, the mapping file is scanned for these two keywords and the appropriate reader and writer are created.

The mapping file fragment below creates a SAIF reader and a Shape writer:

```
READER_TYPE SAIF  
WRITER_TYPE SHAPE
```



## Reader and Writer Selection

The two most important FME keywords specify the reader and writer modules to be used during an FME session. The syntax of these keywords are:

```
READER_TYPE <readerType>  
WRITER_TYPE <writerType>
```

Any valid reader or writer module may be named following these keywords. When the FME session is started, the mapping file is scanned for these two keywords and the appropriate reader and writer are created.

The mapping file fragment below creates a SAIF reader and a Shape writer:

```
READER_TYPE SAIF  
WRITER_TYPE SHAPE
```

## Reader and Writer Keywords

When a reader and writer are created, they scan the mapping file for additional parameters to configure their operation.

All reader and writer parameters are labeled with a two-part keyword. The prefix of the keyword is the current keyword associated with the reader or writer. The suffix identifies the configuration option. The general form of these parameters is:

```
<ReaderKeyword>_<ReaderOption> <parameter value>  
<WriterKeyword>_<WriterOption> <parameter value>
```

**Note:** The above examples should appear as a single continuous line.

By default, the <ReaderKeyword> is the same as the READER\_TYPE setting, and the <WriterKeyword> is the same as the WRITER\_TYPE setting. For example, if the READER\_TYPE and WRITER\_TYPE are configured as:

```
READER_TYPE SAIF  
WRITER_TYPE SHAPE
```

then by default the <ReaderKeyword> is SAIF and the <WriterKeyword> is SHAPE. When the SAIF reader is created, it searches the mapping file for parameters prefixed with SAIF\_ and the Shape writer searches for parameters prefixed with SHAPE\_.

Transformation specifications determine the mapping between source features and destination features. The <ReaderKeyword> starts every source line of a transformation specification and the <WriterKeyword> starts every destination line. In the example above, the transformation module scans the file for pairs of lines starting with SAIF and SHAPE, with the SAIF lines being the source portion of the transformation specification and the SHAPE lines being the destination portion.

The default values for the <ReaderKeyword> and <WriterKeyword> may be overridden by specifying values for READER\_KEYWORD and WRITER\_KEYWORD in the mapping file or on the command line. The syntax of these directives is:

```
READER_KEYWORD <newReaderKeyword>  
WRITER_KEYWORD <newWriterKeyword>
```

These directives are used most often when the READER\_TYPE and WRITER\_TYPE are the same. For example, if FME is to read an Design file, alter the geometry of some of the features, then output a new Design file, a mechanism is needed to identify the source and destination portions of the transformation specifications, as well as the source and destination data sets. The following mapping file fragment shows how to configure FME to do such a translation:

```
READER_TYPE IGDS  
WRITER_TYPE IGDS  
WRITER_KEYWORD I_OUT  
  
# Name the input IGDS file. Note that since no  
# <ReaderKeyword> was specified, the default of IGDS  
# is used  
IGDS_DATASET original.dgn  
  
# Provide the name for the output IGDS file  
I_OUT_DATASET newone.dgn
```

```
# Identify the seed file for the output IGDS file
I_OUT_SEED_FILE seed.dgn
```

```
# Now write a transformation specification. The
# source line will be labeled with the
# <ReaderKeyword> (IGDS), and the destination line
# will be labeled with the <WriterKeyword> (I_OUT)
```

```
IGDS 40 igds_color 3 igds_style %s
I_OUT 40 igds_color 4 igds_style %s \
@Generalize(Douglas,10
```

When a <ReaderKeyword> or <WriterKeyword> is specified, the reader or writer module first scans the file for its configuration options using this keyword as a prefix. If any of its required options are not found, then the configuration file is rescanned to search for the option prefixed by the default prefix which is the same as the reader or writer type. In the example above, if the line:

```
I_OUT_SEED_FILE seed.dgn
```

was given as:

```
IGDS_SEED_FILE seed.dgn
```

the end result would be the same. The Design file writer module first scans the mapping file for I\_OUT\_SEED\_FILE, because I\_OUT is the <WriterKeyword>. However, it will not find this keyword. It then rescans the file using IGDS as the <WriterKeyword> and finds a value for IGDS\_SEED\_FILE.

## Reader Datasets

The dataset a reader will extract data from is specified via the <ReaderKeyword>\_DATASET directive.

The syntax for this directive is:

```
<ReaderKeyword>_DATASET [<datasetSpecification>]+
```

One or more datasets can be specified, separated by spaces. If a dataset specification contains spaces, it must be enclosed in quotes.

When more than one dataset is listed, then FME will create a reader for each of the datasets listed, one after the other. The features read will appear to the rest of FME to have come from a single reader however. The FME's MULTI\_READER is used internally to do this merging. Therefore, when multiple datasets are listed, the features read will have the MULTI\_READER attributes added to them so that their true original source can be determined if necessary. See the Multi Reader chapter in the *FME Readers and Writers* manual.

If one of the datasets cannot be read, then it is skipped and a warning is logged. If none of the datasets can be read, an error is generated.

A <datasetSpecification> can also contain wildcards which FME will expand into a list of files or directories. If the dataset contains any of the metacharacters supported by C-shell glob style syntax, then the dataset is taken as a pattern and expanded to include all files that match.

See [Dataset Wildcards](#).

## Dataset Wildcards

The special characters supported are:

|                |  |
|----------------|--|
| ?              | Matches any single character.  |
| *              | Matches any sequence of zero or more characters.   |
| **             | Matches the current directory and recursively all subdirectories   |
| [chars]        | Matches any single character in chars. If chars contains a sequence of the form a-b then any character between a and b (inclusive) will match. |
| {ab, cd,e,...} | Matches any of the strings ab, cd, e, etc.   |

Examples:

|                           |   |
|---------------------------|---|
| C:\data\*.dgn             | expands to all files in the <b>c:\data</b> directory that end with a .dgn extension   |
| C:\data\**\*.dgn          | expands to all files in the <b>c:\data</b> directory and any subdirectory below it that ends with a .dgn extension            |
| C:\**\*.dgn               | expands to all files on the entire <b>C:</b> drive that end with a .dgn extension   |
| C:\{data,archive}\*.dgn   | expands to all files in the <b>c:\data</b> and <b>c:\archive</b> directories that end with a .dgn extension                   |
| C:\{data,archive}\92*.dgn | expands to all files in the <b>c:\data</b> and <b>c:\archive</b> directories that start with 92 and end with a .dgn extension |
| C:\data\92?034.dgn        | expands to all files in the <b>c:\data</b> directory that start with 92, have any letter or number next, and end with 034.dgn |
| C:\data\92[a-z]034.dgn    | expands to all files in the <b>c:\data</b> directory that start with 92, have any lowercase letter next, and end with 034.dgn |

## Reader Pipelines

Certain readers employ a set of factories to "massage" the features they produce into something easier to work with in a mapping file. In such cases, the factories are stored externally to the mapping file and referenced via the <ReaderKeyword>\_PIPELINE directive.

The syntax for this directive is:

```
<ReaderKeyword>_PIPELINE <pipeline file>
```

***Note: This directive is usually used by FME reader developers (but not normally used by mapping file authors). When this directive is present in a generated mapping file, it should not be altered or adjusted.***

## Reader Start/End Feature

To limit the number of features returned, you can optionally use one or both of these keywords:

```
<ReaderKeyword>_START_FEATURE_NUM <startnum>  
<ReaderKeyword>_MAX_FEATURES <maxfeatures>
```

For example, if you have a dataset with 10 features:

- If neither keyword is specified, all 10 features will be read. This is the default behavior.
- If START\_FEATURE\_NUM is specified and is greater than zero, then all features starting from feature number specified in the keyword will be read. For example,
  - If START\_FEATURE\_NUM is 5, all features from 5 to 10 inclusive will be read.
  - If START\_FEATURE\_NUM is less than or equal to zero it will always be treated as 1.
- If MAX\_FEATURES is specified and is greater than zero then features starting from feature number 1 will be read up to a maximum of MAX\_FEATURES. If MAX\_FEATURES is less than or equal to zero then all features will be read. For example,
  - If MAX\_FEATURES is 5, feature numbers 1 to 5 will be read.
  - If MAX\_FEATURES is 100, features from 1 to 10 will be read (since the dataset only has 10 features).
- If both START\_FEATURE\_NUM and MAX\_FEATURES are specified and are greater than zero, then the number of features read will depend on their values and number of features in the dataset. Here are some examples, considering the same example 10-feature dataset:

|  |   |
|--|---|
| START_FEATURE_NUM = 1<br>MAX_FEATURES = 1    | will read only the first feature                            |
| START_FEATURE_NUM = 1<br>MAX_FEATURES = -1   | all features will be read                                   |
| START_FEATURE_NUM = 1<br>MAX_FEATURES = 10   | features from 1 to 10 inclusive will be read                |
| START_FEATURE_NUM = 10<br>MAX_FEATURES = 1   | only the 10th feature will be read                          |
| START_FEATURE_NUM = 10<br>MAX_FEATURES = 100 | only the 10th feature will be read                          |
| START_FEATURE_NUM = 100<br>MAX_FEATURES = 2  | no features will be read since dataset has only 10 features |
| START_FEATURE_NUM = 6<br>MAX_FEATURES = 2    | features from 6 to 7 inclusive will be read                 |
| START_FEATURE_NUM = 6                        | features from 6 to 10 inclusive will be                     |

|   |                           |
|---|---------------------------|
| MAX_FEATURES = 100                        | read                      |
| START_FEATURE_NUM = 1<br>MAX_FEATURES = 0 | all features will be read |



## Reader Feature Types

To limit the type of features returned, you can optionally use this keyword:

```
<ReaderKeyword>_FEATURE_TYPES <ftname> .. <ftname>
```

where ft1 through n lists the feature type names that the reader will read. Some readers will use this list and efficiently read only those feature types specified while other readers will read everything and but only output the specified feature types.

The feature types themselves are specified with any special characters encoded as per the table <special characters>. This allows easy specification of feature types that contain spaces, for example.

## Log File Configuration

FME logs its configuration, progress, performance, statistics, warnings, and error messages to a log file as it executes.

If features are logged by FME they will be described in the log file, but they will also be recorded in an FFS file. This FFS file uses the same path and basename as the log file name, but it has an “.ffs” extension. This file can be opened in the FME Universal Viewer to visually inspect any features that were logged by FME.

***Note: The number of features written really means the number of features that were passed to the writer, which is not necessarily equal to the number of features that can be viewed in the FME Universal Viewer or in the destination format. What the writer does with the features (whether it splits them, or even rejects them, etc.) is dependent on the individual format. There is no reliable way of relaying that information back to the FME core, which outputs the statistics information.***

## **LOG\_FILENAME**

The file name of the log file is set in the mapping file or on the command line by supplying a value for the LOG\_FILENAME keyword. The example below sets the log file's name:

```
LOG_FILENAME /tmp/fme.log
```

If the named log file already exists, then FME appends to it by default. If no LOG\_FILENAME keyword is found in the mapping file, no session logging is performed.

Tip: After a translation, the log file should be scanned for warning messages and the statistics examined to ensure that the translation was successful. When an error occurs during translation, the log file contains detailed messages describing what happened and possibly a list of the features that caused the problem.

**LOG\_APPEND**

The flag LOG\_APPEND directs FME to either append to the log file if it existed previously or to delete it. Valid choices for this keyword are YES or NO. The default is YES.

## ***LOG\_STANDARDOUT***

To assist in building customized interfaces to FME, log information may be routed to the standard output stream. This stream may be redirected to a graphical user interface or a network connection. To configure FME to log to standard output, the LOG\_STANDARDOUT keyword must be given a value of YES:

```
LOG_STANDARDOUT YES
```

## **LOG\_MAX\_FEATURES**

To prevent the accidental creation of extremely large log files, you can control the number of FME features that may be logged during an FME session. The FME logs features found to be invalid during translation and the mapping file may request features be logged with the @Log function. If a bad input data file is encountered, or a mapping file inadvertently specifies logging the entire dataset, a log file several megabytes in size may result.

The value of the LOG\_MAX\_FEATURES keyword sets a limit on the number of features that can be logged during a translation, which by default is 20. This prevents excessively large log files from being inadvertently generated.

To remove the limit on the maximum number of loggable features, set LOG\_MAX\_FEATURES to -1. The line below sets the maximum loggable features to 50:

```
LOG_MAX_FEATURES 50
```

### ***LOG\_MAX\_RECORDED\_FEATURES***

To prevent the accidental creation of extremely large FFS log files that may be created by logging of features, you can control the number of FME features that may be logged during an FME session. The FME logs features found to be invalid during translation and the mapping file may request that features be logged with the @Log function. If a bad input data file is encountered, or a mapping file inadvertently specifies logging the entire dataset, a log file several megabytes in size may result.

The value of the LOG\_MAX\_RECORDED\_FEATURES keyword sets a limit on the number of features that can be recorded to an FFS file by logging during a translation, which by default is 20. This prevents excessively large FFS log files from being inadvertently generated.

To remove the limit on the maximum number of loggable features, set LOG\_MAX\_RECORDED\_FEATURES to -1. The line below sets the maximum loggable features to 50:

```
LOG_MAX_RECORDED_FEATURES 50
```

## **LOG\_MESSAGE\_NUMBERS**

The value of LOG\_MESSAGE\_NUMBERS indicates whether or not to output the message numbers for the logged messages. The allowed values are YES or NO.

An example output with the keyword set to YES looks like this:

```
INFORM|#231040) Mapping File Identifier is: SHAPE TO NULL  
INFORM|#231001) FME Configuration: Reader Keyword is SHAPE'
```

And if the keyword set to NO, the messages will look like this:

```
INFORM|Mapping File Identifier is: SHAPE TO NULL FME Configuration:  
INFORM|Reader Keyword is SHAPE'
```



## **FME\_BEGIN\_PYTHON and FME\_END\_PYTHON**

*Note: FME does not ship with an Python interpreter; use of FME\_BEGIN\_PYTHON and FME\_END\_PYTHON requires either Python 2.3 or Python 2.4 to be installed on the same computer as FME, and be in the system path. Python may be obtained from either <http://www.python.org> or <http://www.activestate.com>. By default, FME will attempt to load Python 2.4 first, and then Python 2.3. If you wish, you can use the FME\_PYTHON\_VERSION directive to specify a preference. For example:*

- Use FME\_PYTHON\_VERSION 2.3 to load Python 2.3*
- Use FME\_PYTHON\_VERSION 2.4 to load Python 2.4*

*Note: For examples, see [http://www.fmepedia.com/index.php/Startup\\_and\\_Shutdown\\_Script\\_Examples](http://www.fmepedia.com/index.php/Startup_and_Shutdown_Script_Examples)*

## **FME\_BEGIN\_PYTHON**

The FME\_BEGIN\_PYTHON directive specifies a Python script file to execute just prior to the start of translation. The script is executed after the mapping file has been completely parsed, and after the log file has been opened, but before any of the readers or writers have begun to do their processing.

**Note: FME will abort the translation if the execution of FME\_BEGIN\_PYTHON scripts fails. If, for some reason, this behavior is undesirable and you want to continue a translation even if the execution fails, you can use Python's exception handling to trap errors, allowing FME to continue with the translation.**

The syntax is:

```
FME_BEGIN_PYTHON <python script>
```

where the <python script> is any valid Python script file to execute.

The script may access the following global Python variables:

| <b>Global Variable</b> | <b>Contents</b>   |
|------------------------|---|
| FME_LogFileName        | <p>Allows you to write custom (user-defined) messages to the log file. For example:</p> <pre>from pyfme import *<br/>log=FMELogfile()<br/>log.log("hello")</pre> <p><b>Note: Writing custom (user-defined) messages to the log file varies depending on whether it is during the start-up or shutdown phase. In a start-up script (because FME already has the log file open, and it is dangerous to open it more than once) you should use FME_LogMessage.</b></p> |
| FME_LogMessage         | <p>The FME_LogMessage function is used to write messages to the FME log file. It may be invoked in one of two ways:</p> <pre>FME_LogMessage &lt;severity&gt; &lt;messageNumber&gt;<br/>[&lt;arg1&gt; ... &lt;argN&gt;]+</pre> <p>or</p> <pre>FME_LogMessage &lt;severity&gt; &lt;message&gt;</pre>  |

| Global Variable   | Contents  |
|-------------------|---|
|                   | <p data-bbox="670 275 1300 375">&lt;severity&gt; can have one of these values: fme_inform, fme_warn, fme_error, fme_fatal, fme_statistic, and fme_statusreport</p> <p data-bbox="670 401 1344 501">When the first form is used, the message number must be present in a file in the messages subdirectory under the FME installation directory.</p> <p data-bbox="670 506 1344 606">The remaining parameters are used to fill in any %0, %1, ... %n parameter holders in the message. For example, if the message was:</p> <p data-bbox="670 632 1149 661">3011, Opening file %0 for mode %1</p> <p data-bbox="670 686 1292 716">then FME_LogMessage could be called like this:</p> <p data-bbox="670 741 1143 808">FME_LogMessage fme_inform 3011 /tmp/cacher.txt read</p> <p data-bbox="670 833 1336 900">In the second form, the message is output directly to the log file.</p> |
| FME_MappingFileId | The value of the MAPPING_FILE_ID directive specified in the mapping file  |
| FME_MacroValues   | A Python dictionary, indexed by macro name, which holds the value of each macro known within the workspace or mapping file at the end of parsing.   |

## **FME\_END\_PYTHON**

The **FME\_END\_PYTHON** directive specifies a Python script file to execute just after translation has completed, either successfully or prematurely due to an error being encountered.

If the translation ended due to an error, the script is executed after all cleanup is done, all reader and writers are shut down, and the log file has been closed. If the translation was successful, the script is executed after all the readers and writers have finished their work, and the log file has been closed.

The script has access to a number of Python global variables that contain statistics and other information about the translation.

The script may access the following global Python variables, and any global variables set up in the **FME\_END\_PYTHON**. Note that if the translation failed, only the **FME\_Status** and **FME\_FailureMessage** will contain valid values.

| <b>Global Variable</b> | <b>Contents</b>  |
|------------------------|--|
| FME_CPUTime            | The total CPU time from just before the FME_BEGIN_PYTHON was called until just before the FME_END_PYTHON was called.                 |
| FME_CPUUserTime        | The amount of user CPU time used from just before the FME_BEGIN_PYTHON was called until just before the FME_END_PYTHON was called.   |
| FME_CPUSysTime         | The amount of system CPU time used from just before the FME_BEGIN_PYTHON was called until just before the FME_END_PYTHON was called. |
| FME_ElapsedRunTime     | The actual elapsed time used from just before the FME_BEGIN_PYTHON was called until just before the FME_END_PYTHON was called.       |
| FME_ElapsedTime        | The elapsed time from just before the FME_BEGIN_PYTHON script was called until just before the FME_END_PYTHON was called.            |
| FME_FailureMessage     | The failure message if the translation failed, blank if the translation succeeded.   |
| FME_FeaturesRead       | A Python dictionary, indexed by feature type, which holds the number of features read for that feature type.                         |
| FME_FeaturesWritten    | A Python dictionary, indexed by feature type, which holds the number of features written for that feature type.                      |

| Global Variable           | Contents  |
|---------------------------|---|
| FME_Licensing (FME 2013+) | The edition information and FME license type.   |
| FME_LogFileName           | <p>Allows you to write custom (user-defined) messages to the log file. For example:</p> <pre data-bbox="597 409 906 504">from pyfme import * log=FMELogfile() log.log("hello")</pre> <p><i>Note: Writing custom (user-defined) messages to the log file varies depending on whether it is during the start-up or shutdown phase. In a start-up script (because FME already has the log file open, and it is dangerous to open it more than once) you should use FME_LogMessage.</i></p>   |
| FME_LogMessage            | <p>The FME_LogMessage function is used to write messages to the FME log file. It may be invoked in one of two ways:</p> <pre data-bbox="597 945 1258 1018">FME_LogMessage &lt;severity&gt; &lt;messageNumber&gt; [&lt;arg1&gt; ... &lt;argN&gt;]+</pre> <p>or</p> <pre data-bbox="597 1081 1161 1123">FME_LogMessage &lt;severity&gt; &lt;message&gt;</pre> <p>&lt;severity&gt; can have one of these values:</p> <pre data-bbox="597 1186 836 1501">fme_inform fme_warn, fme_error fme_fatal fme_statistic fme_statusreport</pre> <p>When the first form is used, the message number must be present in a file in the messages subdirectory under the FME installation directory. The remaining parameters are used to fill in any %0, %1, ... %n parameter holders in the message. For example, if the message was:</p> |

| Global Variable                 | Contents  |
|---------------------------------|---|
|                                 | <p>3011, Opening file %0 for mode %1</p> <p>then FME_LogMessage could be called like this:</p> <pre data-bbox="597 373 1079 441">FME_LogMessage fme_inform 3011 /tmp/cacher.txt read</pre> <p>In the second form, the message is output directly to the log file.</p>   |
| FME_MacroValues                 | A python dictionary, indexed by macro name, which holds the value of each macro known within the workspace or mapping file at the end of parsing.   |
| FME_MappingFileId               | The value of MAPPING_FILE_ID keyword if specified in the mapping file   |
| FME_MemoryUsage (FME 2013+)     | The amount of memory being used at the end of the translation.  |
| FME_PeakMemoryUsage (FME 2013+) | The maximum amount of memory that was used at any point during the translation.   |
| FME_ProcessID (FME 2013+)       | The ID of the current process.  |
| FME_NumFeaturesLogged           | <p>This variable records the total number of requests for features to be logged. This can be helpful to detect features that a writer may have rejected.</p> <p>Note that to avoid accumulating hundreds of features in the log, FME by default suppresses all log attempts after the first 20. Therefore, the actual number of features logged may be fewer than the number requested.</p> |
| FME_Status                      | 0 if the translation failed and 1 if it was successful  |
| FME_TotalFeaturesRead           | The total number of features read   |
| FME_TotalFeaturesWritten        | The total number of features written  |
| FME_TotalCoordinates            | The total number of coordinates output  |

## FME\_BEGIN\_TCL and FME\_END\_TCL

### *Translation Begin/End Hooks*

When the FME translation engine is incorporated into larger systems, it may be useful to have FME perform some additional site-specific custom processing either before or after each translation.

Users can specify multiple FME\_BEGIN\_TCL and FME\_END\_TCL directives. Errors encountered during the execution of the begin/end Tcl scripts will fail the translation.

The FME\_BEGIN\_TCL and FME\_END\_TCL mapping file directives specify Tcl scripts to be run either before or after (respectively) the actual translation. The FME\_END\_TCL script has access to a number of variables which contain the statistics and status of the translation, and as such can be used to write customized logs or insert records into a database (perhaps using the TclODBC or Oratcl Tcl extensions - see examples 6 and 7 in [Tcl Examples](#), respectively) to make a record of the translation's activity.

The FME\_END\_TCL script can also read through the log file produced by FME to extract certain messages and record those results as well.

The FME\_BEGIN\_TCL and FME\_END\_TCL scripts will share a Tcl interpreter, which means that the FME\_BEGIN\_TCL script can set up some Tcl global variables that will later be available to the FME\_END\_TCL script. However, in most scenarios it is likely that only an FME\_END\_TCL script will be needed. Lastly, the FME\_BEGIN\_TCL and FME\_END\_TCL interpreter is completely separate from any other Tcl interpreters used in FME; for example, the @TCL and @Tcl2 FME functions do not share this interpreter.

The scripts to be executed for FME\_BEGIN\_TCL or FME\_END\_TCL can be specified right in the mapping file, or they can be "source'd" in from an external file. If they are specified in the mapping file, then any quotation marks must be escaped with a \ and continuation characters must be used to join all the lines in the script together into one logical FME mapping file line. This also then requires that the ; statement separator be used in the Tcl code.

Alternately, if the Tcl script is placed in an external file and source'd in, then native Tcl quoting and formatting may be used. For example:

```
FME_END_TCL proc finalize {} { \
    ... Tcl code here ... \
}; \
finalize; \
or
FME_END_TCL source $(FME_MF_DIR_UNIX)/finalization.tcl
```

Note that if the script contains a procedure definition, the script must call the procedure it defined, otherwise the procedure itself will not be executed and no processing will occur.

If any errors are encountered during the processing of either the FME\_BEGIN\_TCL or FME\_END\_TCL script, the errors are written into the log file. If there is no log file, the errors will not be reported. Any errors raised by the Tcl script will be up propagated to the FME translator and will cause the translation to terminate with an error code, so that external callers of the FME translator can take appropriate action.

These hooks are not available within the definition of a custom format. They can, of course, be used in a mapping file or workspace that itself employs a custom format.

The FME Objects Tcl interface may not be used within either of the FME\_BEGIN\_TCL or FME\_END\_TCL scripts. This is because using FME Objects at the translation start/end times would cause a configuration conflict within the

FME translation system. However, the FME\_BEGIN\_TCL/FME\_END\_TCL scripts can themselves start new processes via the "exec" Tcl command, and these new processes could involve FME Objects Tcl without any conflict.

***Note: For examples, see <http://www.fmepedia.com/index.php> Startup\_and\_Shutdown\_Script\_Examples***



## **FME\_BEGIN\_TCL**

The FME\_BEGIN\_TCL directive specifies a Tcl script to execute just prior to the start of translation. The script is executed after the mapping file has been completely parsed, and after the log file has been opened, but before any of the readers or writers have begun to do their processing.

**Note: FME will abort the translation if the execution of FME\_BEGIN\_TCL scripts fails. If, for some reason, this behavior is undesirable and you want to continue a translation even if the execution fails, you can add a Tcl catch command in your Tcl script – this will catch any errors and FME will continue with the translation.**

The syntax is:

```
FME_BEGIN_TCL <Tcl script>
```

where the <Tcl script> is any valid Tcl script to execute, or

```
FME_BEGIN_TCL FME_Decode <encoded Tcl script>
```

which allows for the script to be encoded in an internal FME format produced by Workbench.

The script may access the following global Tcl variable:

| <b>Global Variable</b> | <b>Contents</b>   |
|------------------------|---|
| FME_LogFileName        | <p>Allows you to write custom (user-defined) messages to the log file.</p> <p><b>Note: Writing custom (user-defined) messages to the log file varies depending on whether it is during the start-up or shutdown phase. In a start-up script (because FME already has the log file open, and it is dangerous to open it more than once) you should use FME_LogMessage.</b></p>                         |
| FME_LogMessage         | <p>The FME_LogMessage function is used to write messages to the FME log file. It may be invoked in one of two ways:</p> <pre>FME_LogMessage &lt;severity&gt; &lt;messageNumber&gt; [&lt;arg1&gt; ... &lt;argN&gt;]+</pre> <p>or</p> <pre>FME_LogMessage &lt;severity&gt; &lt;message&gt;</pre> <p>&lt;severity&gt; can have one of these values: fme_inform, fme_warn, fme_error, fme_fatal, fme_</p> |

| Global Variable               | Contents   |
|-------------------------------|--|
|                               | <p>statistic, and fme_statusreport</p> <p>When the first form is used, the message number must be present in a file in the messages subdirectory under the FME installation directory. The remaining parameters are used to fill in any %0, %1, ... %n parameter holders in the message. For example, if the message was:</p> <p>3011, Opening file %0 for mode %1</p> <p>then FME_LogMessage could be called like this:</p> <pre>FME_LogMessage fme_inform 3011 /tmp/cacher.txt read</pre> <p>In the second form, the message is output directly to the log file.</p> |
| FME_MacroValues (<macroname>) | Each element in this array, indexed by macro name, holds the number of each macro known to the workspace or mapping file at the end of parsing.  |
| FME_MappingFileId             | The value of the MAPPING_FILE_ID directive specified in the mapping file   |

## **FME\_END\_TCL**

The FME\_END\_TCL directive specifies a Tcl script to execute just after translation has completed, either successfully or prematurely due to an error being encountered.

If FME\_END\_TCL is followed by the FME\_Decode token, the script is encoded in an internal FME format, as mentioned for FME\_BEGIN\_TCL.

If the translation ended due to an error, the script is executed after all cleanup is done, all reader and writers are shut down, and the log file has been closed. If the translation was successful, the script is executed after all the readers and writers have finished their work, and the log file has been closed.

The script may access the following global Tcl variables, and any global variables set up in the FME\_BEGIN\_TCL. Note that if the translation failed, only the FME\_Status and FME\_FailureMessage will hold any values.

| <b>Global Variable</b>             | <b>Contents</b>  |
|------------------------------------|--|
| FME_CPUTime                        | The total CPU time from just before the FME_BEGIN_TCL was called until just before the FME_END_TCL was called.                 |
| FME_CPUUserTime                    | The amount of user CPU time used from just before the FME_BEGIN_TCL was called until just before the FME_END_TCL was called.   |
| FME_CPUSysTime                     | The amount of system CPU time used from just before the FME_BEGIN_TCL was called until just before the FME_END_TCL was called. |
| FME_ElapsedRunTime                 | The actual elapsed time used from just before the FME_BEGIN_TCL was called until just before the FME_END_TCL was called.       |
| FME_ElapsedTime                    | The elapsed time from just before the FME_BEGIN_TCL script was called until just before the FME_END_TCL was called.            |
| FME_EndingSeconds                  | The seconds since epoch when the translation ended (as returned by [clock seconds])  |
| FME_EndingTimeStamp                | The ending time of the translation, formatted as YYYY-MM-DD HH:MM:SS   |
| FME_FailureMessage                 | The failure message if the translation failed, blank if the translation succeeded  |
| FME_FeaturesRead(<featureType>)    | Each element in this array, indexed by feature type, holds the number of features read for that feature type.                  |
| FME_FeaturesWritten(<featureType>) | Each element in this array, indexed by feature type, holds the number of features written for that feature type.               |

| Global Variable                 | Contents  |
|---------------------------------|---|
| FME_Licensing (FME 2013+)       | The edition information and FME license type.   |
| FME_LogFileName                 | <p>Allows you to write custom (user-defined) messages to the log file. For example:</p> <pre data-bbox="613 415 1453 604">FME_END_TCL set outputFile [open \$FME_LogFileName a] ; \ puts \$outputFile {}; \ puts \$outputFile {writing some stuff at the end}; \ \ close \$outputFile;</pre>  |
| FME_MacroValues(<macroname>)    | Each element in this array, indexed by macro name, holds the number of each macro known to the workspace or mapping file at the end of parsing.   |
| FME_MappingFileId               | The value of MAPPING_FILE_ID keyword if specified in the mapping file.  |
| FME_MemoryUsage (FME 2013+)     | The amount of memory being used at the end of the translation.  |
| FME_NumFeaturesLogged           | <p>This variable records the total number of requests for features to be logged. This can be helpful to detect features that a writer may have rejected.</p> <p>Note that to avoid accumulating hundreds of features in the log, FME by default suppresses all log attempts after the first 20. Therefore, the actual number of features logged may be fewer than the number requested.</p> |
| FME_PeakMemoryUsage (FME 2013+) | The maximum amount of memory that was used at any point during the translation.   |
| FME_StartingSeconds             | The seconds since epoch when the translation was started (as returned by [clock seconds])   |
| FME_StartingTimeStamp           | The starting time of the translation, formatted as YYYY-MM-DD HH:MM:SS  |
| FME_Status                      | 0 if the translation failed and 1 if it was successful  |
| FME_TotalCoordinates            | The total number of coordinates output  |
| FME_TotalFeaturesRead           | The total number of features read   |
| FME_TotalFeaturesWritten        | The total number of features written  |

## Tcl Examples

### Example 1:

When placed into a complete mapping file, this mapping file fragment uses the translation begin/end hooks to cause the translation start and end times to be written to standard output.

```
FME_BEGIN_TCL set gTranslationStartTime [clock format [clock seconds]]
FME_END_TCL puts "\"Translation started time: $gTranslationStartTime \nTranslation end time: [clock format [clock seconds]]\""
```

### Example 2:

When placed into a complete mapping file, this mapping file fragment uses the translation end hook and the global FME\_Status variable to output a translation success or failure message.

Notice that it first defines a procedure which will do this testing and output, and then it actually calls the procedure to do the work.

```
FME_END_TCL proc finally {} { \
    global FME_Status; \
    if {$FME_Status == "1"} { \
        puts "\"Translation was successful\""; \
    } else { \
        puts "\"Translation was NOT successful\""; \
    }; \
}; \
finally
```

This same example could also be written without using a procedure:

```
FME_END_TCL if {$FME_Status == "1"} { \
    puts "\"Translation was successful\""; \
} else { \
    puts "\"Translation was NOT successful\""; \
}
```

### Example 3:

This example shows how macros used by the mapping file can be passed into a beginning or ending script as an argument to a procedure. This could be used to move, copy, e-mail, backup or otherwise post/pre-process the destination datasets. In this example, the destination dataset is backed up prior to the start of translation to save any existing output file that would normally be overwritten.

The mapping file fragment is:

```
# The macro being used in this mapping file to set the destination
# dataset directory is DestDataset.
DWG_DATASET "$(DestDataset)"
# Source in the Tcl script to run at the conclusion of this
# translation. The script is stored in the same directory as
# this mapping file.
FME_BEGIN_TCL source {$(FME_MF_DIR_UNIX)/backup.tcl}; backup {$(DestDataset)}
```

The contents of the "backup.tcl" file are:

```
proc backup {filename} {
    if {[file exists $filename]} {
        file copy -force $filename $filename.bak
    }
}
```

#### Example 4:

This example shows how the FME\_END\_TCL could be used to chain together translations, based on the results of a particular translation. In this example, if the original translation fails, then another fme session will be initiated.

Note that in the Tcl file, the 2> NUL: routes any log and error messages to the null device, which effectively causes them to be ignored.

The mapping file fragment is:

```
# Source in the Tcl script to run at the conclusion of this
# translation. The script is stored in the same directory as
# this mapping file.
FME_END_TCL source {$(FME_MF_DIR_UNIX)/tryAnother.tcl}
```

The contents of the "tryAnother.tcl" file are:

```
set outputFile [open c:/temp/status.txt w+]
if { $FME_Status == "1" } {
    puts $outputFile "Translation was successful"
} else {
    puts $outputFile "Translation failed -- running alternate translation"
    exec fme.exe alternateTranslation.fme 2> NUL:
}
close $outputFile
```

#### Example 5:

This example uses an external script to output a complete set of translation statistics to a summary file at the end of translation.

The mapping file fragment is:

```
# Set a mapping file id so that the Tcl finalization procedure "knows" which
# mapping file was being run
MAPPING_FILE_ID Shape to AutoCAD
# Log all the message numbers so that we can later pull out only those
# we are interested in.
LOG_MESSAGE_NUMBERS yes
# Source in the Tcl script to run at the conclusion of this
# translation. The script is stored in the same directory as
# this mapping file.
FME_END_TCL source {$(FME_MF_DIR_UNIX)/tclFinalization.tcl}
```

The contents of the "tclFinalization.tcl" file are:

```
# Open a file for writing out the translation stats
set outputFile [open c:/temp/stats_out.txt w+]
# Check for the translation status and output the desired message
if { $FME_Status == "1" } {
    puts $outputFile "Translation was successful"
} else {
    puts $outputFile "Translation failed"
    puts $outputFile "Error message was: $FME_FailureMessage"
};
puts $outputFile ""
# Output the unique Mapping file identifier set in the mapping file using MAPPING_FILE_ID
puts $outputFile "Translation summary for $FME_MappingFileId"
# Output the number of features read\written per feature type.
puts $outputFile "-----"
puts $outputFile "                          Features read summary"
puts $outputFile "-----"
# Loop through a sorted listed of the feature types that were read, and output the
# count for each one
set formatSpec "%-65s: %12s"
foreach featType [lsort [array names FME_FeaturesRead]] {
```

```

    puts $outputFile [format $formatSpec $featType $FME_FeaturesRead($featType)]
}
puts $outputFile "-----"
puts $outputFile [format $formatSpec "Total Features Read" $FME_TotalFeaturesRead];
puts $outputFile "-----"
puts $outputFile ""
puts $outputFile "-----"
puts $outputFile "                Features Written summary                "
puts $outputFile "-----"
# Loop through a sorted listed of the feature types that were written, and output the
# count for each one
foreach featType [lsort [array names FME_FeaturesWritten]] {
    puts $outputFile [format $formatSpec $featType $FME_FeaturesWritten($featType)]
}
puts $outputFile ""
puts $outputFile "-----"
puts $outputFile [format $formatSpec "Total Features Written" $FME_TotalFeaturesWritten]
puts $outputFile [format $formatSpec "Total Coordinates Written" $FME_TotalCoordinates]
puts $outputFile "-----"
puts $outputFile ""
# Look for any lines in the logfile that were warnings, and output a count of them to
# the summary file. Also, check if there was any "unexpected input remover"
# statistics line and report a non-zero count if there was (this may happen
# when workbench generated mapping files are run against input datasets
# with different feature types than those that were expected)
# And also fish out the system status log message (which is #246014) and copy
# it into the output file
set logfileExists [file exists $FME_LogFileName]
set warnings 0
if {$logfileExists} {
    set logfile [open $FME_LogFileName r]
    while {[gets $logfile line] >= 0} {
        if {[regexp {WARN} $line]} {
            incr warnings
        } elseif {[regexp {#246014} $line]} {
            puts $outputFile "-----"
            puts $outputFile $line
            puts $outputFile "-----"
        } elseif {[regexp {Unexpected Input Remover} $line]} {
            set totalFeatures 0
            set acceptedFeatures 0
            set rejectedFeatures 0
            set line [regsub {^.*Unexpected} $line {Unexpected}]
            catch {scan $line "Unexpected Input Remover(TestFactory): Tested %d input features -- %d features passed, %d
features failed." totalFeatures acceptedFeatures rejectedFeatures}
            if {$rejectedFeatures > 0} {
                puts $outputFile "-----"
                puts $outputFile [format $formatSpec "Features with Unexpected Feature Types" $rejectedFeatures]
                puts $outputFile "-----"
                puts $outputFile ""
            }
        }
    }
}
close $logfile
}
puts $outputFile "-----"
puts $outputFile [format $formatSpec "Logfile $FME_LogFileName warnings" $warnings]
puts $outputFile "-----"
puts $outputFile ""
puts $outputFile "-----"
puts $outputFile [format $formatSpec "Total Elapsed Time (seconds)" $FME_ElapsedTime]
puts $outputFile [format $formatSpec "Total CPU Time (seconds)" $FME_CPTime]
puts $outputFile "-----"
puts $outputFile ""
# And close the file
close $outputFile

```

## Example 6: Installing TclODBC

This example uses an external script to insert a record of the translation's activity into a database using the TclODBC package. In this example, an Access database is created (if necessary) and populated with a row for each translation that completes.

To use TclODBC within the FME environment, follow these steps:

Read about the TclODBC package at <http://wiki.tcl.tk/2151>

Download the TclODBC package from <http://sourceforge.net/projects/tclodbc>

Unzip the result

Edit the supplied setup.tcl to change:

```
to
    if [string match *lib $i] {
    if [string match *lib* $i] {
```

In a command prompt in the directory holding setup.tcl, run "fme setup.tcl". This will install the TclODBC package in the FME's Tcl environment.

Ensure any scripts that require the TclODBC package start with:

```
package require tclodbc
```

To invoke the sample script below which does the database insertion, the mapping file contains this line:

```
FME_END_TCL source {$(FME_MF_DIR_UNIX)/recordTranslationODBC.tcl}

The contents of the "recordTranslationODBC.tcl" file are:
=====
#
# recordTranslationODBC.tcl
#
# This script records the execution of a translation using
# the TclODBC package. This example includes the creation of the
# Access database and table for storing the translation results.
# One row for each translation run is inserted into the table.
# Note that for an actual production system the "puts" statements
# would be removed.
package require tclodbc
=====
#
# Set up some variables that are used within to create and connect
# to the database via ODBC
set driver "Microsoft Access Driver (*.mdb)"
set dbfile c:/translations.mdb
set dsn XLATION
=====
# Create the database if it isn't already there. We wouldn't do this if the
# database was a server-based one, but for this example we're just using Access.
if ![file exists $dbfile] {
    puts "Creating database $dbfile"
    database configure config_dsn $driver [list "CREATE_DB=\"$dbfile\" General"]
} else {
    puts "Using existing database $dbfile"
}
=====
# Make an ODBC datasource for this database, and connect to it
# First always remove the DSN if it was there already. If we were
# using ODBC to connect to a "real" database, then we'd just assume
# the DSN is already valid
catch {database configure remove_dsn $driver "DSN=$dsn"}
database configure add_dsn $driver [list "DSN=$dsn" "DBQ=$dbfile"]
```



```

database db $dsn
#=====
# Create the table we want to insert into if it wasn't there already
if {[length [db tables XLATION_RESULTS]] == 0} {
  puts "Creating XLATION_RESULTS table in database $dbfile"
  db "CREATE TABLE XLATION_RESULTS (
    MappingFileID VARCHAR(50),
    StartTime TIMESTAMP,
    EndTime TIMESTAMP,
    CpuTime DOUBLE,
    Successful CHAR(3),
    NumFeatures INTEGER)"
} else {
  puts "XLATION_RESULTS table present in database $dbfile"
}
#=====
# All of that was just setup, now we actually can insert our row,
# commit it, and disconnect from the datasource
set Success yes
if {$FME_Status == 0} {set Success no}
db "INSERT INTO XLATION_RESULTS
  (MappingFileID, StartTime, EndTime, CpuTime, Successful, NumFeatures)
  VALUES ('$FME_MappingFileId', \{ts '$FME_StartingTimeStamp'\},
    \{ts '$FME_EndingTimeStamp'\},
    $FME_CPUtime, '$Success', $FME_TotalFeaturesWritten)"

db commit
db disconnect
#=====
# If you were connecting to a server-based database, you probably
# would NOT remove the ODBC DSN at the end. But since we were file based
# we will remove it
database configure remove_dsn $driver "DSN=$dsn"

```

### Example 7: Installing Oratcl

This example uses an external script to insert a record of the translation's activity into an Oracle database using the Oratcl package.

To use Oratcl within the FME environment, follow these steps:

Install and configure Oracle client software on your computer. Verify you can access your database using sqlPlus.

Read about and download the Oratcl package at <http://oratcl.sourceforge.net>.

Unzip the result into \$FME\_HOME/tcl\_library/oratcl4.1 (you will have to create this directory, and the 4.1 may need to change to match version of Oratcl you've downloaded). After unzipping, move the msvcr70.dll and oratcl41.dll to \$FME\_HOME/tcl\_library (up one level).

Ensure any scripts which require the Oratcl package start with:

```
package require Oratcl
```

(It is important that the "O" in Oratcl is uppercase.)

To invoke the sample script below script which does the database insertion, the mapping file contains this line:

```
FME_END_TCL source {$(FME_MF_DIR_UNIX)/recordTranslationOracle.tcl}
```

The contents of the "recordTranslationOracle.tcl" file are:

```

#=====
#
# recordTranslationOracle.tcl
#

```

```

# This script records the execution of a translation using
# the Oratcl package. This example includes the creation of the
# XLATION_RESULTS table for storing the translation results.
# One row for each translation run is inserted into the table.
package require oratcl
#=====
# Login to the Oracle service
set username scott
set password tiger
set service amidala
set loginHandle [oralogon $username/$password@$service]
#=====
# Determine if the xlation_results table we wish to record results to exists
set tableExists no
set statementHandle [oraopen $loginHandle]
orasql $statementHandle "select * from user_tables where table_name = 'XLATION_RESULTS'"
while {[orafetch $statementHandle -datavariabale row] == 0} {
    set tableExists yes
}
#=====
# Create the xlation_results table if we need to
if {$tableExists == "no"} {
    orasql $statementHandle "CREATE TABLE XLATION_RESULTS (
        MappingFileID VARCHAR(50),
        StartTime TIMESTAMP,
        EndTime TIMESTAMP,
        CpuTime FLOAT,
        XlationSuccessful CHAR(3),
        NumFeatures INTEGER)"
}
#=====
# Insert the translation results into the table
set Success yes
if {$FME_Status == 0} {set Success no}
orasql $statementHandle "INSERT INTO XLATION_RESULTS
    (MappingFileID, StartTime, EndTime,
    CpuTime, XlationSuccessful, NumFeatures)
    VALUES ('$FME_MappingFileId',
    timestamp '$FME_StartingTimeStamp',
    timestamp '$FME_EndingTimeStamp',
    $FME_CPUTime, '$Success',
    $FME_TotalFeaturesWritten)" -commit
#=====
# Shut down the statement handle
oraclose $statementHandle
#=====
# Logout of our oracle service
oralogoff $loginHandle

```

## **FME\_JVM\_MIN\_HEAP\_SIZE and FME\_JVM\_MAX\_HEAP\_SIZE**

Through the environment variables FME\_JVM\_MIN\_HEAP\_SIZE and FME\_JVM\_MAX\_HEAP\_SIZE, users can customize the initial and maximum heap size for initializing Java VM. These settings are for FME Plug-ins written in Java.

By default, the initial heap size is 1024K and the maximum is 16384K. JVM will take the default values if these two variables are not set.

The values of these two variables must be multiples of 1024K (for example, 4M and 64M or 4096K and 32768K).

## **FME\_SHARED\_RESOURCE\_DIR**

Through the environment variable FME\_SHARED\_RESOURCE\_DIR, FME can share resource directories that contain the following subdirectories: Transformers, Formats, Coordinate Systems, and Transformer Categories.

## **Windows**

- FME\_SHARED\_RESOURCE\_DIR can point to a shared resource directory. For example, FME\_SHARED\_RESOURCE\_DIR=c:\temp\fred;c:\temp\wilma
- The Registry entry *HKEY\_CURRENT\_USER/Software/Safe Software Inc./Feature Manipulation Engine/Shared Resource Directory* can also point to a shared resource directory.

## ***Linux/UNIX***

- FME\_SHARED\_RESOURCE\_DIR must be set to run workspaces that reference custom transformers, custom formats, and custom coordinate systems. For example, export FME\_SHARED\_RESOURCE\_DIR=~ / fred ; ~ / wilma

## **FME\_STROKE\_MAX\_DEVIATION**

The stroking of arcs can be controlled by this directive, which defines the maximum distance between the arc feature and its stroked linear feature. If this directive exists, an arc feature will be stroked into a linear feature with the smallest number of points, such that the maximum distance between the arc and the stroked line is less than or equal to the value of this directive.

This directive defaults to 0 if not specified. If the value of this directive is smaller than or equal to 0, arc features will be stroked into linear features such that the degree per edge is roughly the value of FME\_ARC\_DEGREES\_PER\_EDGE which defaults to 5. (See the @Arc function in the *FME Functions and Factories* manual for more information on FME\_ARC\_DEGREES\_PER\_EDGE.) If the value is greater than one of the primary or secondary radius, arc features will be stroked into linear features containing 4 points or less.

## Geometry Handling

### **FME\_GEOMETRY\_HANDLING**

#### **Notes:**

*Before the release of FME 2009, this directive was called FME\_USE\_RICH\_GEOMETRY. It has been renamed to more accurately reflect its function in FME.*

*It is important to understand that, for the most part, the addition of the enhanced geometry model will be transparent to users. You will not have to change the way you use FME or set up a mapping file. The only visible difference should be improved output (if there was room for improvement).*

Enhanced geometry provides FME with the capability to handle very complex geometry types, such as polygon features whose boundaries include arc features. It also provides the ability to truly hold measures.

The FME\_GEOMETRY\_HANDLING global directive is used by FME to signal whether to produce improved geometric results, where possible.

The directive only sets the enhanced geometry mode in readers and factories/functions/transformers that were built before the inception of the enhanced geometry model in FME 2006. Most readers, factories, functions and transformers developed after FME 2006 were built using the enhanced geometry model and hence are not affected by this global directive. In addition, the directive does not guarantee that all features output by a reader/function/factory/transformer will be enhanced; only where it is beneficial. For example, the Shape Reader will only output enhanced geometries if the source shapefile contains measures.

Writing of enhanced geometry is handled automatically by all writers. Those writers with native enhanced geometry support will take advantage of the more complex geometries that can be represented, and will maintain, within the constraints of the format, the geometry passed to it. Writers that support only classic geometry will automatically convert any enhanced geometry into classic geometry. This means any geometry (i.e., enhanced or classic) can be sent to any writer. To fully take advantage of writers with enhanced geometry support, features with enhanced geometry must be passed to them. The simplest way to accomplish this is to use a reader developed after FME 2006 and/or set the FME\_GEOMETRY\_HANDLING directive to YES.

The default for this directive is NO. To use the enhanced geometry mode, set the directive to YES. For example:

```
FME_GEOMETRY_HANDLING YES
```

If the directive is set to NO or is missing, the mapping file/workspace/FME Objects application will use classic geometry mode. This means existing workspaces/mapping files/FME Objects applications will produce the same results as they did before the introduction of the enhanced geometry mode.

In Workbench and the Viewer, the FME\_GEOMETRY\_HANDLING option is called Geometry Handling, and valid values for this parameter are Enhanced and Classic. To set this in Workbench, go to the Navigator pane, choose Advanced Settings > Geometry Handling. In the Viewer, select View > Options > Geometry Handling. The default mode is Classic in Workbench and Enhanced in the Viewer.

See the documentation of applicable readers, factories, functions and transformers to find more details on how their behavior will change when the geometry mode is set to Enhanced.

***For more information on the differences between enhanced geometry and classic geometry handling, visit [www.fmepedia.com](http://www.fmepedia.com) and search Geometry Model.***



## Temporary Directory Determination

FME puts many temporary files in the system temporary directory, which on some Windows versions defaults to C:\temp. If there is not enough room in this directory due to space limitations on the C drive, you can override this default and send the FME temporary files to a different directory using the FME\_TEMP environment variable. (In Windows, look under **Control Panel | System | Advanced | Environment Variables**).

FME searches for a temp directory using the following tests in order:

1. If the user environment variable FME\_TEMP exists, use its value as the temporary directory path.
2. If the user environment variable TEMP exists, use its value as the temporary directory path.
3. If a directory called \temp exists on the drive on which FME is running, use that directory.
4. If a directory called \tmp exists on the drive on which FME is running, use that directory.

If all of these tests fail to locate an existing temporary directory, FME aborts with an error message indicating that it couldn't open a temporary file. To remedy this situation, either:

Create an <FMEdrive>:\temp or <FMEdrive>:\tmp directory (where <FMEdrive> is the drive on which FME is running).

OR

Define the FME\_TEMP or TEMP user environment variable, either system-wide or for the user account on which FME is running. For example, if the temporary files are to be placed into the directory j:\scratch, then the environment variable FME\_TEMP would be set to j:\scratch before FME is run.

## ***Java VM Loading***

These settings are for FME Plug-ins written in Java.

These environment variables allow you to specify memory available to Java Plug-ins:

- **FME\_JVM\_MIN\_HEAP\_SIZE**: customize the initial heap size for initializing Java VM.
- **FME\_JVM\_MAX\_HEAP\_SIZE**: customize the maximum heap size for initializing Java VM.

By default, the initial heap size is 1024K and the maximum is 16384K. JVM will take the default values if these two variables are not set.

The values of these two variables must be multiplies of 1024K, for example, they can be 4M and 64M or 4096K and 32768K.

## Mapping File Debugging

FME provides several facilities that may be enabled to assist in debugging mapping files. These debugging facilities are useful for finding either errors in a custom mapping file being developed, or errors in input data. Each debugging option causes diagnostic information to be output to the log file.

The debug options are enabled by listing the desired debug facilities on one or more FME\_DEBUG lines in a mapping file. The syntax of these lines is:

```
FME_DEBUG <option1> [<optionN>]*
```

The valid debugging options and their actions are listed below.

### Valid Debugging Options and Related Actions

| Option        | Action  |
|---------------|---|
| BADNEWS       | This is a debugging option useful only to Safe Software developers. When this is activated, FME will log the location and message of any internal software exception.   |
| DonutFactory  | This activates warning output from the DonutFactory. Points and polygons contained by more than one polygon are logged, as are polygons containing more than one point. This is used to track down problems with extra points or overlapping polygons in the original source data set.  |
| DUMP_SCHEMA   | This option is usually only used on the mapping file generation command line, and is used to cause FME to log the schema information it reads from the source dataset(s). This can be used to determine why a mapping file is not being generated in the expected way, usually to troubleshoot problems with the generation parameters or the source reader. It is also often used by developers using the plug-in SDK to check if their <i>readSchema</i> implementation is correct. |
| FACTORY       | Each factory definition line is logged as the factory is activated. This can be used to ensure that the factories intended to operate are correctly created and configured.   |
| FACTORY_TRACE | This debugging option causes each feature to have the attribute <code>fme_factory_trace</code> added to it. This attribute contains an ordered list of all factories the feature passed through. The <code>@Log()</code> function may be used in conjunction with this option causing this attribute's value to be logged to allow inspection.  |

| Option                 | Action  |
|------------------------|---|
| FME_END_TCL            | This debugging option will log additional diagnostic messages in case, for some reason, the <b>FME_END_TCL</b> directive fails.   |
| FME_STACK_TRACE        | This is a debugging option useful to Safe Software developers only. It provides additional information for developers, but works only in conjunction with the BADNEWS option.   |
| HTTP_VERBOSE           | This logs additional output for the HTTPFactory when using HTTP_VERBOSE, such as a progress counter for the data fetched, and additional header information.  |
| HTTP_DEBUG             | This logs all information provided by HTTP_VERBOSE in the HTTPFactory, with additional debugging information, such as details regarding the HTTP request and response connection information.   |
| MAPPING_FILE           | Every line of the mapping file is logged to the log file after all blanks and comments have been removed, continuation characters have been taken into account, and macros have been expanded. A histogram of the keywords encountered in the mapping file is also listed and used to view the mapping file as FME actually sees it. The keyword histogram can also be used to discover missing continuation characters if it reports extra incorrect keywords. |
| PolygonDissolveFactory | This activates additional statistical output from the PolygonDissolveFactory. When a GROUP_BY clause is used with this factory, setting this debug option causes the number of features input and output from each group to be logged.  |
| ReferenceFactory       | This activates additional statistical output information for the ReferenceFactory. A table showing the number of features processed by the ReferenceFactory is output when this is activated.   |
| Relate                 | This option causes each row read by an @Relate query to be logged to the log file.  |
| TRANSFER_PROCESS       | This is a very expensive debugging option that examines the transformation process. It reports any attributes of source features that were ignored or referenced but not defined during the transformation process. Because it is done on each feature and may produce a great deal of output in the log file, it should only be done while testing mapping files on small input datasets.  |

| Option         | Action   |
|----------------|--|
| TRANSFER_SPECS | Each transfer specification source and destination line is logged to the log file as it is processed. This is useful when tracking down an unmatched transfer specification pair or to ensure that the transfer specifications are expanded appropriately.           |
| UNCORRELATED   | One feature of each feature type is logged if it was not matched by any source line of a transformation specification. This assists in determining exactly which features were not correlated, so transformation specifications may be written for them.             |
| UNGROUPED      | One feature of each feature type is logged if it was correlated and transformed but had no output destination. For example, if a Shape feature of the type <b>road</b> was created but there was no SHAPE_DEF for the <b>road</b> type, the feature would be logged. |

## Mapping File Identification

To assist in supporting mapping files that may be distributed widely, FME supports a directive that allows the exact version of a mapping file to be echoed in the FME log file. This gives quick confirmation that the correct mapping file version was used when browsing the log file.

The MAPPING\_FILE\_ID directive is used for this purpose. The syntax is:

```
MAPPING_FILE_ID <identification string>
```

For example, if a mapping file contained the following line (which should appear as a single, continuous line):

```
MAPPING_FILE_ID TRIM to Shape Release 2.0 Oct 1 '02
```

then the string "TRIM to Shape Release 2.0 Oct 1 '02" would be echoed to the FME log file each time the mapping file is run.

### **Minimum Build Requirement**

To ensure that a mapping file is not used with an older version of FME which may not have all the facilities used in mapping file, the mapping file may specify the minimum FME build number that can run it. The FME\_MINIMUM\_BUILD directive is used for this purpose. The syntax is:

```
FME_MINIMUM_BUILD <build number>
```

For example, if a mapping file contained the line:

```
FME_MINIMUM_BUILD 5600
```

then no FME with a build earlier than 5600 would be permitted to run the mapping file.

## Contact Us

### Contacting Safe Software

For technical support, click Help > Contact Safe Support.

[Visit our website](#) for more information on contacting Safe Software.