# Table of Contents

FME Desktop Advanced 1.1  
About This Document 1.2  
  Course Overview 1.2.1  
  Course Resources 1.2.2  
Advanced Parameter Use 1.3  
  FME Parameters 1.3.1  
  User Parameters 1.3.2  
  Exercise: Metadata Writer Parameterizing 1.3.3  
  Parameter Types 1.3.4  
  Linking Parameters 1.3.5  
  Pre-Linked Parameters 1.3.6  
  Exercise: Data Translation Code Review 1.3.7  
  Shared Parameters 1.3.8  
  Parameter Settings 1.3.9  
  Specialist Parameters 1.3.10  
  Exercise: Data Translation Code Review pt2 1.3.11  
  Module Review 1.3.12  
  Q&A Answers 1.3.13  
Performance Considerations 1.4  
  Performance and FME 1.4.1  
  64-Bit FME 1.4.2  
Logfile Interpretation 1.4.3  
  Logfile Command Section 1.4.3.1  
  Logfile Configuration and Setup Section 1.4.3.2  
  Logfile Translation/Transformation Section 1.4.3.3  
  Logfile Summary Section 1.4.3.4  
  Logfile Timestamp Interpretation 1.4.3.5  
Exercise: Cell Phone Signal Processing - Log File 1.4.4  
Reader and Writer Optimization 1.4.5  
  Assessing Reader Performance 1.4.5.1  
  Improving Reader Performance 1.4.5.2  
  Assessing Writer Performance 1.4.5.3  
  Improving Writer Performance 1.4.5.4  
Exercise: Cell Phone Signal Processing - Read/Write Performance 1.4.6  
Transformer Optimization 1.4.7  
  Improving Transformer Performance 1.4.7.1  
  More Transformer Performance Improvements 1.4.7.2
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exercise: Cell Phone Signal Processing - Transformation Performance</td>
<td>1.4.8</td>
</tr>
<tr>
<td>Database Optimization</td>
<td>1.4.9</td>
</tr>
<tr>
<td>Optimizing Database Readers</td>
<td>1.4.9.1</td>
</tr>
<tr>
<td>Optimizing Database Writers</td>
<td>1.4.9.2</td>
</tr>
<tr>
<td>Exercise: Garbage Collection Day Project</td>
<td>1.4.10</td>
</tr>
<tr>
<td>Parallel Processing</td>
<td>1.4.11</td>
</tr>
<tr>
<td>Exercise: Performance Review Project</td>
<td>1.4.12</td>
</tr>
<tr>
<td>Server and Cloud Performance</td>
<td>1.4.13</td>
</tr>
<tr>
<td>Module Review</td>
<td>1.4.14</td>
</tr>
<tr>
<td>Q+A Answers</td>
<td>1.4.15</td>
</tr>
<tr>
<td>Custom Transformers</td>
<td>1.5</td>
</tr>
<tr>
<td>Creating Custom Transformers</td>
<td>1.5.1</td>
</tr>
<tr>
<td>Exercise: Creating a Custom Transformer</td>
<td>1.5.2</td>
</tr>
<tr>
<td>Input and Output Ports</td>
<td>1.5.3</td>
</tr>
<tr>
<td>Exercise: Editing a Custom Transformer</td>
<td>1.5.4</td>
</tr>
<tr>
<td>Schema Handling</td>
<td>1.5.5</td>
</tr>
<tr>
<td>Automatic Schema Handling</td>
<td>1.5.5.1</td>
</tr>
<tr>
<td>Editing Schema</td>
<td>1.5.5.2</td>
</tr>
<tr>
<td>Exercise: Published Parameters</td>
<td>1.5.6</td>
</tr>
<tr>
<td>Custom Transformer Types</td>
<td>1.5.7</td>
</tr>
<tr>
<td>Creating Linked Transformers</td>
<td>1.5.8</td>
</tr>
<tr>
<td>Switching Transformer Type</td>
<td>1.5.9</td>
</tr>
<tr>
<td>Custom Transformer Versioning</td>
<td>1.5.10</td>
</tr>
<tr>
<td>Exercise: Custom Transformer Modes</td>
<td>1.5.11</td>
</tr>
<tr>
<td>Parallel Processing</td>
<td>1.5.12</td>
</tr>
<tr>
<td>Exercise: Parallel Processing</td>
<td>1.5.13</td>
</tr>
<tr>
<td>Looping</td>
<td>1.5.14</td>
</tr>
<tr>
<td>Exercise: Looping</td>
<td>1.5.15</td>
</tr>
<tr>
<td>Module Review</td>
<td>1.5.16</td>
</tr>
<tr>
<td>Q+A Answers</td>
<td>1.5.17</td>
</tr>
<tr>
<td>Advanced Readers and Writers</td>
<td>1.6</td>
</tr>
<tr>
<td>Writing Zip Files</td>
<td>1.6.1</td>
</tr>
<tr>
<td>Fanouts</td>
<td>1.6.2</td>
</tr>
<tr>
<td>Feature Type Fanouts</td>
<td>1.6.2.1</td>
</tr>
<tr>
<td>Dataset Fanouts</td>
<td>1.6.2.2</td>
</tr>
<tr>
<td>Exercise: Development Zone Translation</td>
<td>1.6.3</td>
</tr>
<tr>
<td>Generic Reader and Writer</td>
<td>1.6.4</td>
</tr>
<tr>
<td>Generic Reader</td>
<td>1.6.4.1</td>
</tr>
<tr>
<td>Generic Writer</td>
<td>1.6.4.2</td>
</tr>
<tr>
<td>Exercise: Community Map Translation</td>
<td>1.6.5</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Dynamic Translations</td>
<td>1.6.6</td>
</tr>
<tr>
<td>Creating Dynamic Translations</td>
<td>1.6.7</td>
</tr>
<tr>
<td>Exercise: Dynamic Community Map Translation</td>
<td>1.6.8</td>
</tr>
<tr>
<td>Dynamic Schema Handling</td>
<td>1.6.9</td>
</tr>
<tr>
<td>Dynamic Schema Sources</td>
<td>1.6.9.1</td>
</tr>
<tr>
<td>Dynamic Feature Types</td>
<td>1.6.9.2</td>
</tr>
<tr>
<td>Dynamic Attributes</td>
<td>1.6.9.3</td>
</tr>
<tr>
<td>Dynamic Geometry</td>
<td>1.6.9.4</td>
</tr>
<tr>
<td>Exercise: Creating a new Community Map Dataset</td>
<td>1.6.10</td>
</tr>
<tr>
<td>Advanced Dynamic Schemas</td>
<td>1.6.11</td>
</tr>
<tr>
<td>Exercise: Creating a table-driven Community Map Dataset</td>
<td>1.6.12</td>
</tr>
<tr>
<td>Module Review</td>
<td>1.6.13</td>
</tr>
<tr>
<td>Q+A Answers</td>
<td>1.6.14</td>
</tr>
<tr>
<td>Advanced Attribute Handling</td>
<td>1.7</td>
</tr>
<tr>
<td>Constructing Values</td>
<td>1.7.1</td>
</tr>
<tr>
<td>Editor Dialogs</td>
<td>1.7.2</td>
</tr>
<tr>
<td>FME Functions</td>
<td>1.7.3</td>
</tr>
<tr>
<td>Exercise: Taxation Report Project</td>
<td>1.7.4</td>
</tr>
<tr>
<td>Conditional Values</td>
<td>1.7.5</td>
</tr>
<tr>
<td>Exercise: Flood Risk Assessment</td>
<td>1.7.6</td>
</tr>
<tr>
<td>Exercise: Flood Risk Assessment: Simple Filtering</td>
<td>1.7.6.1</td>
</tr>
<tr>
<td>Exercise: Flood Risk Assessment: Complex Filtering</td>
<td>1.7.6.2</td>
</tr>
<tr>
<td>Exercise: Flood Risk Assessment: Conditional Values</td>
<td>1.7.6.3</td>
</tr>
<tr>
<td>Multiple Feature Attributes</td>
<td>1.7.7</td>
</tr>
<tr>
<td>Exercise: Precipitation Calculations</td>
<td>1.7.8</td>
</tr>
<tr>
<td>Null Attributes</td>
<td>1.7.9</td>
</tr>
<tr>
<td>Handling Null Attributes</td>
<td>1.7.9.1</td>
</tr>
<tr>
<td>Setting Null Attributes</td>
<td>1.7.9.2</td>
</tr>
<tr>
<td>Exercise: Parks Dataset Sorting</td>
<td>1.7.10</td>
</tr>
<tr>
<td>Module Review</td>
<td>1.7.11</td>
</tr>
<tr>
<td>Q+A Answers</td>
<td>1.7.12</td>
</tr>
<tr>
<td>Course Wrap-Up</td>
<td>1.8</td>
</tr>
<tr>
<td>Product Information and Resources</td>
<td>1.8.1</td>
</tr>
<tr>
<td>Community Information and Resources</td>
<td>1.8.2</td>
</tr>
<tr>
<td>Feedback and Certificates</td>
<td>1.8.3</td>
</tr>
<tr>
<td>Thank You</td>
<td>1.8.4</td>
</tr>
<tr>
<td>Exercise: A Pointless Quiz!</td>
<td>1.8.5</td>
</tr>
</tbody>
</table>
FME Desktop Advanced Training Manual

This is the manual for the advanced-level training course for Safe Software’s FME Desktop application.

The training builds upon basic training to cover functionality that is important to all FME users wishing to take their skills to the next level.

Course Structure

The full course is made up of five main sections. These sections are:

- User Parameters
- Performance
- Custom Transformers
- Advanced Reading/Writing
- Advanced Attribute Handling

Current Status

The current status of this manual is: COMPLETE: this manual can be used for training

It is valid for FME2016.0, FME2016.1, and FME2016.1.1

The status of each chapter is:

- Chapter 0: Complete content. No exercises
- Chapter 1: Complete content and exercises
- Chapter 2: Complete content and exercises
- Chapter 3: Complete content and exercises
- Chapter 4: Complete content and exercises
- Chapter 5: Complete content and exercises
- Chapter 6: Complete content. No exercises
**NB:** Even for completed content, Safe Software Inc. assumes no responsibility for any errors in this document or their consequences, and reserves the right to make improvements and changes to this document without notice. See the full licensing agreement for further details.
About This Document

This advanced course is for users with prior experience of FME Desktop.

Look out for residents of the City of Interopolis, who will appear from time-to-time to give you advice and dispense FME-related wisdom.

Mr. E.Dict (Attorney of FME Law says...)

On behalf of the City of Interopolis, welcome to this training course. Here is the standard legal information about this training document and the datasets used during the course.

Be sure to read it, particularly if you're thinking about re-using or modifying this content.

Licensing and Warranty

Permission is hereby granted to use, modify and distribute the FME Tutorials and related data and documentation (collectively, the “Tutorials”), subject to the following restrictions:

1. The origin of the Tutorials and any associated FME® software must not be misrepresented.

2. Redistributions in original or modified form must include Safe Software's copyright notice and any applicable Data Source(s) notices.

3. You may not suggest that any modified version of the Tutorials is endorsed or approved by Safe Software Inc.

4. Redistributions in original or modified form must include a disclaimer similar to that below which: (a) states that the Tutorials are provided “as-is”; (b) disclaims any warranties; and (c) waives any liability claims.

Safe Software Inc. makes no warranty either expressed or implied, including, but not limited to, any implied warranties of merchantability, non-infringement, or fitness for a particular purpose regarding these Tutorials, and makes such Tutorials available solely on an “as-is” basis. In no event shall Safe Software Inc. be liable to anyone for direct, indirect, special, collateral, incidental, or consequential damages in connection with or arising out of the use, modification or distribution of these Tutorials.
This manual describes the functionality and use of the software at the time of publication. The software described herein, and the descriptions themselves, are subject to change without notice.

**Data Sources**

**City of Vancouver**

Unless otherwise stated, the data used here originates from open data made available by the City of Vancouver, British Columbia. It contains information licensed under the Open Government License - Vancouver.

**Others**

Forward Sortation Areas: Statistics Canada, 2011 Census Digital Boundary Files, 2013. Reproduced and distributed on an "as is" basis with the permission of Statistics Canada. © This data includes information copied with permission from Canada Post Corporation.

Digital Elevation Model: GeoBase®

Fire Hall Data: Some attribute data adapted from content © 2013 by Wikipedia, used under a Creative Commons Attribution-ShareAlike license

Stanley Park GPS Trail: Used with kind permission of VancouverTrails.com.

**Copyright**

© 2005–2016 Safe Software Inc. All rights are reserved.

**Revisions**

Every effort has been made to ensure the accuracy of this document. Safe Software Inc. regrets any errors and omissions that may occur and would appreciate being informed of any errors found. Safe Software Inc. will correct any such errors and omissions in a subsequent version, as feasible. Please contact us at:

**Safe Software Inc.**
**Phone:** 604-501-9985  
**Fax:** 604-501-9965  
**Email:** train@safe.com  
**Web:** www.safe.com

Safe Software Inc. assumes no responsibility for any errors in this document or their consequences, and reserves the right to make improvements and changes to this document without notice.

**Trademarks**

FME® is a registered trademark of Safe Software Inc. All brand or product names are trademarks or registered trademarks of their respective companies or organizations.

**Document Information**

**Document Name:** FME Desktop Advanced Training Manual 2016.1
All screenshots and functionality relate to FME Desktop 2016.1.1, Build 16601
Course Overview

This training course builds upon the basic framework of workspace creation in FME Desktop. They are not so much topics that every user needs to know, but ones that will help all workspace authors who wish to take their FME skills to the next level.

Course Structure

The full course is made up of five main sections. These sections are:

- Advanced Parameter Use
- Performance Considerations
- Custom Transformers
- Advanced Reading and Writing
- Advanced Attribute Handling

The instructor may choose to cover as many of these sections as they feel are required, or possible in the time permitted. They may also cover the course content in a different order and will skip or add new content to better customize the course to your needs.

Therefore the length and content of the course may vary, particularly when delivered online.

Prerequisites

This training material is intended for persons with some prior experience of using FME. The content assumes a basic familiarity with the concepts and practices of FME Desktop; at least to the extent covered by the FME Desktop Basic Training.

In particular it would be helpful to be familiar with:

- FME transformers and basic transformation techniques
  - See Chapter 2 of the Basic Training
- Managing Readers, Writers and feature types in a workspace
  - See Chapter 4 of the Basic Training
- Data filtering and attribute management techniques in FME Workbench
  - See Chapter 5 of the Basic Training

About the Manual

The FME Desktop advanced training manual not only forms the basis for advanced FME Desktop training – in-person or online – but is also useful reference material for future work you may undertake with FME.

.1 UPDATE

The FME development cycle includes a number of follow-up releases; service packs we used to call them. A dialog box like this denotes features new or updated in FME2016.1, the first service pack since the .0 release.
Course Resources

A number of sample datasets and workspaces will be used in this course.

On Your Training Computer

The data used in this training course is based on open data from the City of Vancouver, Canada.

Most exercises ask you to assume the role of a city planner at the fictional city of Interopolis and to solve a particular problem using this data.

Whether it’s a local computer or a virtual computer hosted in the cloud, you’ll find resources for the examples and exercises in the manual at the following locations:

<table>
<thead>
<tr>
<th>Location</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:\FMEData2016\Data</td>
<td>Datasets used by the City of Interopolis</td>
</tr>
<tr>
<td>C:\FMEData2016\Resources</td>
<td>Other resources used in the training</td>
</tr>
<tr>
<td>C:\FMEData2016\Workspaces</td>
<td>Workspaces used in the student exercises</td>
</tr>
<tr>
<td>C:\FMEData2016\Output</td>
<td>The location in which to write exercise output</td>
</tr>
<tr>
<td>&lt;documents&gt;\My FME Workspaces</td>
<td>The default location to save FME workspaces</td>
</tr>
</tbody>
</table>

You should also find FME pre-installed, plus a digital copy of this manual.

Please alert your instructor if any item is missing from your setup.

You can find the latest version of FME Desktop and FME Server for Windows, Mac, and Linux - together with the latest Beta versions - on the Safe Software web site.

Course Etiquette

For online courses, please consider other students and test your virtual machine connection before the course starts. The instructor cannot help debug connection problems during the course!

For live courses, please respect other students’ needs by keeping noise to a minimum when using a mobile phone or checking e-mail.
**Advanced Parameter Use**

Parameters are controls that define how FME operates; for example, how a Reader reads data, how a transformer transforms it, and how a Writer writes it.

Almost every component in FME has parameters; of one type or another.

**Types of Parameter**

When looking at the types of parameter, it’s helpful to first consider the types of people who use FME and their role in the process.

**Workspace Authors** are the people who design and create a workspace. They use FME Workbench and set parameters to control how the workspace runs.

**Workspace Users** are the people who make use of a workspace, without necessarily having created it first. The user might have very little knowledge of FME, and may never have used FME Workbench, but they still may need to set parameters to control how the workspace runs.

In light of these two roles, we can say there are two different types of parameter; **FME Parameters** (for authors to use) and **User Parameters** (for FME users to use).

---

**Ms. Analyst says...**

*Hi, I'm Ms. Analyst. I'm here to help leverage your user parameter strategies and will touch base from time-to-time throughout this chapter.*
FME Parameters

FME parameters are those built into the FME Workbench interface. They directly control a translation and can be found in various places, such as transformer dialogs or feature type dialogs. However, for the most part you will find all FME Parameters in the Navigator Window of FME:

- Overwrite Existing File: Yes
- Line Termination: System
- Write Last Line Terminator: Yes
- Character Encoding: SYSTEM
- Write UTF Byte Order Mark: Yes
- MIME Type: text/plain

Here, for example, are the FME Parameters for a text file Writer. They include options to overwrite or append to an existing text file, and the type of character encoding that should be used.

These are parameters the workspace author will use. The user is not expected to set these, because they are not assumed to have enough experience of Workbench to know where to find the parameter or how to set it.

For example, the author might decide that the Byte Order Mark should not be written in this output.

They double-click the parameter to open a dialog in which they can change the parameter value.

Ms. Analyst says...

In case you were wondering, the Byte Order Mark (aka BOM) is a special character found in the header of a text file. It denotes whether the text is in a Unicode encoding, which Unicode encoding it is, and what the endianness of the text is. It helps different applications to read the data correctly.

You want to know what “endianness” means now, don’t you? It’s not important. Outside of our core competency.
User Parameters

User Parameters are those that are created by an FME author, but for an FME user to use. In other words, they are a way for the end-user of the workspace to provide their input to a workspace.

Creating a User Parameter

User parameters appear in a special section of the Navigator window, labelled User Parameters. Here, for example, two user parameters have been defined:

Each of these user parameters will allow the end-user of a workspace to enter information into the translation; whether to process line features and what color to write them in.

A user parameter is easy to create by right-clicking on the User Parameters label and choosing Add Parameter:

A dialog appears in which the author can define the parameter. In this case they are creating a parameter in which the user can enter their name:
**Entering Information into a User Parameter**

Once a user-parameter is defined, when the workspace is run the user is prompted to give their input.

In FME Workbench (or the FME Quick Translator) the user is prompted through a simple dialog:

In FME Server, the user is prompted through a web page:
Ms. Analyst says...

In the FME Quick Translator, the user is always prompted to fill in user parameters when running a workspace. However, in FME Workbench, prompting only happens when the prompt option is turned on in the menubar:

Think of it as a nano-paradigm shift between the two applications.

Using a User Parameter

Getting input from a user is pointless if it is not used, so it’s also necessary to actually do something with that input.

User parameters can be exploited in a number of places. Firstly they can be tied to an FME parameter (more information on that in the next section), but they can also be used to provide values to transformers and attributes in a workspace.

For example, here the author is making use of the color and username parameters (in the FeatureColorSetter and AttributeManager transformers):
The author set up these transformers to use the parameter input by selecting it in the transformer like so:

Now when the workspace is run, the end user can select the color of features to write, plus enter their name into a text field and have it entered into the UpdatedBy attribute in the output.

**Miss Vector says...**

Do you remember me? I'm Miss Vector. I ask the hard questions around here. But to start with tell me this, what are our two roles of FME user?

1. Creator/Inspector
2. Author/User
3. Reader/Writer
4. Maker/Consumer
Here's another question, only slightly less easy. Look at the ParameterFetcher transformer. What does it do?

1. Fetches the name of a user parameter
2. Fetches the value of a user parameter
3. Fetches the type of a user parameter
4. Fetches the user a cup of tea
Exercise 1  Parameterize a Metadata Writer

<table>
<thead>
<tr>
<th>Data</th>
<th>Parks (MapInfo TAB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>Allow user input to metadata fields</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>Use of FME parameters. Creation and use of User Parameters</td>
</tr>
<tr>
<td>Start Workspace</td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\Parameters-Ex1-Begin.fmw</td>
</tr>
<tr>
<td>End Workspace</td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\Parameters-Ex1-Complete.fmw</td>
</tr>
</tbody>
</table>

In this example, imagine that you are a GIS technician working for a city planning department.

The team responsible for maintaining parks has a workspace that translates their data from the source MapInfo TAB format to Google KML. It also writes a file of XML metadata to show who translated the data and when.

At the moment there are a number of problems they face.

- The XML output is not particularly well formatted
- The date attribute is being rejected by an online XML validator
- All of the XML metadata fields are hard-coded in an AttributeCreator transformer. This is quite inconvenient (especially when they want to run the workspace on FME Server!)

You have been assigned to help solve these problems. At least one of these requires you to create user parameters to take the place of hard-coded values.

1) Start Workbench

Start Workbench and open the workspace C:\FMEData2016\Workspaces\DesktopAdvanced\Parameters-Ex1-Begin.fmw

The metadata part of the translation consists of the two transformers and XML Writer feature type.

The Sampler transformer ensures that only one record is written to the output metadata, by discarding all but one feature, and the AttributeCreator creates a set of attributes to write to the metadata.

Open the parameters dialog for each transformer in turn. These are FME parameters, set by the workspace author and not available to the end-user. Here, for example, are the parameters for the Sampler transformer:
2) Change XML Writer Parameter

The style of the XML file being written is controlled by an FME parameter called Pretty Print:

To ensure the output is always well-formatted, we should set this parameter to Yes - but we won't create a user parameter from it, because we don't want the end-user to change it.

In the Navigator window locate the XML Writer, expand the parameters list, and locate the parameter labelled Pretty Print. Double-click on it.

In the dialog that opens, change the value to Yes and then click OK to close the dialog.
We have now - as a workspace author - changed an FME parameter.

3) Create User Parameter
The output schema has three variable attributes: username, user company (organization), and user email. We should create a user parameter for each of these to allow the end-user to enter that information.

Firstly, locate the User Parameters section of the Navigator window, right-click on it, and choose the option to Add Parameter:

In the new dialog, select Text as the type of parameter to create (there will be more on parameter types in the next section). Each parameter needs a name, so call this one UserNameParam. Now enter a prompt, such as “Enter your name.”

Click OK to close the dialog and create the parameter, which will now appear in the Navigator window.

4) Create Remaining User Parameters
Repeat the previous step twice more, this time creating parameters called UserEmailParam and UserCompanyParam.

The prompts should be “Enter your email address” and “Enter your company name.”

When done the Navigator window looks like this:
5) Use User Parameter – Method 1

Each of the user parameters we've just defined provide values that need to go into attributes in the Writer schema. There are a number of ways to extract the value for such a purpose and we'll use a different way for each parameter, just to illustrate the different methods.

So, firstly open the parameters dialog for the AttributeCreator. This transformer is what currently creates the attributes for the output.

Click in the Value field for the AuthorName attribute. Click on the drop-down arrow, then select User Parameter > UserNameParam.

Once done the value field will change to a special icon and show the parameter that was chosen:

While in this dialog, click on the AuthorEmail and AuthorCompany fields, and press the minus button to delete them. That's just so we can demonstrate dealing with these a different way.
6) Fix Date Attribute
Looking at the AttributeCreator we can see that the date field is being entered as a fixed value. Although not a user parameter as such, it’s obvious that the user must be setting this manually at run time.

Additionally, the date is not structured to an ISO standard, which is why the output fails XML validation.

Let’s fix these issues. First click on the drop-down arrow next to the date attribute value, then choose Open Text Editor...

In the text editor remove any existing content and replace it with:

@Timestamp(^Y-^m-^d)

This uses an FME function to return a date value, the structure defined will match the ISO date standard.

Firefighter Mapp says...

An alternative method is to simply set the date value to “TODAY” and using the DateFormatter transformer to convert it to a real date in an ISO standard.

7) Use User Parameter – Method 2
A second way to extract the value from a user parameter is with a ParameterFetcher transformer.

Place a ParameterFetcher transformer (after the AttributeCreator is fine). Open the parameters dialog.

Select UserEmailParam as the parameter to fetch. Enter AuthorEmail as the name of the target attribute:
Ms. Analyst says…

Did you notice that the list of parameters available includes many FME-related system parameters? These are particularly useful for use on FME Server.

8) Use User Parameter – Method 3

The final method to extract the value from a user parameter is with a schema attribute value.

To achieve this, locate the metadata feature type on the canvas and right-click the AuthorCompany attribute. Then select the Edit Value option:

In the dialog that opens, you can enter a fixed (constant) value, but in our case we’ll click on the drop-down arrow, select User Parameters, and then select UserCompanyParam:
Click OK to close the dialog and the feature type should look like this. Notice how the attribute that has had its value set is now highlighted with a specific icon:

9) Save and Run Workspace

Save the workspace and then – as if you were the end-user – run it. Be sure to set the Prompt option on the toolbar first:

When prompted enter your details into the fields that have been newly created:

Locate and open the XML file to ensure the contents have been inserted as expected:

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <xfme:ParksMetadata>
    <xfme:AuthorName>Mark Ireland</xfme:AuthorName>
    <xfme:AuthorEmail>mark.irland@safe.com</xfme:AuthorEmail>
    <xfme:AuthorCompany>Safe Software</xfme:AuthorCompany>
    <xfme:CreateDate>2015-12-09</xfme:CreateDate>
  </xfme:ParksMetadata>
</xfme:xml-tables>
```

CONGRATULATIONS

By completing this exercise you have learned how to:
- Set an FME parameter (on the XML Writer)
- Create a text type user parameter
- Use a user parameter in a regular transformer
- Use a user parameter in a ParameterFetcher transformer
- Use a user parameter in a Edit Value dialog
- Use the TimeStamp function in a text editor to create an XML-valid date
Types of User Parameters

There are many different types of user parameters and many different ways to make use of them. The most common parameter types can be grouped as:

- Text Parameters
- Numeric Parameters
- Choice Parameters

Text Parameters

Text parameters are a simple way to accept plain text values into a workspace. A Text parameter allows a single line of text, while Text (Multiline) parameters allow the user to enter text broken over a number of lines.

There is no limitation on the characters that can be entered. However, if you want the user to enter encoded characters, then you must use type Text (Multiline), like so:

That way FME will retain and use the actual value, here shown in the FME Data Inspector:
Ms. Analyst says…

*It’s worth being aware that not every transformer and format in FME will handle encoded text. If you are unsure, then it’s safer to use a Text parameter – that everything will support – rather than a Text (Multiline) parameter that is not universally supported. Encoded text will not be handled, but the translation should not cause an error.*

**Numeric Parameters**

There are two types of numeric parameters: Float and Integer. As their names suggest, these are simply ways for a user to enter a floating point number or an integer number.

![Parameter Types](image)

These parameters are good examples of how FME will parse the input to ensure it matches the parameter type:

![Translation Parameters](image)

This image shows how non-numeric characters in either type, or a decimal point in the integer type, will be detected and rejected with a red-colored field.

Ms. Analyst says...

*The Slider is another numeric parameter type, but it’s a bit more “outside the box”. Instead of manually entering a number - the user gets to pick the value from a sliding scale widget.*

**Choice Parameters**

A choice parameter is when the user is presented with a fixed list of options and selects one of them. Different choice type parameters allow the user to pick from a list, pick multiple entries from a list, or type in text as an alternative to a list:
Here the user is being asked for their name. However, the names of all users are already known – presumably this is for a particular company’s staff – so a list of them is created:

That way, the user is prompted to select their name from a list. They don’t have to type it in manually.

A Choice with Alias parameter is the same as a Choice parameter in that the end-user gets to pick a value from a list. However, a lookup table maps the chosen entry to a value that gets provided to FME.

For example, this workspace takes incoming features and matches them to a database using an EmployeeID.

EmployeeID is provided by the end-user, but they can’t always remember their own ID number. So, the author creates a Choice with Alias user parameter.

The parameter is configured like so:
Notice that there are two fields in this configuration dialog; the display name and the actual value.

When a user selects their name from the list, then the value provided to the workspace is actually their employee ID. That way employee ID can be used as a match in the Joiner, without the end-user having to remember it!

---

Ms. Analyst says…

*Choice (Multiple) and Choice with Alias (Multiple) are very similar parameters (to Choice and Choice-with-Alias), but let the enduser select multiple values. For example, if a manager wanted to run reports on several employees, this is what they could use. Multiple values are returned space-delimited.*

---

Miss Vector says...

*Besides the above, there are some more - specialist - parameter types. Which of the following is NOT a valid attribute type?*

1. Coordinate System Name
2. Password
3. String Encoding
4. URL
Linking Parameters

As we know, there are FME parameters that control FME directly, and user parameters that allow input from a user.

Sometimes a workspace author needs the user's input to apply directly to an FME parameter, and this is done by linking the user parameter to the FME parameter.

For example, an FME author has a workspace that writes to a textfile. There is a textfile parameter to "Write UTF Byte Order Mark". The author wishes for the end-user to control whether that marker is created, but doesn't want them to have to search for that parameter.

So, besides the FME parameter, they create a user parameters:

Their user parameter has Yes and No choices, but as yet it does not do anything. It must be linked to the FME parameter.

The author does this by right-clicking the FME parameter and choosing Link to User Parameter:
Then they select their user parameter to link to. Alternatively they can do the reverse; right-click the User parameter and choose Apply To [FME Parameter].

Now the FME parameter is linked to the User Parameter, so whatever the user chooses will be applied directly to the Write UTF Byte Order Mark parameter:

If the author changes their mind, there is always an option to unlink the user parameter and return to direct author control:
Creating Direct Links

In the previous example, a user parameter was created separately and then linked to the FME parameter. However, there is a shortcut to this, where it is possible to both create and link a parameter simultaneously.

In the Navigator window, simply right-click an existing FME parameter and choose the option to “Create User Parameter”.

This opens a dialog and automatically fills in a definition to create a new user parameter.
Click OK and the user parameter is created and automatically linked to the FME parameter.

Ms. Analyst says...

You can do the same one-step action inside a transformer dialog too, like so:

Here the workspace author is creating a user parameter linked to the Elevation FME parameter in a 3DForcer transformer.

Advantages and Disadvantages of Direct Links

You might be wondering why you would ever link a user parameter separately, or why we showed that process first. It’s because there are advantages and disadvantages to both methods.

Advantages

Creating a linked FME parameter directly has the obvious advantage that it is a single-step process. The creation and linking of the user parameter is done in a single action.

Additionally, a user parameter created from an FME parameter automatically gets the correct data type.

For example, if the 3DForcer parameter requires a floating point number, any user parameter created from it will automatically be of type float, and can’t be changed.
Disadvantages

The automatic setting of data type can, however, be a limitation too. Say, for example, the author wanted to provide a list of permitted elevations; 0.0, 10.0, 50.0, etc. A float parameter does not allow that. They would have to create a choice user parameter separately and then link it to the FME parameter as discussed.

Of course, the author needs to take care that the values provided by the user parameter were of a type that matched those expected by the FME parameter. FME isn't able to parse the input from a user parameter to ensure it matches the FME parameter it is linked to.

The other disadvantage is one of persistence of the user parameter.

It's like this: if a user parameter is created directly from an FME parameter on a transformer, then it is forever tied to that transformer. If the transformer is deleted, then the FME parameter will be deleted too.

However, if a user parameter is created separately, and linked manually to a transformer’s FME parameter, then it will remain in the workspace, even if the transformer is deleted.

This could be seen as either an advantage or disadvantage, depending on whether you would like this behavior or not!
**Pre-linked Parameters**

In some scenarios, user parameters are automatically created and linked to an FME parameter, without any sort of manual action by the workspace author.

For example, any time a Reader or Writer is added to a workspace, their source/destination dataset parameters are automatically turned into user parameters.

Here, a Source MapInfo TAB parameter is automatically linked to a user parameter called SourceDataset/MITAB:

![Navigator](Navigator.png)

This automatically occurs for parameters that are important to the end-user and that appear in nearly all workspaces.

---

**Miss Vector says...**

If you – as the workspace author – don’t want or require the end-user to have access to pre-linked parameters, then what can you do?

1. Delete the Reader/Writer
2. Unlink the user parameter
3. Delete the FME parameter
4. Delete the user parameter
The Public Safety department at the city has just bought into FME and started using it for data translations. However, having not (yet) taken the FME Desktop training course, they are not confident users and would like some assistance. You have been tasked to carry out a "code review" on one of their workspaces. At least one of the issues you find is likely to involve creating user parameters to take the place of hard-coded values.

1) Start Workbench

Start Workbench and open the workspace C:\FMEData2016\Workspaces\DesktopAdvanced\Parameters-Ex2-Begin.fmw

This is the workspace created by your colleagues:

Notice that it converts from an Esri Geodatabase to Esri Shapefile format. Currently the tables to process are chosen by disabling unwanted ones in the workspace. Similarly, they are setting the destination coordinate system and data encoding using Navigator parameters. This is all very user-intensive.

Also notice that the only annotations in the workspace are there to help the end user make such edits. Really there should be no need for that; published parameters should prompt the user instead, and that is what we will implement here.

2) Clean Up Auto-Created User Parameters

Open up the User Parameters section of the Navigator window. Notice how there are already user parameters for the source and destination datasets:
Your public safety colleague tells you that the source data will never change, so that parameter is of no use. So delete the user parameter labelled “SourceDataset_FILEGDB.”

However, she tells you that the destination location can be set by the user, so keep the parameter for DestDataset_SHAPE.

3) Create Encoding Parameter
The Public Safety team want to make it easier to set the encoding of the output dataset. Currently users are pointed towards the point on the Navigator where that Writer parameter exists by a workspace annotation!

This shows you how difficult it is for them to locate the correct parameter in the Navigator window. Let’s solve that with a user parameter.

Locate the Shape Writer in the Navigator window and expand the list of FME parameters. Identify the Character Encoding parameter, right-click on it and choose Create User Parameter:

Simply click OK on the dialog that opens and a user parameter is created and linked to the FME one. Now there is a user parameter to make it easy to set that FME parameter.

4) Create Coordinate System Parameter
Another requirement, you are told, is an ability to set the output coordinate system. Again this is currently done by using an annotation to point the user towards the Navigator Window.

However, if you simply publish the Writer’s coordinate system parameter – try it and see – then there will be a problem. The parameter will allow the end-user to select ANY coordinate system supported by FME.

This is not necessarily very useful. Since the data is located in Vancouver, BC, it makes little sense for the user to be able to reproject it to (for example) NZMG (a New Zealand coordinate system).

It would be preferable if the parameter only allowed the end-user to select a coordinate system from a smaller list.
So create a new user parameter (User Parameters > Add Parameter), and set the Type to be **Choice with Alias**. Set the Name to be CoordSysParam, and set the prompt to be Select Output Coordinate System:

![Add/Edit User Parameter dialog](image)

Now click the [...] button to the right of the Configuration setting. This opens a dialog in which to configure the parameter. Normally we would enter values manually in a Choice with Alias parameter, but for coordinate systems (and Reader/Writer formats) we have the option to have FME define them for us.

Click on the button labelled Import and choose Coordinate System(s):

![Edit Choice with Display Name](image)

This opens a list of coordinate systems that we can import as values in our user parameter.

Locate and put a checkmark in the box for the following coordinate systems:

- UTM83-10
- BCALB-83
- LL83
- CANBC-LCC

Then click OK to close this dialog. You will be returned to the configuration dialog and find that names and values have been automatically entered for these coordinate systems:
The left hand side shows what the user is prompted to select, the right hand side what the value fed to FME will be.

Click OK and then OK again to close the remaining dialogs and create the user parameter.

5) Link Coordinate System Parameter

Now we have the user’s selection but we still have to apply it to the real parameter. So locate the Writer’s coordinate system parameter, right-click on it, and choose Link to User Parameter:

When prompted, select the newly created CoordSysParam and click OK to accept the selection. Now when the workspace is run the user is prompted to select a coordinate system, and that system's shortname value is passed to FME.

6) Create Tables Parameter

The final task for us here is to create a way to decide which tables are going to be read. If you remember, at the moment the way your colleagues do this is by disabling various Reader feature types. However, there has to be a better method.

This is an interesting task because we want to control the source tables (Libraries, Parks, etc.), but based on the selection of output tables (CommunityFacilities, Environment, and Miscellaneous).

For example, we want the user to select output feature types like "Environment", which needs both "Parks" and "DrinkingFountains" Reader feature types.

But this we can do very easily. Firstly locate the Feature Types to Read parameter in the CommunityMap Reader "Features to Read" parameters (in the Navigator window):
Right-click on it and choose Create User Parameter. A dialog will open that is already populated with a list of feature types:

Check the box that is labelled Use Alternate Display Name. This provides the ability to give alternative names for each feature type. What we need to do is use this dialog to group together common Reader feature types under a single display name.

Delete the entry for GarbageSchedule, as this data isn’t connected and is not needed.

Then, match the contents of the workspace by editing the Display Names. They should match as follows (the order is not important):

<table>
<thead>
<tr>
<th>Display Name</th>
<th>Feature Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Community Facilities</td>
<td>Libraries</td>
</tr>
<tr>
<td>Community Facilities</td>
<td>CommunityCentres</td>
</tr>
<tr>
<td>Environment</td>
<td>Parks</td>
</tr>
</tbody>
</table>
Underneath that change the prompt to read “Tables to Write” and then click OK to close the dialog.

What we have done here is set up a list of output layers to select from, with a list of input layers that each refers to.

7) Save and Run Workspace

Save the workspace. Then start up the FME Quick Translator application, located on the Start menu under the FME Desktop Utilities folder.

In there, select Run from the Getting Started menu:

Browse to the newly saved workspace, select it, and click Open. You will be presented with a list of published parameters, just as the end-user would see it:
Pick Unicode 8-bit (utf-8) as the encoding. Select a coordinate system, noting how the user is restricted to those chosen by us. Select one or two of the tables to write and click OK to run the workspace.

The translation will be carried out. Inspect the data to ensure the results are correct. The Community Facilities – for example – should be made up of both libraries and community centres.

**Advanced Exercise**

*Speaking of Best Practice, don’t forget to tidy up the workspace and give it a better style and structure!*

**CONGRATULATIONS**

*By completing this exercise you have learned how to:*
- Remove pre-linked parameters
- Create a simple pre-linked parameter
- Create a Choice with Alias user parameter
- Use a Choice with Alias parameter to define coordinate systems
- Manually link a user parameter to an FME parameter
- Publish the Feature Types to Read parameter
- Set up and use alternative names in the Feature Types to Read parameter*
Shared and Embedded Parameters

Shared and Embedded parameters are not specific types of parameters, instead they refer to two different ways in which parameters might be used.

Shared Parameters

There is no limit on the number of times a user parameter can be used or linked to an FME parameter. The value obtained from a user parameter can be used as many times as is required.

When a parameter is used in two or more places it can be described as a shared parameter.

For example, a workspace has a user parameter called TOLERANCE (here being used inside a Generalizer):

However, the workspace author has decided to apply the same parameter in three places in total; two Generalizers and a Snapper:

- Generalizer [Generalizer:1]
  - Transformer Name: Generalizer
  - Algorithm: Douglas (Generalize)
  - Preserve Shared Boundaries: No
  - Generalization Tolerance ($\geq 0$): (Linked to ‘TOLERANCE’)
  - Smoothness Factor (1-30): <Unused>

- Generalizer_2 [Generalizer:1]
  - Transformer Name: Generalizer_2
  - Algorithm: Douglas (Generalize)
  - Preserve Shared Boundaries: No
  - Generalization Tolerance ($\geq 0$): (Linked to ‘TOLERANCE’)
  - Smoothness Factor (1-30): <Unused>

- Snapper [Snapper:5]
  - Transformer Name: Snapper
  - Group By: <not set>
  - Snapping Type: End Point Snapping
  - Snapping Tolerance: (Linked to ‘TOLERANCE’)
  - Add Additional Vertex: NEVER

The advantage is that the same value can be used without the user having to enter it multiple times.
**Embedded Parameters**

Sometimes in FME, string values need to be constructed from multiple components. When a string is constructed in such a way as to use multiple user parameters, we call it *Embedding Parameters*.

For example, here a file being written is constructed from two user parameters: one is a fixed output path and the other is a user's name:

```
Target Filename: $(OutputFolder)\$(UserName).txt
```

Basically a user parameter is created to accept the user's name (UserName) and a second parameter (OutputFolder) accepts a folder to write output. The FME parameter on an AttributeFileWriter transformer constructs a single value from these two user parameters.

Similarly, the value from one user parameter can be embedded inside the definition of another! You might call this scenario *nested user parameters*.

---

**Ms. Analyst says...**

*Going forward, another example of embedded/nested user parameters can be found in the section on Private Parameters*
Parameter Settings

When a user parameter is created two checkbox options exist; one is labelled Published and the other Optional.

**Published Parameters**

The purpose of this option is to expose or hide the parameter from the end user. If the Published box is checked, then the user will be prompted to enter a value. If the box is unchecked then they will not be prompted, and the parameter will be treated as "private".

**Private Parameters** have two uses.

Firstly a private parameter is a way for a workspace author to create a shared parameter without having it exposed to the user.

For example, if they want to supply the same tolerance value to several Snapper transformers – but that value is set by the author, not the user – then a private parameter is used.

A second use of a private parameter is to embed a user's partial input into a larger parameter.

For example, here the workspace author wants the user to enter a filename to be written as output, but the name of the folder to write it in is determined by the author (here as a mix of fixed path and a job ID):
Ms. Analyst says...

You might have noticed that there are a number of FME Server Parameters available to workspace authors who intend to deploy their creation on an enterprise scale.

In fact, if you look at the above screenshot, you might notice that a Server parameter (FME_JOB_ID) has been embedded into the FullPath private parameter!

Optional Parameters

The Optional checkbox tells FME whether the user parameter is compulsory or optional.

If, for example, the user parameter provides a tolerance value to a Generalizer transformer, the author will want to turn off this checkbox and make the parameter compulsory. A Generalizer that is not given a tolerance value will usually fail and making tolerance compulsory is one way to prevent that happening:

Alternatively, a parameter might exist for the user to set the color of a set of features. In this case the checkbox might be left checked so that it is optional and the user does not have to select a color. In that scenario the translation will not fail; the features will simply retain their original color.
<table>
<thead>
<tr>
<th>Miss Vector says...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tell me, is it possible to have a compulsory, private parameter? (i.e. both settings boxes are unchecked)</td>
</tr>
<tr>
<td>1. Yes</td>
</tr>
<tr>
<td>2. No</td>
</tr>
<tr>
<td>3. Yes, but you need to set a value immediately</td>
</tr>
<tr>
<td>4. Yes, but only for text or numeric parameters</td>
</tr>
</tbody>
</table>
Specialist Parameters

There are one or two parameter types that fall outside of the general text/numeric/choice types, and yet are worth taking a close look at.

Scripted Parameters

Scripted parameters go one step further than an embedded parameter.

Whereas an embedded parameter allows simple construction of a new value by concatenation, a scripted parameter allows a full Python or Tcl script to be used to construct a value.

For example, here is the definition for a Tcl scripted parameter that, again, concatenates a filename from the user with a set path from the author.

However, in this case, the script is used to test whether the workspace is being run on a Windows or Linux system, so it can set the output path accordingly:

```tcl
set realname ''
if {[string match 'C:*' $FME_MacroValues(FME_HOME)]} {
    set realname 'C:\Output\'+$FME_MacroValues(UserFileName)
} else {
    set realname '/Output/++$FME_MacroValues(UserFileName)
}
return realname
```

Note that the script must include a return statement, to return a value to the parameter.

All Scripted Parameters are private by default, and can never be published, as it would be absurd for a user to be prompted to enter Python code when a workspace runs!

Ms. Analyst says…

*Use the ‘print’ command (in Python) or ‘puts’ command (in TCL) to write from the script to the FME log file.*
Attribute Name Parameter

Sometimes an FME parameter is designed to accept either a fixed value or the value of an attribute. We call these parameters _OR_ATTR parameters, because they allow a value OR an attribute.

For example, this LabelPointReplacer allows the label to come from a fixed value, or an attribute:

![LabelPointReplacer Parameters](image)

When the end-user is required to set this FME parameter, then it’s simply a case of turning it into a linked user parameter. When the workspace is run, FME will scan the workspace to find what attributes are available to that transformer, and allow the user to select one or enter a fixed value.

**However!**

Perhaps the workspace author does not want the user to be able to enter a fixed value. They want the user to only be able to select an attribute.

In this scenario we need to create a user parameter with a special type called Attribute Name:

![Add/Edit User Parameter](image)

Now when the workspace is run, the user is permitted to select an attribute, and ONLY an attribute:
However (again)!

There is a catch to this operation. The user parameter – as the type suggests – is simply returning an attribute name; it does not return the attribute value.

If the workspace is run in this state then the LabelPointReplacer is supplied with the attribute name (not value) and uses it as the label, like so:

What the author must do is open the parameters dialog and change the parameter (either directly in the FME parameter, or via the Text Editor window) to be: `$Value(UserAttrSelection)`

The `@Value()` function replaces the name of the attribute with its actual value:
Now when the workspace is run the output will be correct:
Exercise 3  Grounds Maintenance Project

<table>
<thead>
<tr>
<th>Data</th>
<th>Parks (MapInfo TAB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>Parameterize and implement a translation log</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>Creation and use of complex User Parameters</td>
</tr>
<tr>
<td>Start Workspace</td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\Parameters-Ex3-Begin.fmw</td>
</tr>
<tr>
<td>End Workspace</td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\Parameters-Ex3-Complete.fmw</td>
</tr>
</tbody>
</table>

In a previous project (in the FME Basic Desktop training) you created a project to transform parks data by calculating the size and average size of each park.

The team who are using this now want to implement it on FME Server. At the same time they want to improve some of the functionality and implement a custom translation log.

As you created the original workspace, you are assigned to carry out these upgrades to your former work.

- Set the output to write to a folder for that user
- Ask whether to filter out dog parks
- Ask which attribute to create labels of
- Create a translation log in CSV format

1) Start Workbench

Start Workbench and open the workspace C:\FMEData2016\Workspaces\DesktopAdvanced\Parameters-Ex3-Begin.fmw

You can see that the workspace reads some MapInfo parks data, filters out dog parks, calculates park area and average area, creates labels, and writes out the data back to MapInfo.

There are two existing published parameters - one for the source dataset and one for the destination. The source dataset will never change, and we are going to create a new parameter for the destination, so delete these two parameters.

2) Add User Parameter

If we are going to write the output to a folder specific to the current user, we need to know who that user is.

So, next create a text format user parameter to ask for the user's name:
Ensure that the optional checkbox is unchecked; we want users to have to enter a value in here.

3) Add User Parameter
There are various ways we can implement the requirements here; we'll do the version that involves sharing user parameters.

So, create a new user parameter of type Folder (Existing). Uncheck BOTH of the Published and Optional fields; i.e. it will be a private parameter for us to use (not the end user) and it will be required.

Set the Name field to something like OutputFolder and for the Value browse to C:\FMEData2016\Output\Training

Dr Workbench says...

Technically, we could use a text-type parameter. The only benefit of a folder parameter is it lets us browse to the location. But since it's a private parameter, that the user will never see, it doesn't really matter. Anyway, both methods will cure our problem.

4) Set Output Location
Now let's use the two parameters we've created.

*NB: First ensure you deleted the two existing source/destination user parameters in step 1, else this step won't work!*

Locate the FME parameter for Destination MapInfo Folder in the Navigator window and double-click it to open the editing dialog.

In that dialog manually enter:
$(OutputFolder)\$(username)

Alternatively, use the text editor where you can add these by double-clicking, to reduce the chance of error.

You've basically concatenated/embedded the two user parameters into the FME parameter.

When you run the workspace you will be prompted to enter your name and then the output data will now be written to C:\FMEData2016\Output\Training\<username>

Dr Workbench says...

There are various ways we could have done this. We could have set the OutputFolder parameter to C:\FMEData2016\Output\Training\$(username) and then linked it to the FME Destination MapInfo parameter. You'll see in a moment why we didn't do that!

5) Add User Parameter

The next task is to check whether dog parks are required in the output. The Tester transformer in the workspace shows that an attribute (DogParks) will have a value of Y or N to denote its status. We need to ask the user and add their decision to the Tester.

So create a new user parameter. It will be a Choice type parameter that is not optional:

Set it up to be a simple Yes/No question of whether to include dog parks in the output.

6) Update Tester

To use the DogParks parameter open up the Tester parameters dialog. Add a second test clause:

$(DogParks) = Yes
Now when the workspace runs, if you choose not to keep dog parks they will be filtered out from the workspace. This concept - of directing features depending on the value of a user parameter - is a very useful one to be aware of.

7) Add User Parameter

OK. Next task is to allow the user to pick which attribute to use for a label.

As noted in the previous section of training, if we just publish the label parameter in the LabelPointReplacer the user will be able to enter text as well as select an attribute. We want them to have to select an attribute and not to be able to enter text.

So, create a new user parameter of type Attribute Name. This one can be optional, as the user failing to select an attribute is equivalent to saying “no labels required”:

Dr Workbench says...

Click the run button and see what appears in the list of prompts; you should see a parameter to Select the Label Attribute. But look! The parameter shows "No Attributes Available". Why is this?

This is because the list of available attributes depends on where the parameter is used. Since we have not used the parameter yet, no attributes are available!

Similarly, if we used the parameter in a location where we have attributes A and B, and also in a different location with attributes B and C, the only attribute available to the parameter is B. That’s because a parameter of this type will only show attributes that exist in all places that it is used.

8) Update LabelPointReplacer

To use the LabelAttribute parameter open up the LabelPointReplacer parameters dialog.
Remember (again from the previous section) that we can't just apply this user parameter to the Label FME parameter. That would just return the name of the attribute; we want the attribute value.

So in the Label parameter manually enter (or open the text editor and enter): 

```
@Value(${LabelAttribute})
```

Now when the workspace runs you are also prompted to select an attribute to label the parks with. If you choose no attribute, no labels are created (just point features).

9) Add Log Writer
The final task is to create a CSV format translation log. That is not too difficult to do.

Use Writer > Add Writer to add a new CSV format Writer with the following setup:

<table>
<thead>
<tr>
<th>Writer Format</th>
<th>CSV (Comma Separated Value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Writer Dataset</td>
<td>C:\FMEData2016\Output\Training</td>
</tr>
</tbody>
</table>
| Writer Parameters | Append to File: Yes (Checked)  
Output Field Names on First Line: No (Unchecked) |
| Add Feature Type(s) | CSV File Definition: Manual |

When you click OK the dialog will open for you to define the table schema.

On the General tab set the CSV File Name as `TranslationLog`:

In the User Attributes tab, define the attributes User and Date:
Click OK to close the dialog.

**10) Connect Feature Type**

We need a single record to trigger this feature type; but just one feature else we will get multiple records.

Place a Creator transformer down and connect it to the TranslationLog feature type:

![Diagram showing Creator transformer connected to TranslationLog feature type]

**11) Set Output Folder**

We should set the output location for the log to be relative to where the user files are being written.

So, locate the destination dataset parameter for the CSV Writer, right-click on it and choose Link to User Parameter:

![Diagram showing parameter linking to User Parameter]

When prompted select the OutputFolder (private) parameter that we created earlier.
Dr Workbench says...

You might be wondering what the point of that last part was. Why did we link the parameter when both were pointing to the same folder already?

The point is that now the MapInfo and CSV Writers share a parameter defining their output folder. If we wish to change where they are being written to (say when the path changes from FMEData2016 to FMEData2017) we only need to edit the private parameter to fix both Writers.

If you don’t believe me, try it and find out for yourself!

12) Set User Attribute
The very last step here is to provide values for the User and Date fields of the translation log (CSV Writer).

Right-click on the attribute called User on the feature type and choose the option to Edit Value:

In the dialog that pops up enter $(username) (you’ll be prompted for it shortly after you start to type):

This is another example of sharing user parameters. This parameter is now used here and in the MapInfo Writer name.

13) Set Date Attribute
Now add a TimeStamper transformer to provide a value to the Date attribute:
We are done! Save the workspace and then run it!

You should find your choice of data (with or without labels) written to a folder under your name, and a record of the translation added to the CSV file in the parent folder.

CONGRATULATIONS

By completing this exercise you have learned how to:

- Create a user parameter that is not optional
- Create a private user parameter
- Create a Folder type user parameter
- Embed/Concatenate user parameters inside an FME parameter
- Create a choice type user parameter
- Use a user parameter in a Tester transformer to redirect features
- Create and use an Attribute Name user parameter
- Share a user parameter by using it in multiple locations
Module Review

This chapter looked at User Parameters and some of the more advanced techniques involved in their use.

What You Should Have Learned from this Module

The following are key points to be learned from this session:

Theory

- FME is used by both workspace authors and end-users. In general, authors make use of FME parameters; end-users make use of user parameters.
- User parameters accept user input that can be used within a workspace
- User parameters can be linked to FME parameters as an indirect form of control
- Many types of user parameters exist, to allow the user to enter different types of information in a controlled way
- Parameters can be shared to reuse their content in multiple places
- Parameters can be embedded or scripted to construct complex values from simple input

FME Skills

- The ability to use FME parameters and to create user parameters
- The ability to extract information from user parameters and/or link user parameters to FME parameters
- The ability to use complex parameter types such as Choice with Alias, or Attribute Name.
- The ability to use shared, embedded, and private parameters

Further Reading

For further reading why not browse articles relating to Published Parameters on our blog?
Questions

Here are the answers to the questions in this chapter.

Miss Vector says...

Who are our two roles of FME user?

1. Creator/Inspector
2. Author/User
3. Reader/Writer
4. Maker/Consumer

Miss Vector says...

Look at the ParameterFetcher transformer. What does it do?

1. Fetches the name of a user parameter
2. Fetches the value of a user parameter
3. Fetches the type of a user parameter
4. Fetches the user a cup of tea

Of course, in most cases, you can easily use the AttributeManager instead.

Miss Vector says...

Which of the following is NOT a valid attribute type?

1. Coordinate System Name
2. Password
3. String Encoding
4. URL

Miss Vector says...

If you – as the workspace author – don’t want or require the end-user to have access to pre-linked parameters, then what can you do?

1. Delete the Reader/Writer
2. Unlink the user parameter
3. Delete the FME parameter
4. Delete the user parameter

The user parameter does not have an unlink option, and FME parameters cannot be deleted, so you would delete the user parameter. You could choose unlink on the FME parameter - but that would automatically...
delete the user parameter anyway!

Miss Vector says...

Is it possible to have a compulsory, private parameter? (i.e. both settings boxes are unchecked)

1. Yes
2. No
3. Yes, but you need to set a value immediately
4. Yes, but only for text or numeric parameters

Yes, you can have a private, compulsory parameter - but when you uncheck both boxes the default value field turns red and requires an immediate value.
Performance Considerations
Performance and FME

Performance means producing the results you need as efficiently as possible.

What do we mean by Performance?

According to Wikipedia, performance is a measure of the amount of useful work accomplished relative to the time and resources used.

In FME terms, good performance is the ability to process spatial and tabular data with the correct results (useful) as fast as possible (time) and using as few system resources – like memory – as possible.

What Can Impair Performance?

There are a number of factors – you might call them bottlenecks – that can cause FME to run slower than you might hope. Most of the content of this chapter covers how to overcome or relieve these factors.

Excessive Disk Operations

Physical disk drive throughput is relatively slow compared to other computing processes; therefore the more FME has to read and write to a physical disk drive, the more performance is impaired. The best performance comes from storing temporary data in memory, rather than on disk.

Underuse of Resources

Performance can be impaired when an FME translation is not using all of the available system resources. For example, there are few licensing limits to prevent FME from using all of the CPU cores on a system, and more memory resources are available to 64-bit FME than 32-bit.

Excessive Data Amounts

If performance is described as the amount of “useful” work, then nothing is going to impair performance more than unnecessary processing. The more excess information that gets read, and the further it is carried into the workspace, the less useful the work is and the more performance will suffer.

Jake Speedie says…

Hi, I’m Jake Speedie and I’m here to power you through this section on FME performance.

Did you know, I read that if RAM was a jet plane flying at Mach 1.5, then the equivalent disk access would be a banana slug, travelling at 0.007mph; the difference is that much!
64-Bit FME

64-bit FME is not for everyone, but used in the right place it can have tremendous benefits.

What is 64-bit FME?

A 32-bit operating system is capable of using memory addresses up to $2^{32}$. This limits them to using a maximum of 4 gigabytes of system memory.

A 64-bit operating system can use memory addresses up to $2^{64}$. This allows them to use a theoretical maximum of 16 billion gigabytes of memory! In practice, a 64-bit system uses nowhere near that much memory, but it still allows 64-bit software to access a much greater amount of additional memory than 32-bit.

64-bit FME is a version of FME specifically designed to take advantage of a 64-bit operating system. It is well-suited to processing very large amounts of data, due to its ability to use a greater amount of memory.

---

NEW

The 64-bit version of FME 2016 has greatly improved performance (on 64-bit platforms) with an ability to use much more of the available system memory.

---

What are the Disadvantages of 64-bit?

There are a number of disadvantages to using 64-bit FME.

Operating System

64-bit FME requires a 64-bit operating system and the hardware to support it. On a 32-bit operating system you would only be able to run 32-bit FME.

Format Limitations

Not every format in FME is supported on 64-bit. Usually that’s because the format itself – or its proprietary software – isn’t available on 64-bit platforms. In that situation you would need to use 32-bit FME.

64-bit Clients

Not a limitation, but 64-bit does require care in selecting the correct database clients; for example when reading 64-bit Oracle you’ll need to install a special 64-bit client for FME to be able to connect.

---

Jake Speedie says…

The Safe Software web site has a list of supported formats with information on which are supported on 64-bit.
Miss Vector says…

Let’s see if you can figure out the one false statement from these facts about 64-bit FME:

1. 32-bit Windows can use only 32-bit FME. 64-bit Windows can use either 32-bit or 64-bit FME
2. You can install both 32-bit and 64-bit FME on a 64-bit computer, and use either one as necessary
3. You can install 32-bit and 64-bit engines on the same FME Server core
4. A workspace authored on 32-bit FME cannot be run on a 64-bit engine
Log File Interpretation

The FME log file is your best friend for assessing performance. It tells you how long a translation took, where the time went, and how well FME was able to use the available system resources.

Log Messages

The first thing to notice is that the log is made up of a number of messages, each of which consists of a number of fields:

- Absolute Date [Optional]
- Absolute Time [Optional]
- Cumulative Time (for translation)
- Elapsed Time (for this message)
- Message Type
- Message

The Message Type field tells us the nature of the information. It will be one of the following:

- ERROR: An error in the translation that usually requires FME to terminate processing.
- WARN: A warning that signifies a problem that is not sufficient to terminate processing.
- INFORM: An information message relating a non-error item.
- STAT: A message on translation statistics such as the number of features processed.

Configuring the Log Window

There are a number of options to adjust the log file and what is displayed. To access these select Tools > FME Options > Translation within FME Workbench.

Some of the most important options are as follows:

Log Timestamps

With this option turned on each message in the log window gets stamped with the time and date it occurred. Timestamps are an invaluable aid to assessing performance and should be kept on in most cases.
**Log Full Precision**

Logging full precision means any coordinates that are reported will be listed to their full precision. This is only of real use when trying to compare coordinates between different features.

**Log Message Filters**

Filtering options allow each type of message to be turned on or off in the log window. It can be particularly useful to turn off INFORM and STAT messages in order to make it easier to spot ERRORs and WARNs; however it does appear strange at first to run a translation and not see the usual stream of information!

**Log Debug**

The Log Debug option turns on a series of extra log messages that are usually hidden from the user. Not only will a lot of the underlying mapping file be exposed, there will also be a number of ERROR messages labelled BADNEWS.

These messages can help during debugging, but it’s very unlikely you’ll want to keep them turned on in general FME use. Many of the BADNEWS messages are “errors” that FME has trapped and kept to itself (like an end-of-file message).
Deconstructing the Log File

This section looks in more detail at each part of an FME log.

Jake Speedie says...

Being able to interpret a log file is vital for performance tuning. If you can’t understand what FME is doing then there isn’t much chance you’ll be able to improve upon it!

The overall structure of an FME log is basically four different sections.

- Command-line statement
- Configuration and setup information
- The translation and transformation itself
- A summary of the translation

Command Line Statement

At the very top of a log file appears the command line statement. This is the command that FME Workbench is using to run the translation:

Command-line to run this workspace:

```
C:\apps\FME2016.0\fme.exe C:\FMEData2016\ParksTranslation.fmw
   --SourceDataset_MITAB "C:\FMEData2016\Data\Parks\Parks.tab"
   --DestDataset_GML "C:\FMEData2016\Data\Parks\Parks.gml"
```

In terms of performance, this section doesn’t tell us much. However, it is useful to confirm what version and which instance of FME is running, particularly when you have several versions installed.

This section also tells us what published parameters are in the workspace and what their values are.

Perhaps the most useful part of this section is that you can just copy and paste this statement to run the workspace through a command line, or using a batch file.
Configuration and Setup Information

Configuration and setup messages in the FME log tell us vital information about FME's version and configuration, the system resources and how FME intends to use them, and what system paths are being used.

For example, here you can see which version of FME is being used, its license type, and the machine name. If you do have multiple FME versions installed, here's where you can check to ensure the correct one is running.

Further on we can see that the system has nearly 108GB of free space and 4 GB of virtual memory (the maximum possible on 32-bit). We can also see what operating system we are running FME on and what the current language and encoding settings are:

System Status: 81.97 GB of disk space available in the FME temporary folder
(C:\Users\imark\AppData\Local\Temp)
System Status: 4.00 GB of virtual memory available
Operating System: Microsoft Windows 7 64-bit Service Pack 1 (Build 7601)
FME Platform: WIN32
Locale: en_CA
Code Page: 1252 (ANSI - Latin I)

Later in the log is important information about the system resources and FME's memory management:

Process limits are 24.00 GB of physical memory and 4.00 GB of address space
Start freeing memory when process usage exceeds 2.83 GB of memory or 1.41 GB of address space
Stop freeing memory when process usage is below 2.12 GB of memory and 2.56 GB of address space

In this case there is a limit of 4GB memory per process, indicative of 32-bit processing, out of 24GB of total memory for the machine. The following numbers indicate how FME will manage memory resources. It will use nearly 3GB of memory and then it will start to release memory by caching features to disk. This caching will stop once memory use is less than about 2GB.

This way FME will perform to its potential automatically, while not taking so much memory that the system may fail or other processes on the system would suffer.

Of course, when run on 64-bit FME, the numbers will be slightly different, illustrating the benefits of this platform:

Process limits are 24.00 GB of physical memory and 8.00 TB of address space
Start freeing memory when process usage exceeds 71.99 GB of memory or 8.00 TB of address space
Stop freeing memory when process usage is below 53.99 GB of memory and 6.00 TB of address space

Here FME will not have to start caching data to disk until nearly 72GB of memory has been used!

Temporary Folder

For performance tuning one of the most important parts of the log reports the temporary folder being used. When memory resources become low and FME starts to cache to disk, the temporary folder is where it will write data to.
Firstly it's important to ensure this folder does have enough temporary disk space for the translation being carried out. Depending on the workspace and the transformations being carried out, this may be many times greater than the size of the original dataset.

Secondly it's useful if the disk being written to is both fast and unused by any other process. It will not, for example, help performance to share the temporary disk with the operating system.
Translation and Transformation Statements

The main body of the FME log concerns the translation and transformation of data. This section often appears confusing to new users, until they understand how FME operates.

Firstly, there can be several parallel streams of processing in a workspace. If you are not specifically setting the order of these streams, the order of the log will be similarly indecisive.

Secondly, there is the issue of transformer type.

As you might know, some transformers operate on a feature-by-feature basis, while others work on a group of features at a time.

When a series of transformers are all feature-based, FME works by pushing features through on an individual basis, not as a group. For example, the first feature is read and then processed by each transformer in turn. Then the next feature is read and processed by each transformer, and so on.

This makes for a log file that is harder to understand, because you don’t see one specific entry for each Reader or transformer in turn. Nor do you see a message for each feature. Instead, a cumulative time is calculated and output at regular time-based intervals.

Key Messages

There are several key messages that may appear in a log that signify specific events in a translation.

Emptying factory pipeline

This message signifies that reading of data is at an end. Because of how FME processes data, this message may appear near the beginning of a translation or near the end. But after this point the source datasets are closed and only transformation and/or writing will take place.

By checking the timestamp for this message you can get an approximate value for how long the workspace took to read the source data.

Unexpected Input Remover

This message denotes where features read by a Reader are tested against the available feature types (and merge filters) to see if they will be allowed to pass into the workspace.

This message may indicate a performance issue as it highlights potentially unnecessary work, where features are being read but not used in the translation.

ResourceManager: Optimizing Memory Usage. Please wait...

This message alerts you to the fact that the memory management limits mentioned in the configuration part of the log have been reached, and that FME is having to work around the issue, possibly by caching data to disk.
If you see this message frequently, then it’s time to either redesign your translation or switch to a 64-bit setup. It is a key indicator that a lack of memory is affecting performance.

**Miss Vector says…**

*How would you find the Emptying Factory Pipeline setting in the log window?*

1. **Looking for the color:** it is the only message highlighted in red
2. **Looking for the message type:** it is the first message of type STATS
3. **By using the log search tools**
4. **By opening Tools > FME Options to turn off all other messages**
Translation Summary

The final part of a log file includes a report of the number of features read and written:

```
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Features Read Summary</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>Parks</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>Total Features Read</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>Parks</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>Total Features Written</td>
</tr>
</tbody>
</table>
```

More importantly, from a performance point of view, it includes a brief report of the time taken by the translation and the amount of memory used:

Translation was SUCCESSFUL with 0 warning(s) (80 feature(s) output)
FME Session Duration: 0.8 seconds. (CPU: 0.5s user, 0.2s system)
END - ProcessID: 41044, peak process memory usage: 94844 kB,
current process memory usage: 94744 kB

Peak memory usage is an important statistic. It signifies how hard FME is having to work. If this number can be reduced performance will often improve

Written Features

One of the most misinterpreted statistics in an FME log is the number of features written.

What this really means is “the number of features sent to the Writer”. It doesn’t always mean the same number of features actually gets written to the output dataset, or that the output dataset will contain only that number of features.

For example, in the above screenshot 80 features are reported as sent to an (Esri Shape) Writer. However, a warning earlier in the log tells us:

```
STATS |---------------------------------------------------------------|
WARN   |Rejected 80 output features                                   |
INFORM |---------------------------------------------------------------|
```

So in reality, the Writer rejected all of these features, in this case because their geometry was invalid. The Writer was set up to write line features, yet these are polygons.

Similarly, a format may have geometry limitations that cause the output dataset to be slightly different to the numbers logged.

For example, MicroStation DGN format has a limit on vertex numbers for each element (feature). If the MicroStation Writer receives a feature with too many vertices it will split that feature into multiple MicroStation features (elements in MicroStation speak) to avoid going over the vertex limit.
Thus, the number of features that actually appear in the dataset can be different to the number of features logged as being sent to the Writer!

**WARNING**

Currently, features read or written by the FeatureReader/FeatureWriter transformers are not included in the summary at the end of the log. To find this information you would need to examine the feature counts displayed on the connections in or out of those transformers.
# Log Timings

The timestamps in FME logs are invaluable for assessing performance.

Remember, the structure of the log is this:

<table>
<thead>
<tr>
<th>Absolute Time</th>
<th>Cumulative Time</th>
<th>Time for Operation</th>
<th>Message Content</th>
</tr>
</thead>
</table>

Time for Operation is the amount of time spent on that step of the translation, and Cumulative Time is the sum of those operations.

Although cumulative time is the amount of time the FME process has spent translating or transforming data, it does NOT include the amount of time spent waiting for other processes to do their work.

For example, examine this section of log timings:

| 2016-03-10 14:43:06 | 8.5 | 0.0 |
| 2016-03-10 14:43:13 | 8.8 | 0.3 |
| 2016-03-10 14:46:29 | 18.0 | 9.1 |
| 2016-03-10 14:49:29 | 25.8 | 7.9 |

The difference between the start absolute time (14:43) and the finish absolute time (14:49) is over six minutes. However, FME is only reporting 25.8 seconds of processing time!

The “lost” time is down to external processes that FME was waiting on.

For example, maybe the message content would look like this:

| 2016-03-10 14:43:06 | 8.5 | 0.0 | Preparing SQL query |
| 2016-03-10 14:43:13 | 8.8 | 0.3 | Sending SQL query |
| 2016-03-10 14:46:29 | 18.0 | 9.1 | Received initial database response |
| 2016-03-10 14:49:29 | 25.8 | 7.9 | Received data from database query |

Now we can see that it was a database query that FME was waiting for. The more the query was not well-formed or the database not well-structured, the longer the time difference would be.

Similarly, reading/writing data from/to a disk can account of missing time.
Exercise 1  

Cell Phone Signal Processing

| Data               | City Neighborhoods (Google KML)  
| Cell Phone Signals (CSV) |
|---------------------|---------------------------------|
| Overall Goal        | Analyze and improve the workspace performance |
| Demonstrates        | Interpreting an FME log file |
| Start Workspace     | C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex1-Begin.fmw  
|                     | C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex1-Begin-Logfile.log |
| End Workspace       | None |

A close friend and fellow FME user works for a cell phone company. His current FME project is to analyze cell phone signals.

His source dataset contains a series of recordings that show how strong the cell signal is at different locations.

The project is to filter out locations that receive a really poor signal, tag them with the neighborhood they belong to – to show which neighborhoods have poor coverage – and write the rest of the data out as a series of attribute-less data points.

He has created a prototype workspace that processes the data and produces the correct results.

However, the workspace runs perhaps a little slower than it could, which is bad news when this is just the prototype and he wishes to eventually run it on the entire country’s cell data.

He asks for our help and sends us his workspace, the log file, and the source datasets. First of all let’s check the workspace and deconstruct its log to find out what is happening.

1) Start Workbench

Open the workspace C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex1-Begin.fmw

Notice there are two key feature types: one for the cell signals in CSV format, the other for the Neighborhood boundaries in KML format. They each have their own set of attributes:
Jake Speedie says…

The Cloner transformer is there to increase the amount of data so that improvements to the workspace are more visible. You can also use the Number of Copies parameter to increase/reduce the amount of data to suit your computer, which may be faster/slower than the one used here. Otherwise, just pretend it isn’t there!

.1 UPDATE

The Cloner transformer in this exercise gained a <Rejected> port in FME2016.1

There are three additional transformers:

The Clipper is used to copy the neighborhood name onto each reading, and the Tester is used to filter out cell signals that are below strength. The Logger is recording cell signals to the log window/file.

Finally there are two Writers, each with a single feature type:

Don’t run the workspace yet – we don’t know how long it might take!

2) Open Log File

Besides the workspace we also have a copy of the log file, so start up a text editor and open up the log for inspection. You can find
it at: C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex1-Begin-Logfile.log

Let’s look for some of the indicators as to how this workspace is performing.

Firstly the configuration section tells us that the user is working with FME2016 with a fixed license, Smallworld Edition.

This is the release version of FME2016.0. Looking on the Safe web site I can see there is a newer update to that (FME 2016.0.1.1, build 16177). There might be a performance improvement in there (we could check the What’s New file to see) but there's unlikely to be much advantage to upgrading; certainly not as much as if the user was using FME2011, for example.

Now look for the more important performance indicators:

There is adequate of disk space and plenty of memory. The workspace is being run on a 64-bit platform but only using 32-bit FME, which is a bit disappointing as there is 25 GB of memory.

Let’s skip to the foot of the log now and see how long it took to run and how much memory was consumed at the peak:

One concern is that the translation took approximately 11 minutes but CPU processing is only 306 seconds (5 minutes). We're obviously losing time somewhere.

The peak memory is also worrying. It’s close to – if not above – the amount required for FME to start releasing memory and reorganizing data. In fact if we scan the log content we can find many messages of this type:

So FME had to start optimizing memory usage at multiple points. It probably resulted in some disk caching, and that might be the cause of the lost time.
3) Run Workspace

If you would like to do so, run the workspace. Obviously you can expect that it will take approximately 11 minutes to complete on the average 32-bit machine. Do your results match what occurred in the above log file?

Jake Speedie says...

*It’s important to note that, because of different machine specifications, you may get vastly different results to this log. That’s fine; the important part is the techniques used, not the exact timings. Remember you can adjust the Cloner transformer to account for differences.*

If you have access to 64-bit FME, then why not try it to see if the performance improves. Notice that there’s no problem in opening the same workspace in 32-bit and 64-bit FME. Workbench is the same; it’s just how the workspace is run that is different.

On my machine the use of 64-bit FME improves the performance considerably:

INFORM|FME Session Duration: 5 minutes 19.3 seconds. (CPU: 295.7s user, 18.7s system)
INFORM|END - ProcessID: 51788, peak process memory usage: 9973288 kB, current process memory usage: 103544 kB
Translation was SUCCESSFUL

Notice how the time used is only 50% of what it was, and the time lost to writing to the file system is reduced by a large amount. Peak memory usage has increased, indicating that 32-bit FME was being restricted by a lack of available memory.

Jake Speedie says...

*In case you’re interested, the cell phone power values appear to be in dBm units – which is Decibel-Milliwatts. So now you know!*

CONGRATULATIONS

*By completing this exercise you have learned how to:*
- Inspect a workspace to find out what it does
- Deconstruct a log file to assess performance
Reader and Writer Optimization

Reading and writing data is obviously a major part of most workspaces and so being able to optimize those steps can improve performance greatly.

But before optimizing reading and writing, we have to be able to assess what the current level of performance is.
Assessing Reader Performance

To be able to improve the efficiency of a Reader requires an estimate of how well it is working in the first instance; yet this can be hard to separate out in a workspace that is also transforming data.

The key message that signifies reading is complete is “Emptying Factory Pipeline”. Here, for example, reading of the data finished after 144 seconds of processing (of course the actual elapsed time might be longer if FME was waiting for a database or the file system to respond):

```
2014-12-08 10:46:52| 144.1| 0.8|INFORM|Emptying factory pipeline
```

False Reader Performance

Sometimes the time for that message can be misleading. Take this workspace that reads a set of address points and finds their nearest neighbor in a second dataset:

According to the log file the data took 27.5 seconds to read:

```
2014-12-08 13:13:52| 27.5| 0.0|INFORM|Emptying factory pipeline
```

And in total the whole workspace took 27.6 seconds to run:

```
INFORM|Translation was SUCCESSFUL with 0 warning(s)
FME Session Duration: 27.6 seconds. (CPU: 26.8 user, 0.7 system)
```

But that doesn’t seem right. How could it be that the data took 27.5 seconds to read but only 0.1 seconds to process?

In fact, this is because FME was processing the data at the same time as it was reading it. It won’t read the entire dataset before processing, because that would be inefficient. So although reading didn’t finish for 27.5 seconds, during that time FME was already processing the features it had read and the 0.1 seconds is the time it took to handle the final feature and end the process.

True Reader Performance

So, how can we assess the true amount of time taken to read the data? The answer is to disable all transformation and simply run the reading part of the workspace:
Now when the workspace is run it is reading the data only, with no transformation, and the factory pipeline message appears after a mere 5.4 seconds:

```
2014-12-08 13:15:12| 5.4| 0.0|INFORM|Emptying factory pipeline
```

So from this we can assess that the data reading takes only 5.4 seconds out of the 27.6 total.

This is also important to know during processing, because the log window can also give the impression that the workspace is still reading (and is therefore yet to process) data. For example:

```
2016-03-08 13:24:54| 4213.3| 1904.1|INFORM|Reading source feature #5000
2016-03-08 14:04:58| 6617.3| 2494.4|INFORM|Reading source feature #10000
2016-03-08 14:52:03| 9501.4| 2884.9|INFORM|Reading source feature #15000
```

Here a very complex workspace is taking hours to complete. At first glance you might mistakenly believe the data was still being read in preparation for processing, because “Reading source feature” messages were still appearing. In fact, data is being processed simultaneously with the reading.

---

**Jake Speedie says…**

*Remember the exact structure of the log will depend greatly on whether the transformers being used are feature-based or group-based. A group-based transformer will hoard features until it is ready to process them all, and this will look very different in the log to a feature-based transformer that processes features one at a time.*
Improving Reader Performance

The most important method to improve reading performance is to minimize the amount of data that is being read. As already mentioned, reading excess features counts as unnecessary work and is therefore inefficient.

Jake Speedie says...

I walk into a restaurant and - without even looking at the menu - say to the waiter: "I'll have one of everything". It certainly makes ordering quicker, but my food takes much longer to be delivered, and is very much more expensive!

If that sounds ridiculous, consider that FME is much the same as a restaurant. If you read the entire contents of a dataset, when you only need a part of that data, then you're wasting resources and slowing down the process. Not to mention putting stress on the CPU (Chef Processing Unit).

Filtering Input

For example, this workspace reads nearly 14,000 features, but immediately discards all except 400 of them (ones where the owner's name begins with "C":

In this scenario, if possible, it would be much more efficient to simply just read those 400 features. All formats have various sets of parameters that speed up feature reading by filtering the amount of data being read.
The first of these – search envelope – defines the data to read as a geographic area. Then only that area of data needs to be read. These parameters are available on every spatial data Reader, but have the most effect when the source data is spatially indexed. Then the query is being carried out at its most efficient.

Similarly, there are a number of parameters designed to let the user define how many features to read. These parameters include the ability to define a maximum number of features to read, and what feature to start at. There is also a parameter that defines which feature types (layers or tables) should be read. By using these judiciously, the amount of data being read can be reduced and the translation sped up.

Other formats – particularly databases – have additional clauses that can help reduce the data flow:

![Feature Type Properties](image)

Here, for example, this Geodatabase Reader has a ‘WHERE Clause’ parameter that applies the “owner name begins with ‘C’ test” in a way that is more efficient than reading the entire contents of a large table and using a Tester transformer.

In short, when you want to filter source data, and can use a specific Reader parameter to do so, it is more efficient than reading all of the source data and then filtering it with a transformer.

**Jake Speedie says…**

*Quick reading tip for CSV data. Use the PointCloud XYZ format to read CSV data, then the PointCloudCoercer transformer to turn it into points. It works way faster than just using the CSV format! But look out for the same sort of performance improvements for CSV in FME2017.*
Excess Feature Types

Another potential bottleneck - specifically for formats with a table list – is the case where you have more feature types than are necessary.

Here the user has added a number of tables to their PostGIS database Reader:

![Feature Types](image)

However, if you look at the workspace, many of these tables are not even connected to anything. The unconnected tables are still being read but the data is being ignored:

![Workspace](image)

Presumably the user added the tables for some reason, but then decided they did not need them, In that case they should delete the feature type from the FME workspace. Then the table will not be read and performance will improve.

Dangling Readers

Another problem scenario - this time for file-based datasets - is a dangling Reader. This is where a user deletes all of the feature types, but not the underlying Reader:
Here the user added a Reader to read an AutoCAD dataset. The feature type (layer) definitions were deleted from the workspace, but the Reader remains.

In this case, when the workspace is run, all the data is still read from the file, but then immediately discarded as "unexpected input."

Remember, any extra data that is read – of whatever amount – takes time and resources to read, and impacts performance. In this case the user should have deleted the Reader too.

TIP

One other obvious way to improve reading performance is to upgrade the underlying system to minimize the amount of time FME spends waiting for a response.

For example, tune your database so that it responds to queries quicker, and use a solid-state hard drive so that file datasets can be read quicker.
Assessing Writer Performance

As with Readers, you can't improve the performance of a Writer unless you can first assess how well it is already performing. But assessing the speed of writing has the same complexity as reading: FME starts writing data as soon as it becomes available, and doesn't necessarily wait until processing is done.

False Writer Performance

Take the previous workspace, which read a set of address points and found their nearest neighbor, but now has a Writer too:

According to the log file, we find that the output dataset is open for writing before the source dataset has even finished being read!

If you examined this log you would get quite a false representation of how well the Writer is performing.

True Writer Performance

So, how can we assess Writer performance? Plainly this is different to Readers. If you isolate the Writers by disabling everything else, there won't be any data to write!

However, it can be done with a two-step process. Firstly you would add a Recorder transformer – after all other transformation has taken place – to preserve the data in FFS format at the moment it is about to be written:
Now replace the Recorder with a Player transformer – to re-read the preserved FFS data – and disable everything else up to that point:

Now the data will be played back into the workspace, and is followed up by being written to the output.

In this example the "Emptying Factory Pipeline" message appeared after 7.2 seconds and the translation took 8.9 seconds in total, indicating 1.7 seconds was spent writing data.

Jake Speedie says…

Be aware that fanouts will complicate your view of the log, not to mention slow the process somewhat.

For example, if I do a Dataset Fanout on the above workspace, FME creates a separate Writer for each fanned-out dataset. Firstly this causes a performance hit – because FME has to cache all the output data and create multiple Writers – and additionally I get a section of log for each output dataset.
Improving Writer Performance

There are various ways to speed up writing data. Compared to reading, tuning the underlying systems is a more important improvement, whereas the number of features is less important as it’s much harder to write extra data unintentionally.

File System Improvements

If you are writing to a file system then make sure the disk is fast and responsive – use solid state drives – and that the operating system is not busy writing other files at the same time, as the latter could cause a significant bottleneck.

Also check if you are using RAID as some configurations need multiple writes and can definitely slow things down.

Database Improvements

If you are writing to a database, then existing indexes and joins can cause the process to be slower than expected. In many cases it will be quicker to drop an index, write the data, and then recreate the index.

More information on database performance with FME comes in a later section.

Multiple Writers

Perhaps the most important technique for improving Writer performance involves the scenario where a workspace has multiple Writers.

In short, it’s important to get the writers into the optimum order, to ensure that the Writer that is to receive the largest amount of data is written first.

The reasoning behind this is that the first Writer in a workspace starts to write data as soon as it is received. Other writers cache theirs until they are ready to start writing.

Therefore, if the largest amount of data is written immediately, lesser amounts of data have to be written to, and stored in, a cache.

This can improve performance tremendously, particularly when the translation is especially unbalanced; for example one million features go to one Writer, and only ten features go to another.

Jake Speedie says...

Think of it like an airport. It’s more efficient when you load the busiest flights first, because it empties the terminal waiting areas quicker. For more information see this FME Evangelist article.

Setting Writer Order

There are two ways to affect the order that writing occurs in.

Firstly each Writer is listed in the Navigator window in Workbench and can be re-ordered by moving them up and down in the list in the Navigator window:
The first Writer in the list is the one that is executed first, therefore it should be the one to receive the most data.

The second method is to use a workspace parameter called Order Writers By:

This parameter can be left to Position in Workbench Navigator in which case the order of Writers as defined in the Navigator takes priority. Alternatively it can be set to First Feature Written. In that case the Writer that receives the first feature will be the first to start writing data.

**Miss Vector says…**

*Given this screenshot, which should we make the first Writer in this workspace?*
1. A
2. B
3. C
4. Don’t know!

Improving Writer Performance
If you recall, a close friend who works for a cell phone company has asked for your help in improving his FME workspace's performance. His project is to analyze cell phone signals; to filter out locations that receive a really poor signal, tag them with the neighborhood they belong to – to show which neighborhoods have poor coverage.

Now we’ve deconstructed the log and uncovered which areas are of concern, let’s start to clean up any performance issues with the Readers and Writers.

1) Start Workbench
Start Workbench and open the workspace C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex2-Begin.fmw (or stick with Performance-Ex1-Begin.fmw if you have it open – it’s the same workspace).

The first thing we should do is ensure we’re logging all the required timestamps. So select Tools > FME Options from the menubar in Workbench.

Click on the Translation icon (a green "play" button). Ensure that the Log File Defaults has “Log timestamp information” turned on.

Since the workspace doesn’t fail – it’s just a little slow – we aren’t debugging anything so make sure Log Debug is turned off.

2) Assess Reader Performance
Let’s first assess how well the Readers are doing their job. It might be that they aren’t really the bottleneck in our workspace.

First check the original log file for the Emptying Factory Pipeline message.

It occurs after 42.3 seconds, but let’s see if that’s accurate.
Select all of the objects after the feature types (i.e. transformers and Writer feature types). Press Ctrl+E to disable them (Ctrl+E is a toggle to Enable/Disable):

```
2016-03-16 14:25:01| 3.7| 0.0|INFORM|FME Session Duration: 4.1 seconds. (CPU: 3.4s user, 0.2s system)
2016-03-16 14:25:01| 3.7| 0.0|INFORM|END - ProcessID: 51976, peak process memory usage: 73304 kB
```

Run the workspace. The data will be read, but not processed or written.

Check the time taken to do this. On my machine the result is this:

So it’s actually reading the data very quickly - only 4.1 seconds. So the figure in the original log is showing how FME is processing the data as it is being read. Still, as a matter of best practice we should check to see if we can find any improvements to be made.

3) Check Data Filtering
The workspace is filtering data with a Tester. Is there any way that we could improve on our reading time by carrying out this test directly on the source data?

Firstly, we want all the data spatially, so there’s no use in setting any Search Envelope parameters. In any case the CSV data is not – initially – spatial and has no such parameters.

Secondly, could we apply that test to the data as it is being read? Well, neither Reader has a WHERE clause field, as neither is a database. The CSV Reader does have filter parameters:

- **Filter Enabled**: No
- **Filter Parameters**: <not set>
- **Advanced**

...but a glance at the Readers and Writers Manual shows this is a regular expression, not really suitable for filtering by an arithmetic expression.
4) Check Other Reader Issues

Are there any other issues with the Readers that might be slowing performance? Yes, there are!

Firstly, notice that the KML Reader that is reading the neighborhood data also includes a whole number of feature types that we aren’t interested in.

The only feature type we need is Neighborhoods, and that’s already connected into the Clipper transformer.

All the other feature types are producing data we don’t need. They might not be slowing us much, but they certainly won’t be speeding up the translation.

So, we should select all unconnected feature types on the canvas and delete them. The quickest way to do this is select Tools > Remove Unattached from the menubar:

...and click OK to accept the removal of all unattached feature types.

Running this now I get:

```
2016-03-16 14:32:22 | 3.7 | 0.0|INFORM|FME Session Duration: 3.7 seconds. (CPU: 3.5s user, 0.2s system)
2016-03-16 14:32:22 | 3.7 | 0.0|INFORM|END - ProcessID: 52636, peak process memory usage: 71888 kB
```

In this exercise it's not a big difference, but it's always worth checking.

5) Assess Writer Performance

You can - if you like - check the writer performance (using the two-step technique in the notes). However, for me the result without writing is this:

```
INFORM|FME Session Duration: 4 minutes 30.6 seconds. (CPU: 245.2s user, 22.6s system)
INFORM|END - ProcessID: 51544, peak process memory usage: 1776868 kB
```

So we can say it's taking nearly seven minutes to write the data. Reading it back with a Player and writing the data confirms this figure. It's not quick, so let's look for the most obvious improvement: the order of the Writers.
6) Set Writer Order

As mentioned, the best way to improve Writer performance is to ensure the Writer receiving the largest amount of data appears first in the Navigator window.

In this workspace there are two Writers. One writes the problem (ColdSpot) locations. The other writes the good locations:

It's obvious which Writer is handling the most data, and it is not currently the first to be processed, so let's fix that. Click on the GoodLocations Writer in the Navigator window and drag it above the Cold Spots writer:

Also make sure the advanced workspace parameter Order Writers By is set to Position in Workbench Navigator.

Re-enable any components of the workspace that were disabled, and run the full translation again. Remember, the original log reports the following results:

```
INFO|FME Session Duration: 11 minutes 6.5 seconds. (CPU: 306.7s user, 37.8s system)
INFO|END - ProcessID: 50764, peak process memory usage: 2966368 kB
```

Now, on my computer, I get the following:

```
INFO|FME Session Duration: 4 minutes 1.9 seconds. (CPU: 219.7s user, 19.4s system)
INFO|END - ProcessID: 46312, peak process memory usage: 1776304 kB
```

That's way better. I've reduced the time taken by 60% from the original. Additionally, peak memory use has dropped by nearly 50%! If you look carefully at the log you will (probably) find that the message "Optimizing Memory Usage" is no longer present, meaning FME is no longer starved of enough memory.

**CONGRATULATIONS**

By completing this exercise you have learned how to:
- Assess Reader performance
- Check Readers for performance-improving parameters
- Check Readers for performance-impacting issues
- Assess Writer performance
- Improve Writer performance by ordering Writers
Transformation Optimization

As with Readers and Writers, you can’t make performance improvements unless you can accurately interpret a log file and measure a baseline performance for your translations.

There are two aspects to transformation performance: the time taken and the amount of memory used.

Transformation Length

Assessing the time taken in transformation requires a two-step process.

First disable Writers and run the translation, taking a note of the elapsed time:

Then disable the transformers too and run the workspace again, to calculate the time taken to read the data only. The difference in elapsed time between reading the data and reading/transforming the data, is the elapsed transformation type.

It’s important not to add an Inspector or Logger transformer to the end to see what is happening to the output. This will only slow the translation down and give you a false measure. You must also be sure to disable the actual Writer, and not just the feature types or connections to them.

The only Writer that is useful in this scenario is the Null format Writer. This causes a Writer to be present, but it does nothing except to log features and then discard them. The benefit is improved logging of feature counts, but without any data having to be written.

With the above workspace, I now know it took 5.4 seconds to read the data, and the whole process (excluding writing) took 28.2 seconds, so I can infer that the transformation part takes 22.8 seconds.

With the Reader and Writer figures as well, I now have a complete breakdown of how long each section of my workspace takes.

Jake Speedie says…

In a larger workspace you could isolate different sections by disabling different connections, and therefore determine the performance of individual parts of the workspace.

Transformation Memory Use
The time taken to carry out a translation is only one aspect of performance. Another important aspect is the amount of memory used during processing. You can't tell how much memory each individual transformer used, but you can see the maximum memory used during a translation by examining the very foot of the log file:

```
INFORM| Translation was SUCCESSFUL with 0 warning(s) (13597 feature(s) output)
INFORM| FME Session Duration: 28.3 seconds. (CPU: 27.3s user, 0.6s system)
INFORM| END - ProcessID: 28336, peak process memory usage: 178388 kb
```

For performance tuning, the idea is to reduce this number, as the more memory used the more chance there is of laborious disk caching taking place.
Improving Transformation Performance

In most cases, slow, memory-consuming translations are caused by group-based transformers.

Remember that in feature-based transformation a transformer performs an operation on a feature-by-feature basis where a single feature at a time is processed. Such a transformation only takes as much memory as is necessary to store a single feature.

However, a group-based transformer performs an operation on a group or collection of features and it takes as much memory as is necessary to store all features of the group!

It is this grouping of data that causes performance degradation.

Jake Speedie says....

You’ll get better performance when you put the least amount of data into a group-based transformer as possible. For example, put feature-based filter transformers BEFORE the group-based process, not after it (see following exercise). Another technique is to make group-based transformers more feature-based...

Making Group-based Transformers more Feature-based

Obviously, when a group-based transformer is needed, then it must be used. That is inevitable. However, many group-based transformers have a parameter that, in effect, makes them more feature-based.

One common parameter is called "Input is Ordered by Group" and appears near the Group By parameter in most transformer dialogs:

The condition for applying these is that the groups of features are pre-sorted into their groups. When this is the case, and I can set this parameter to Yes, then FME is able to process the data more efficiently.

For example, in the above screenshot the user is using postal code as a group-by parameter (i.e. each address looks for it’s nearest neighbor in the same postcode). If the incoming data is already sorted in order of postcode then the user can set the Input Is Ordered parameter to yes and allow FME to treat this more like a feature-based transformer.

Jake Speedie says...

Let’s think back to the airport departure gate boarding passengers. Most airlines first call passengers who require assistance boarding, then passengers with children, business-class passengers, and finally economy passengers (starting with passengers at the front). That’s because it’s easier to board passengers when they are sorted into similar groups, and the same applies to FME. When passengers (or spatial
features) arrive in a random order it’s not as simple to handle them.

Besides the “Input is Ordered by Group” parameter, some transformers have their own, unique, parameters for performance improvements.

For example, the NeighborFinder expects two sets of data: Bases and Candidates. By default FME caches all incoming Bases and Candidates because it needs to be sure it has ALL of the candidates before it can process any bases.

But, if it knows the candidate features will arrive first (i.e. the first Base feature signifies the end of the Candidates) then it doesn’t need to cache Base features. It can process them immediately because it knows there are no more candidates that it could match against.

The user specifies that this is true using the parameter Candidates First:

Look at this log file for a workspace that uses a NeighborFinder. By default it looks like this:

```
Translation was SUCCESSFUL with 0 warning(s) (13597 feature(s) output)
FME Session Duration: 29.6 seconds. (CPU: 27.7s user, 1.5s system)
END - ProcessID: 28540, peak process memory usage: 231756 kb
```

With Candidates First turned on it looks like this:

```
Translation was SUCCESSFUL with 0 warning(s) (13597 feature(s) output)
FME Session Duration: 28.4 seconds. (CPU: 27.4s user, 0.8s system)
END - ProcessID: 26429, peak process memory usage: 178412 kb
```

It’s about 5% faster than before, but more importantly it’s used nearly 25% less memory!

But how does a user ensure the Candidate features arrive first? Well, like Writers you can change the order of Readers in the Navigator, so that the Reader at the top of the list is read first.

It doesn’t improve performance per se, but it does let you apply performance-improving parameters like this.

---

**Miss Vector says…**

Which of these transformers have group-related parameters for improving performance (pick all that apply and see if you can get the answers without looking at the transformers):

1. StatisticsCalculator
2. SpikeRemover
<table>
<thead>
<tr>
<th>3. PointCloudCombiner</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. FeatureMerger</td>
</tr>
</tbody>
</table>
Transformer Selection

If you’ve used FME for any length of time, you’ll know that it’s possible to do almost any task in several different ways.

For example, if you were isolating all road features that pass through a park you could use either the Clipper, like so:

Or the LineOnAreaOverlayer, with a test for _overlaps => 1, like so:

The performance for the LineOnAreaOverlayer would be this:

Translation was SUCCESSFUL with 0 warning(s) (134 feature(s) output)
FME Session Duration: 3 minutes 57.6 seconds. (CPU: 196.2s user, 37.6s system)
END - ProcessID: 38036, peak process memory usage: 3129576 kB

While the Clipper (in Multiple Clippers mode) would be this:

Translation was SUCCESSFUL with 0 warning(s) (134 feature(s) output)
FME Session Duration: 23.5 seconds. (CPU: 22.2s user, 0.8s system)
END - ProcessID: 14236, peak process memory usage: 1471896 kB

And in Clippers First mode would be this:

Translation was SUCCESSFUL with 0 warning(s) (134 feature(s) output)
FME Session Duration: 21.4 seconds. (CPU: 20.4s user, 0.6s system)
END - ProcessID: 30123, peak process memory usage: 183344 kB

So the result is the same, but the performance vastly different.

Obviously in the above example the Clipper is the fastest (and be sure to note how the Clippers First mode has reduced memory use by nearly 90%).

But each transformer has different functionality, and if you wanted to output park features with a list of roads or a count of the roads passing through the park, then the LineOnAreaOverlayer would be the transformer of choice, because it has a specific list parameter.
Basically, each transformer works in a different way, has subtle variances in functionality, and will have different performance for any given task. Therefore a translation will benefit in performance if the author is careful in their choice of transformers, and maybe carries out some performance testing first.

### Attributes and Transformation

As mentioned (in Reader Performance) reducing data helps performance because it saves FME from either holding it in memory or caching it to a disk.

However, this isn’t just helped by reducing the number of features; it is also helped by reducing the size of each individual feature.

One aspect of this is attributes. Carrying attributes through a translation impacts performance, so if the attributes are not required in the output, it’s best to remove them *as early as possible* in the translation.

For example, the incoming schema looks like this:

![Incoming Schema](attachment://incoming_schema.png)

But the outgoing schema looks like this:

![Outgoing Schema](attachment://outgoing_schema.png)

Since so many of the source attributes are not required in the output, it makes sense to remove them from the translation, and as early as possible by using a transformer (AttributeManager, AttributeRemover, or AttributeKeeper) directly after the source feature type:
This ensures that none of the extra attributes become a drain on resources by being processed by any further transformers.

**Lists**

One specific type of attribute to beware of is a List. A list in FME is an attribute that can have multiple values. Because of this it can be a big drain on resources.

For example, use a Joiner to join a feature to 1000 records and the list for that feature will have 1,000 sets of records. This is bad enough, but if the list is exploded and all of the original attributes kept, then there will be 1,000 features each with 1,000 sets of attributes!

In general, beware of creating lists unnecessarily and of keeping them in a workspace beyond the point at which they are still of use.

**Geometry and Transformation**

Like attributes, geometry can be removed from a feature, in this case using the GeometryRemover transformer.

Many FME users create translations that handle tabular – non-spatial – data. If you are reading a spatial dataset, then writing it to a tabular format, be sure to remove the geometry early in the workspace, just as you would an attribute.

Another particular problem is carrying around spatial data as attributes. Spatial database formats - for example Oracle or GeoMedia - usually store geometry within a field in the database; for example GEOM. When FME reads the data it converts the GEOM field into FME geometry and drops the field from the data.

However, if you read a geometry table with a non-geometry Reader, the translation could end up with the geometry stored as an FME attribute. A similar thing could happen when a workspace reads only one geometry column of a multiple geometry table.

Geometry will create very large and complex attributes, which take up a great deal of resources. If you don’t need them, then it’s worth removing them.
Basically, you should only carry through the translation any geometry and attributes you need for the output of your workspace. If the data is not required, then it can and should be removed as early in the workspace as possible.

General Optimizations

Here are a few general suggestions that can be used to improve the performance of a workspace. These come straight from the developers at Safe Software.

Jake Speedie says…

It’s said that race driver Michael Schumacher would tilt his head slightly when racing, to allow more air into the engine intake. If, like him, you measure performance down to the millisecond, then these tips are for you!

Avoid Run with Full Inspection

If you aren’t debugging a translation, then avoid using the Run with Full Inspection option. It stashes all data for every connection in the workspace, meaning performance is significantly reduced.

Remove (or Disable) Excess Loggers and Inspectors

Similarly, if you aren’t debugging a translation there’s no need for Logger or Inspector transformers. Remove – or disable – them and your workspace will run more efficiently.

Use Inspectors, not Loggers

If you are intending to inspect a large number of features, then use the Inspector and not the Logger transformer. Logging speeds have improved greatly in the last few versions of FME, but it is still a relatively slow process compared to sending features to the Inspector.

Use the Command Line

Once you have constructed your workspace, run it from the command line instead of from Workbench. It may operate slightly faster.

License Type

It might only be a tiny amount, but an FME with a floating (concurrent) license has to query the license server and so is very marginally slower than fixed licenses.
Exercise:

Exercise 3  Cell Phone Signal Processing

<table>
<thead>
<tr>
<th>Data</th>
<th>City Neighborhoods (Google KML)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cell Phone Signals (CSV)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Overall Goal</th>
<th>Analyze and improve the workspace performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demonstrates</td>
<td>Improving Transformation Performance</td>
</tr>
</tbody>
</table>

| Start Workspace               | C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex3-Begin.fmw |
| End Workspace                 | C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex3-Complete-complete-Advanced.fmw |

You've been working on improving the performance of a friend's workspace. His project is to analyze cell phone signals; to filter out locations that receive a really poor signal, tag them with the neighborhood they belong to – to show which neighborhoods have poor coverage.

So far you've deconstructed the log, uncovered areas of concern, and cleaned up the Readers and Writers as much as possible.

Now let's lookin into using our new knowledge of transformation performance to try and speed up the workspace a bit more.

1) Start Workbench
Start FME Workbench and open the workspace Performance-Ex3-Begin.fmw (or the workspace from the previous exercise, if you still have it open).

2) Check for Extra Transformers
The first aspect of the workspace to check is for any extra transformers that aren’t needed and that will be slowing performance. The most obvious is the Logger transformer:

This was presumably used for debugging the original workspace but is now doing nothing for us. So, delete the Logger transformer attached to the CSV Reader.

3) Remove Attributes
Another quick fix we can do is to remove any attributes we don’t need, right at the start of the workspace. Check the schemas of the Reader and Writer feature types:
The Readers contain quite a lot of attributes, on both datasets. The Writers contain very few attributes, and the GoodLocations feature type has none at all. This suggests we can remove some attributes that are not going to be needed in the output.

Put an AttributeManager transformer after the Neighborhood feature type, but before the Clipper, and set it up to keep only the NeighborhoodName attribute:

Jake Speedie says…

"You might be thinking, "is it really worth removing attributes from only six neighborhood features?" The answer is a resounding "YES" - because those attributes are being copied on to 1.6 million CSV features."

Now we can place a second AttributeManager to remove excess attributes from the CSV (signal) data. I know we're supposed to ignore that Cloner transformer, but the AttributeManager should definitely be before it. That way you're not cloning extra attributes for no reason.

So place an AttributeManager before the Cloner and use it to remove all attributes except StationID, Power, and Quality:
Reducing attributes like that should definitely reduce the amount of memory being used. On my computer it goes from this:

```
INFORM|FME Session Duration: 4 minutes 1.9 seconds. (CPU: 219.7s user, 19.4s system)
INFORM|END - ProcessID: 46312, peak process memory usage: 1776304 kB
```

...to this:

```
INFORM|FME Session Duration: 3 minutes 36.2 seconds. (CPU: 194.4s user, 19.9s system)
INFORM|END - ProcessID: 53072, peak process memory usage: 1349336 kB
```

A nice reduction in time and memory use just from removing some excess attributes and an unwanted Logger transformer.

4) Check Group-Based Processes
The Clipper is a group-based transformer; it has to be since it is processing both the neighborhoods and the cell signal data. If you run the translation now - as in the screenshot below - you would see that the features build up in front of the transformer, but none are released until the processing is fully complete:

```
So we should check if there's a way to turn the transformer into one that operates on a feature basis.
Open the Clipper parameters. Notice that there is a parameter for Clipper Type. Change this to Clippers First:
```
This will make this virtually a feature-based transformer. Each clippee will not need to be cached because the full set of clippers is already known.

However, we have to make sure the Clippers really do arrive first, and this we can do by making the Clippers the first Reader in the Navigator window.

So, click the VancouverNeighborhoods Reader in the Navigator window and drag it above the CSV Reader:

Run the workspace now and features will emerge from the transformer as they arrive, like so:

NB: I assume the numbers in and out of the transformer are different because FME doesn’t update the counts for every single feature (that would just slow down the translation).

5) Run Workspace

Now run the workspace to see what we have so far.

Remember, after Reader improvements we had this result:

INFORM|FME Session Duration: 4 minutes 1.9 seconds. (CPU: 219.7s user, 19.4s system)
INFORM|END - ProcessID: 46312, peak process memory usage: 1776304 kB

After removing attributes we had this:

INFORM|FME Session Duration: 3 minutes 36.2 seconds. (CPU: 194.4s user, 19.9s system)
And after setting the Clippers First parameter we now have this:

```
INFORM|FME Session Duration: 3 minutes 41.1 seconds. (CPU: 200.7s user, 18.7s system)
INFORM|END - ProcessID: 52820, peak process memory usage: 96220 kB
```

That's a stunning improvement in memory use, just from "unblocking" a single transformer. Can we do even better?

### 6) Rearrange Transformers

Another important part of performance is to assess the order of transformers (as we did with the AttributeManager/Cloner) to ensure no excess work is being performed.

Looking at the workspace, the Neighborhood attribute is only required by the bad (low power) features. It isn’t needed by the good locations.

But, we’re still attaching the information onto all of the features, good or bad.

Can we prevent that? Yes! We can move the Tester transformer to before the Clipper. Then we aren't attaching useless information to features that don't need it.

So, select the Tester transformer and press Ctrl+X to cut it from the workspace. Notice that the connections are healed automatically, though they aren’t quite right.

Now press Ctrl+V to paste the Tester back into the workspace, but unconnected. Now drag it into the CSV data stream, between the Cloner and the Clipper:

Finally, let’s fix the feature mapping.

Move the connection from Clipper:Inside > GoodLocations to Tester:Failed > GoodLocations:
Re-run the workspace. The result will be something like this:

INFORM|FME Session Duration: 1 minute 49.0 seconds. (CPU: 89.6s user, 18.7s system)
INFORM|END - ProcessID: 53180, peak process memory usage: 95240 kB

Hurrah! Compared to the original log we’re over five times faster with 95% less memory use! Your friend should be very happy with what we’ve managed to achieve with his workspace.

Advanced Exercise

There is, perhaps, one other way we can upgrade this workspace’s performance. It’s radical and unintuitive, but it might just work! If you have time to try this, then it is an interesting solution.

Do you remember the previous tip about using the PointCloud XYZ reader for CSV data? Follow these instructions to see what effect that technique has.

7) Save Workspace

Save a copy of the workspace completed in the steps above.

8) Delete Reader/Transformers

Firstly, delete the CSV Reader (and its feature type), the AttributeManager for the CSV data, and the Tester transformer too. For a fair comparison, keep the Cloner.

9) Add Reader

Now select Reader > Add New Reader and in the Add Reader dialog enter the following values:

<table>
<thead>
<tr>
<th>Reader Format</th>
<th>Point Cloud XYZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reader Dataset</td>
<td>C:\FMEData2016\Data\CellSignals\CellSignal.csv</td>
</tr>
</tbody>
</table>

The Reader parameters are very important here, so click the Parameters button.

Firstly make sure the Separator Character is set to a comma (not “space”). Also check the option for “File Has Field Names.”
Then in the Point Cloud Component Map, set the longitude and latitude attributes to X/Y components:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Component</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>longitude</td>
<td>x</td>
<td>Refl64</td>
</tr>
<tr>
<td>latitude</td>
<td>y</td>
<td>Refl64</td>
</tr>
<tr>
<td>StationID</td>
<td>StationID</td>
<td>UInt16</td>
</tr>
<tr>
<td>Power</td>
<td>Power</td>
<td>Refl64</td>
</tr>
<tr>
<td>Quality</td>
<td>Quality</td>
<td>Refl32</td>
</tr>
<tr>
<td>num_measures</td>
<td>num_measures</td>
<td>UInt16</td>
</tr>
<tr>
<td>Code</td>
<td>Code</td>
<td>String</td>
</tr>
</tbody>
</table>

It’s important to get this exactly right, as you will have to re-add the Reader to fix any problems.

Click OK to close the dialog and OK again to add the Reader. Connect the new feature type to the Cloner Input port.

10) Add PointCloudFilter
A point cloud feature is not the same as a number of point features, so we will have to convert the data at some stage. However, let’s see if we can make use of some point cloud functionality first.

Add a PointCloudFilter transformer to the workspace after the Cloner. This is the point cloud equivalent to a Tester and may be quicker than testing each point individually:

Open the parameters dialog for this transformer.

Under Expression select the option to open an arithmetic editor. In the editor, look under the list of Point Cloud components to the left and double-click on at the bottom of that list. Enter Power as the component name.

Now click on the end of the expression, enter a less than operator (<) and type the value -125 (make sure you put -125 and not +125). The dialog should look like so:

Click OK to close that dialog. Enter a new port name of Bad Signal. Click OK.

The workspace now looks like this, with a new output port:
11) Add PointCloudCoercers

Add two PointCloudCoercer transformers to the workspace, one attached to each output from the PointCloudFilter. One output should be directed to the Clipper:Clippee port, the other to the GoodLocations output feature type:

The final step is to set the coercer parameters and expose some attributes. Open the parameters dialogs for each of the PointCloudCoercers in turn. In both cases, set Output Geometry to Individual Points.

Again, in both cases, set Preserve Point Components As to Attributes.

For the data, which doesn’t need attributes, leave the Point Components to Preserve section empty. For the Bad Signal data, enter the following component names to extract them as attributes:

- StationID
- Power
- Quality

12) Save and Run Workspace

Now save and run the workspace. This time the log will report the performance as:
INFORM|FME Session Duration: 1 minute 18.7 seconds. (CPU: 61.9s user, 16.0s system)
INFORM|END - ProcessID: 46136, peak process memory usage: 124624 kB

Well done! The memory use is up slightly, but the workspace is running faster than ever.

CONGRATULATIONS

By completing this exercise you have learned how to:

- Assess transformation performance
- Improve transformation performance by removing excess transformers
- Improve transformation performance by removing excess attributes
- Improve transformation performance by using group-based parameters
- Improve transformation performance by rearranging transformers
- Use the PointCloud XYZ Reader to handle CSV data
Database Optimization

Besides the previous performance tips and tricks, there are some that apply only to databases and some that apply only to specific formats of database. Most are also either for reading or writing (not both).
Database Reading

Reading and filtering data (querying) from a database is nearly always faster when you can use the native functionality of the database.

For Readers this means using the various parameters that occur in the Navigator window of Workbench, like this SQL Server Reader:

Here there is a WHERE clause and a search envelope. When you set these then FME’s query to the database includes these parameters. This is much faster than reading the full database table and filtering it by attribute (Tester) or geometry (SpatialFilter).

Additionally, Reader feature types often contain a set of parameters containing WHERE clauses and SELECT statements, which mean you can set a query to apply to just one particular table in your workspace:

Database Transformers

Besides Readers, transformers can also be used to query database data. The best to use is the SQLExecutor (or SQLCreator) as these pass their queries to the database using native SQL. If you don’t want to write SQL then you can use the FeatureReader transformer; but be aware this transformer is more generic and won’t give quite the same performance.

The SQLExecutor is particularly worth being aware of, as it issues a query for each incoming feature. This can be useful where you need to make multiple queries.

For example, here a query is issued to the database for every library feature that enters the SQLExecutor:
Generally the output from the SQLExecutor is an entirely new feature. If you want to simply retrieve attributes to attach to the incoming feature, then the Joiner transformer is more appropriate.

---

**Jake Speedie says…**

*FME is fast, but if you can filter or process data in its native environment, it’s likely to be faster still. For example, a materialized view (a database object containing the results of a query) is going to perform better than reading the data and filtering it in FME. Similarly, a SQL Join is going to perform better than reading two tables into FME and using the FeatureMerger transformer. Sometimes it really is a case of working smart, not hard!*

---

**Queries and Indexing**

Of course, all queries will run faster if carried out on indexed fields – whether these are spatial or plain attribute indexes – and where the queries are well-formed.

To assess how good performance is, remember the log interpretation method of checking timings:

For example, take this section of log timings:

```
2016-07-10 14:43:06 | 8.5 | 0.0 |
2016-07-10 14:43:13 | 8.8 | 0.3 |
2016-07-10 14:46:29 | 18.0 | 9.1 |
2016-07-10 14:49:29 | 25.8 | 7.9 |
```

The workspace took over six minutes to complete this part, but FME is only reporting 25.8 seconds of processing! If the query is to a database then the conclusion is that the fields are either not indexed or the query is badly formed.

To confirm this you could open a SQL tool – for example the SQL Server Management Studio – and run the query there. If it takes as long to run there as in FME, then you know for sure FME is not the bottleneck in your performance.

---

**Jake Speedie says…**

*Structure your SQL commands so that indexed attributes are first, followed by the most limiting of the other matches. For example, if one portion of the clause matches 10 rows, and another matches 1000 rows, put the one matching only 10 first.*

Besides indexes, there are other in-built functions that can cause databases to return data slower than expected.
For example, when reading from a Geodatabase geometric network, all of the connectivity information must be verified. Therefore, reading is faster if the network information is ignored.

This can be achieved using the Ignore Network Info parameter for the Geodatabase reader, with a similar parameter called Ignore Relationship Info to ignore Relationships:

Similarly, Excel formulas can be expensive to read, so turning them off when the schema is generated can speed up reading the data:

Look for parameters like these on many FME formats.
Database Writing

Whereas the performance of reading from a database is largely dependent on the database setup itself, when writing to a database there are very many FME parameters that can help to fine tune the workspace performance.

Remember that writing to a database incurs network overheads. There has to be a balance between the amount of data being sent (the network traffic), database performance, and the risk of losing uncommitted data.

Each database Writer has a set of parameters for handling the number of features to write for any particular transaction. Not every format supports these, but the two most common parameters are Features per Transaction and Features per Bulk Write.

Features per Bulk Write

The Features per Bulk Write parameter tells FME how many features to send at a time to the database. Features sent to a database Writer will get cached in memory by FME until this number of features is reached, and only then will they be sent to the database. This is also known as chunk size.

This parameter is a way to balance server performance and network traffic with FME performance. The higher the number the more features FME caches, but the fewer requests it needs to make to the database (and therefore less network traffic).

A lower number means FME caches less data, but there are more requests made to the database.

Features per Bulk Write also needs to be considered against the value of Features per Transaction.

Features Per Transaction

Features per Transaction (also sometimes called Transaction Interval) controls how many features are entered into a database before a commit command is issued.

Each commit adds delay to the writing process, so setting this parameter must balance the speed of the translation (a higher number) against the risk that a translation may fail and features need to be rolled-back (a lower number).

For example, if this parameter is set to a value of 1, then each and every feature is committed individually. If the process fails then only the last feature will be lost from the database, but the costs is much reduced performance.

Alternatively, if the parameter is set to a very high value (more than the number of features being written) then only one commit takes place and performance improves. However, if the translation fails, then all features previously written will be rolled-back and lost to the database.

If Features per Transaction is less or equal to Features per Bulk Write, then FME basically caches a number of features and sends them to the database where they are immediately committed.

If Features per Transaction is greater than Features per Bulk Write, then FME is sending features to the database where they will be cached until the transaction commit total is reached.

Jake Speedie says…

The Transaction and Chunk parameters can differ from format to format. For example, SQL Server has a single Bulk Option flag rather than a numeric setting. Therefore it’s very important that you check out the FME Readers and Writers Manual to confirm what parameters are available for your database, and how exactly they operate.
Writing and Indexing

Whereas indexes can improve performance for reading data, for writing they can cause a great reduction in the speed of translation.

That’s because the index will often get re-built with every feature (or every transaction) that is committed to the database.

To remedy this it’s suggested that indexes are dropped (deleted) before carrying out bulk inserts of data into a database table.

A Writer feature type also has options to truncate or drop tables when writing to them:

For the above reason, dropping a table is more efficient than truncating it because the drop action also removes the indexes.

For similar reasons, you may want to turn off networking connectivity when writing data to a Geodatabase geometric network.

Jake Speedie says...

If you think these sort of things don’t make much of a difference, consider these two (real-life) log files:

2016-04-05 06:19:04|1069.3| 0.2|INFORM|Transaction #1488 was successfully committed
2016-04-05 06:19:04|1069.7| 0.5|INFORM|Transaction #1489 was successfully committed

2016-04-07 15:19:01|1536.1| 0.5|INFORM|Transaction #1488 was successfully committed
2016-04-07 15:19:09|1536.7| 0.5|INFORM|Transaction #1489 was successfully committed

A user had a workspace to write to a database. He ran it on two different machines and complained that, whereas Machine A was fast, Machine B was way slower.

As you can see from the first log, Machine A committed transaction 1489 in 0.5 seconds. Machine B also committed transaction 1489 in 0.5 seconds. However, the elapsed time (15:19:01 to 15:19:09) was 8 seconds. In short, a network or database issue with Machine B was causing a non-FME slowdown in each database commit.

8 seconds might not sound a lot, but with 1,500 transactions that’s an accumulated delay of over 3 hours!
<table>
<thead>
<tr>
<th>Exercise 4</th>
<th>Garbage Collection Day Project</th>
</tr>
</thead>
</table>
| **Data**  | Addresses (PostGIS or Esri Geodatabase)  
Garbage Zones (Esri Geodatabase) |
| **Overall Goal** | Analyze and improve the workspace performance |
| **Demonstrates** | Database Optimization |
| **Start Workspace** | C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex4-PostGIS-Begin.fmw  
C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex4-Geodatabase-Begin.fmw |
| **End Workspace** | C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex4-PostGIS-Complete.fmw  
C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex4-Geodatabase-Complete.fmw |

Members of the public often call the city to ask what day their garbage collection is. To help the city has an internal system hosted on FME Server. Members of the planning department can lookup an address ID, enter it into a published parameter, and the system retrieves the garbage pickup information.

The system works, but is perhaps slower than it should be. Let’s run this short exercise to discover why.

---

**Miss Vector says...**

*This exercise uses either a PostGIS database hosted on Amazon RDS or an ESRI Geodatabase. Be sure to open the correct workspace and follow the correct instructions for your format.*

---

1) **Create Database Connection (PostGIS Only)**

To use a PostGIS database as source requires a connection to it.

In a web browser visit [http://fme.ly/database](http://fme.ly/database) - this will show the parameters for a PostGIS database running on Amazon RDS.

Start Workbench and select Tools > FME Options from the menu bar.

Click on the icon for the Database Connections category, then click the [+] button to create a new connection. In the "Add Database Connection" dialog, first select PostgreSQL as the database type. Then enter the connection parameters obtained through the web browser.

Give the connection a name (if you call it *FME Training Database Connection* it will match the starting workspace) and click Save.
NB: Yes, the password is fmedata as well!

Then click OK to close the FME Options dialog.

2) Open and Run Workspace

Open the workspace C:\FMEData2016\Workspaces\DesktopAdvanced\Exercise2d-PostGIS-Begin.fmw or Exercise2d-Geodatabase-Begin.fmw

The workspace looks something like this:

![Diagram of workspace](image)

Basically a published parameter accepts an address ID. The postal address database is read and filtered against this ID. The chosen address is used in a spatial overlay against garbage zones. The result is formatted in HTML and written out with a Text File Writer.

**.1 UPDATE**

FME2016.1 now includes a dedicated HTML format writer, which could be used in place of the Text File writer in this exercise.

To get a comparison, run the workspace. Use Prompt Mode to be prompted for an address ID. A suitable address ID to use is 127209 (PostGIS) or 6135 (Geodatabase).

The result, in a web browser, is this:
The performance will read like this:

**PostGIS**

INFORM|FME Session Duration: 9.2 seconds. (CPU: 3.6s user, 0.5s system)
INFORM|END - ProcessID: 11576, peak process memory usage: 130092 kB

**Geodatabase**

INFORM|FME Session Duration: 2.2 seconds. (CPU: 1.9s user, 0.2s system)
INFORM|END - ProcessID: 9376, peak process memory usage: 86648 kB

The Geodatabase is quicker because it is being read from your own file system, not a remote database.

3) **Set Up WHERE Clause**

Neither PostGIS or Geodatabase have a WHERE clause for the Reader itself, but their feature types do. So open the properties dialog for the Postal Address Reader feature type and click the Format Parameters tab.

In the WHERE Clause parameter enter:

**PostGIS**

"AddressId" = $(AddressID)

**Geodatabase**

OBJECTID = $(AddressID)
For PostGIS, be sure to notice the lower-case "d" in the "Id" part of the field name! Also note the difference in use of quotes between the two formats.

4) Delete Tester

Now we have the WHERE clause, the Tester transformer is no longer required, so delete it.

5) Re-Run Workspace

Re-run the workspace. This time only 1 feature is read from the database. The performance improves accordingly:

**PostGIS**

```
INFORM|FME Session Duration: 5.7 seconds. (CPU: 2.4s user, 0.4s system)
INFORM|END - ProcessID: 3208, peak process memory usage: 128836 kB
```

**Geodatabase**

```
INFORM|FME Session Duration: 0.6 seconds. (CPU: 0.5s user, 0.1s system)
INFORM|END - ProcessID: 10728, peak process memory usage: 86248 kB
```

Memory usage hasn’t improved, but the translation ran faster.

**CONGRATULATIONS**

*By completing this exercise you have learned how to:*

- Create a database connection
- Use a SQL WHERE clause to avoid reading all data
Parallel Processing

Parallel Processing is a way to improve performance on high-end machines.

What is Parallel Processing?

Each FME translation is usually a single process on your computer. Parallel processing is when you transform your data as several simultaneous processes. The fact that they run simultaneously means the whole translation will run several times quicker than it used to.

Parallel processing allows FME to make use of multiple cores on a computer. There are four levels of parallel processing in FME, and each maps to the number of cores in this way:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>No parallelism</td>
<td>1 Process</td>
</tr>
<tr>
<td>Minimal</td>
<td>Cores / 2</td>
</tr>
<tr>
<td>Moderate</td>
<td>Cores</td>
</tr>
<tr>
<td>Aggressive</td>
<td>Cores x 1.5</td>
</tr>
<tr>
<td>Extreme</td>
<td>Cores x 2</td>
</tr>
</tbody>
</table>

So, for example, on a quad-core machine, minimal parallelism will result in two simultaneous FME processes. Extreme parallelism would result in eight.

There is also a hard cap for each license level:

<table>
<thead>
<tr>
<th>FME Edition</th>
<th>Process Cap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Edition</td>
<td>4 processes</td>
</tr>
<tr>
<td>Professional Edition</td>
<td>8 processes</td>
</tr>
<tr>
<td>All Other Editions</td>
<td>16 processes</td>
</tr>
</tbody>
</table>

So, if you have a Base Edition license you are never going to get more than four processes at one time, regardless of machine type and the parallelism parameter.

Jake Speedie says…

Parallel Processing is very effective when you are offloading a task elsewhere – for example calling a Server with the HTTPFetcher – as each process is a tiny impact on the FME system resources. However, be aware, each parallel process involves starting and stopping an FME engine, and this takes time. So, don’t parallelize your processes when the task already takes less than the time to stop/start FME!

Transformers and Parallel Processing
There are a number of basic FME transformers that have built in options for parallel processing. Parallel processes work on groups of features, so the transformer must be group-based and have a group-by parameter in order for the user to be able to define the parallel processing groups.

For example, this Bufferer transformer is set up to buffer a set of street features:

![Bufferer Parameters](image)

Each street (i.e. each segment with the same street name) will be processed as a separate group. To speed up the translation, each group is being handled as a separate process (sadly the user cannot confirm the source data is already ordered by group, which would improve performance even more).

When a translation is run in parallel mode, then a number of “worker” processes appear in your process manager:

<table>
<thead>
<tr>
<th>Image Name</th>
<th>PID</th>
<th>User Name</th>
<th>CPU</th>
<th>Memory (K)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmeworkbench.exe *32</td>
<td>31372</td>
<td>inmark</td>
<td>03</td>
<td>366,984 K</td>
<td>FME Workbench</td>
</tr>
<tr>
<td>fmeworker.exe *32</td>
<td>45012</td>
<td>inmark</td>
<td>11</td>
<td>27,828 K</td>
<td>FME EXE</td>
</tr>
<tr>
<td>fmeworker.exe *32</td>
<td>49508</td>
<td>inmark</td>
<td>23</td>
<td>29,444 K</td>
<td>FME EXE</td>
</tr>
<tr>
<td>fmeworker.exe *32</td>
<td>49840</td>
<td>inmark</td>
<td>18</td>
<td>26,408 K</td>
<td>FME EXE</td>
</tr>
<tr>
<td>fmeworker.exe *32</td>
<td>49936</td>
<td>inmark</td>
<td>23</td>
<td>25,712 K</td>
<td>FME EXE</td>
</tr>
<tr>
<td>fmeworker.exe *32</td>
<td>50004</td>
<td>inmark</td>
<td>04</td>
<td>12,044 K</td>
<td></td>
</tr>
<tr>
<td>fmeworker.exe *32</td>
<td>50164</td>
<td>inmark</td>
<td>01</td>
<td>9,824 K</td>
<td></td>
</tr>
</tbody>
</table>

**Parallel Processing Groups**

Best performance gains are when you have a small number of groups with a large amount of data. When there are many groups with only a few features then any performance gain will not be great and, in fact, the whole process might even be slower. Disk access can be a big bottleneck there.

Because each group gets processed independently, there can be no relationship between features in different groups. If features are related, and their results dependent on each other, then they must be in the same group.

However, if all data is unrelated and the contents of the group are unimportant, then it's possible to make artificial groups using a ModuloCounter or RandomNumberGenerator transformer.

For example, here the user has millions of lines to buffer (separately) and uses a ModuloCounter to assign them to one of four groups for parallel processing. Note the Group By parameter in the Bufferer is set to the _modulo_count attribute:
1.1 UPDATE

The Bufferer transformer in the above screenshot gained a <Rejected> port in FME2016.1
Exercise 5  
Performance Review Project

<table>
<thead>
<tr>
<th>Data</th>
<th>Various</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>Analyze and improve the workspace performance</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>Parallel Processing</td>
</tr>
</tbody>
</table>

**Start Workspace**

C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex5a-Begin.fmw  
C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex5b-Begin.fmw  
C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex5c-Begin.fmw

**End Workspace**

C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex5a-Complete.fmw  
C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex5b-Complete.fmw  
C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex5c-Complete.fmw

Included here are a number of workspaces generated by your colleagues. As the resident FME expert you have been asked to assess the performance of each workspace.

**1) Workspace A**

Start Workbench and open workspace C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex5a-Begin.fmw

This workspace carries out spatial overlays and nearest neighbor finding. It determines which neighborhood each postal address and fire hall is located in, and finds the nearest firehall to each address. The nearest firehall must be in the same neighborhood.

Examine the workspace and assess it for its ability to employ parallel processing. You should answer the following questions:

- Do any transformers permit parallel processing?
- Is there an existing group I can parallel process by?
- Is there an artificial group I can create to parallel process by?
- Will parallel processing speed up the workspace performance, or make it worse?

The answers to these questions can be found in the finished workspace:  
C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex5a-Complete.fmw

**2) Workspace B**

Start Workbench and open workspace C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex5b-Begin.fmw
This workspace reads contour data from Esri Shapefile datasets and converts them into raster DEM tiles - one tile for each shapefile.

Again, examine the workspace and assess it for its ability to employ parallel processing. You should answer the following questions:

- Do any transformers permit parallel processing?
- Is there an existing group I can parallel process by?
- Is there an artificial group I can create to parallel process by?
- Will parallel processing speed up the workspace performance, or make it worse?

The answers to these questions can be found in the finished workspace:
C:/FMEData2016/Workspaces/DesktopAdvanced/Performance-Ex5b-Complete.fmw

3) Workspace C
Start Workbench and open workspace C:/FMEData2016/Workspaces/DesktopAdvanced/Performance-Ex5c-Begin.fmw

This workspace may be familiar to you. It comes from the Desktop Basic course and is a project to find all single-family addresses within 50 metres of a major road.

.1 UPDATE

The Bufferer transformer in the above screenshot gained a <Rejected> port in FME2016.1

Once more, examine the workspace and assess it for its ability to employ parallel processing. You should again answer the following questions:
Do any transformers permit parallel processing?
Is there an existing group I can parallel process by?
Is there an artificial group I can create to parallel process by?
Will parallel processing speed up the workspace performance, or make it worse?

The answers to these questions can be found in the finished workspace:
C:\FMEData2016\Workspaces\DesktopAdvanced\Performance-Ex5c-Complete.fmw

CONGRATULATIONS

By completing this exercise you have learned how to:
- Assess when to use parallel processing
- Set up transformers to use parallel processing
- Create an artificial group for parallel processing
Performance, FME Server, and FME Cloud

FME Server adds an extra dimension to FME performance, that of scalability. In terms of performance the item most easily scalable is the number of FME engines.

Increasing the number of engines supports a higher volume of jobs and the FME Server Core contains a Software Load Balancer (SLB) to distribute jobs to the FME engines in a balanced way.

In this scenario, FME Server is used not for its web-based abilities, but rather for its potential in processing large amounts of data in relatively less time.

Using Server for Bulk Translations

By default, utilizing multiple engines is only possible when you have multiple workspaces that can be run. When you have only a single workspace, and wish to process it more efficiently on FME Server, then you need to divide that workspace into multiple jobs.

To do so I can create a master workspace that divides my source data into separate parts and sends each to a different job using the FMEServerJobSubmitter transformer.

For example, I can calculate the bounds of tiles to be created and share the load over multiple server engines by running the workspace once for each tile.

FME Cloud

FME Cloud is an installation of FME Server hosted by Safe Software on Amazon Web Services technology and used on a pay-as-you-go basis. The benefit is that you don’t have to purchase FME Server, simply make use of it whenever you have a job that can take advantage of its power.

The key to automating this for performance benefits are the FME Cloud custom transformers available on the FME Store:

With the FMECloudInstanceLauncher transformer I can run my master workspace (as in the example above) and have it automatically start an FME Cloud instance and run the job on it.

This way I can start a new instance for each job, or run several jobs on one instance, depending on the type of instance and how many engines it has running on it.
Module Review

This chapter looked at FME Performance and some of the techniques available to improve it

What You Should Have Learned from this Module

The following are key points to be learned from this session:

Theory

- Performance is the measure of useful work done in a given time. Excess data and caching of data to disk are two factors that can impact performance greatly
- 64-bit FME can make use of more system memory, but does not have the same format reach that 32-bit FME does
- Analyzing a log file helps to determine where performance improvements can be made
- Improve reading performance by reducing the amount of data being read
- Improve writing performance by ordering the FME Writers correctly
- Improve transformation performance by removing excess attributes and properly managing group-based transformers
- Make use of all Reader/Writer parameters to improve database performance
- Employ parallel processing to make FME multi-threaded
- Upload larger tasks to multiple engines on FME Server

FME Skills

- The ability to analyze and deconstruct an FME log file
- An understanding of potential methods for improving Reader, Writer, and transformer performance
- The ability to use database parameters to improve performance
- The ability to apply parallel processing in an FME workspace

Further Reading

For further reading why not browse articles tagged with Performance on our blog?
Questions

Here are the answers to the questions in this chapter.

Miss Vector says...

Let’s see if you can figure out the one false statement from these facts about 64-bit FME:

1. 32-bit Windows can use only 32-bit FME. 64-bit Windows can use either 32-bit or 64-bit FME
2. You can install both 32-bit and 64-bit FME on a 64-bit computer, and use either one as necessary
3. You can install 32-bit and 64-bit engines on the same FME Server core
4. A workspace authored on 32-bit FME cannot be run on a 64-bit engine

#1 and #2: True. 64-bit Windows can have both 32-bit and 64-bit versions installed, but only the 64-bit version will take advantage of any extra memory available to it.
#3: True. You can install engines of both types. Job Routing techniques will allow you to designate which jobs should be processed by which engine.
#4: False. The authoring platform has nothing to do with the platform a workspace can be run on.

There are many helpful articles on 64-bit on the FME knowledge center.

Miss Vector says...

How would you find the Emptying Factory Pipeline setting in the log window?

1. Looking for the color: it is the only message highlighted in red
2. Looking for the message type: it is the first message of type STATS
3. By using the log search tools
4. By opening Tools > FME Options to turn off all other messages

Miss Vector says...

Given this screenshot, which should we make the first Writer in this workspace?
1. A
2. B
3. C
4. Don’t know!

Sorry, trick question, but it is impossible to tell from just this screenshot. That’s because it’s not just the most features that is important, but the size of those features. There are more features in Writer C, but they are just point features. Writer B is writing features with no geometry, but we don’t know how many attributes there are and of what type. And Writer A is writing unknown geometry. If they were park boundaries consisting of many thousands of vertices, that may be more data in total than the others. In short, you must use your own judgement and knowledge of the data being processed to make this sort of decision.

Miss Vector says...

Which of these transformers have group-related parameters for improving performance (pick all that apply and see if you can get the answers without looking at the transformers):

1. StatisticsCalculator
2. SpikeRemover
3. PointCloudCombiner
4. FeatureMerger
Custom Transformers

Custom Transformers are very powerful tools at either a basic or an advanced level.

What is a Custom Transformer?

A custom transformer is a sequence of standard transformers condensed into a single transformer. Any existing sequence of transformers can be turned into a custom transformer.

![Diagram of Custom Transformer]

Custom Transformer Purposes

Among other functions, custom transformers help to:

- Tidy Workspaces
  - By condensing chunks of content the workspace canvas becomes less cluttered
- Reuse Content
  - A sequence of transformers encapsulated in a single object can be reused throughout a workspace and shared with colleagues.
- Employ Advanced Functionality
  - Using a Custom Transformer enables additional functionality to be used, such as looping and parallel processing

First Officer Transformer says…

Welcome aboard this Safe Software training chapter on Custom Transformers. I’ll be your guide to all of the functionality involved. As you can see, Custom Transformers are excellent tools for carrying out Best Practices in FME, both speeding up your projects and reducing turbulence in the Workbench canvas.

Creating a Custom Transformer

A Custom Transformer can be created from scratch – i.e. you start with an empty custom transformer and add content into it – or can be created from an existing sequence of transformers.

Custom transformers are created by either selecting Create Custom Transformer from the canvas context (right-click) menu, or by selecting Transformers > Create Custom Transformer from the menubar. The shortcut key for this function is Ctrl+T.
If a number of existing transformers are selected when you issue the Create Custom Transformer command, then they are automatically added to the new custom transformer; otherwise the new custom transformer is created empty except for an input and output port.

Here a user is creating a new custom transformer based on a series of existing ones:

The new custom transformer will be pre-populated with these four transformers.

---

.1 UPDATE

Although not related to custom transformers, the context menu in the above screenshot gained options for Hide Connections and Connect Junction in FME2016.1. See this blog article for more information about these new functions.
Naming a Custom Transformer

All Custom Transformers require a name and (optionally) a category and description. A dialog in which to define these automatically appears when you create a new custom transformer.

The category can be set to match any existing category of FME transformers, or a custom category of your own.

Notice also the “Use Extended Description” parameter. This allows you to enter extra information about the custom transformer, such as requirements for use, development history, and legal terms and conditions; in fields that support the use of rich text.

These fields are particularly important when you intend to share the custom transformer with work colleagues or clients.

The New Custom Transformer

A newly created custom transformer then looks like this:

Notice that it appears under a new tab on the Workbench canvas and consists of the original transformers with additional input and output objects.

When you click on the Main tab, to return to the main canvas view, the three original transformers have now been replaced by a custom transformer object that is automatically connected into the existing workspace:
This custom transformer looks and behaves in the same way as any standard FME transformer; with input and output ports (that match the input/output objects in the custom transformer tab), plus a parameters dialog.

### Editing a Custom Transformer

To edit the contents of a custom transformer, simply click on the tab for that transformer. This opens the transformer definition and you may edit the content in the same way that you would in the main canvas.

In the Navigator window, where a workspace would have a section labelled Workspace Properties, a custom transformer has Transformer Properties.

This is where the information – name, category, description, etc. – that was entered earlier can be edited.

---

**First Officer Transformer says…**

*If a custom transformer has been created from scratch, without any original transformers selected, it would start out empty and look like this:*

Then you can start building or editing the transformer from scratch. There is not a lot of difference between creating content in the main canvas and turning it into a custom transformer, and creating an
empty custom transformer and creating the content in there.
Creating Custom Transformers

Using Custom Transformers

Once a custom transformer has been created, it is connected into the workspace and - apart from a different color - looks just like a normal FME transformer would.

However, the resemblance to a normal transformer is not just in appearance. In the same way that multiple instances of a transformer can be used in a workspace, a custom transformer can be used any number of times too. This makes Custom Transformers not just a way to tidy a workspace, but also as a way of re-using content.

To place extra copies of a custom transformer - again like a regular transformer - you use the Transformer Gallery (look under a section labelled "Embedded Transformers):

...or you can use Quick Add:

As with FME transformers, each newly placed instance of a custom transformer will be renamed (or numbered) as necessary, in order to avoid a clash of names.

TIP

The transformer name, category, and description all appear in the Quick Add tool - as well as appearing in the help window - so it is definitely worthwhile setting these parameters.

Editing a Re-used Custom Transformer

Multiple instances of a custom transformer all use the same core definition; i.e. there may be multiple copies on the Main tab, but only a single tab for the custom transformer definition.

A key benefit of this approach is that every instance can be updated or edited simply by editing the custom transformer definition.

For example, if a parameter is changed for the Aggregator inside this custom transformer:
...then the parameter automatically changes for ALL instances of the transformer that have been placed.

This makes the editing of a sequence of transformers much easier, because the edit only needs to be made once, no matter how many times that sequence has been used.
A colleague - new to FME - has created a workspace that calculates the population density for neighborhoods in the city of Vancouver, and comments that this technique could be reused for other projects.

You mention custom transformers as a way of doing this and will now demonstrate to her how to turn this workspace into a general solution that calculates the average density of items in a known space.

1) Start Workbench

Start by opening your colleague's workspace: C:\FMEData2016\Workspaces\DesktopAdvanced\CustomTransformers-Ex1-Begin.fmw

You may wish to run the workspace and examine the output to see what it does and how it works. Basically it calculates the population density (persons per square kilometre) for the years 2001 and 2011:

![Table View](image)

.1 UPDATE

The AreaCalculator transformer in this workspace gained a <Rejected> port in FME2016.1

2) Create Custom Transformer

The key components for the custom transformer are the AreaCalculator and ExpressionEvaluator transformers. If you examine the workspace you'll see two ExpressionEvaluators (one for the year 2001, one for 2011) but we don't need to include both in the custom transformer.

So select the AreaCalculator transformer and the first ExpressionEvaluator, right-click on them, and choose the context menu option 'Create Custom Transformer.'

In the Create Custom Transformer dialog enter a name, category, and description for the new custom transformer. A good name for the transformer will be the DensityEvaluator.
First Officer Transformer says…

You can’t call it the DensityCalculator; FME already has one of those!

Be sure the Attribute References parameter is set to “Handle with Published Parameters” (more on that later) and click OK:

The custom transformer will now be created.

3) Inspect Custom Transformer

Flip back and forth between the DensityEvaluator tab and the Main tab to see how the custom transformer is constructed, and how it is placed in the workspace itself.

Back in the Main tab, click on the cog wheel icon on the custom transformer to open its parameters dialog.

The main parameter is one created automatically by FME to accept the attribute to be processed. You'll see it is automatically preset to the TotalPopulation2001 attribute.
4) Run Workspace
Save the workspace and then run the workspace, to ensure the output has not changed. However, note that this is just the start of this custom transformer, and we should tidy it up (make it more generic) before trying to reuse it in other scenarios.

Miss Vector says…
Why do you think that we left the CSMapReprojector transformer out of our custom transformer? Any ideas?

CONGRATULATIONS
By completing this exercise you have learned how to:
• Create a custom transformers from a sequence of existing transformers
• Handle attribute references automatically in a custom transformer
Custom Transformer Input/Output Ports

Again, like a normal FME transformer, a custom transformer has a number of input and output ports:

These input and output ports are defined by input/output objects in the custom transformer definition itself:

Renaming Ports

The first thing to know is that these input/output objects can be renamed, in order that the transformer ports get named appropriately. You can either double-click the object, choose Rename from the context menu, or press F2, in order to rename the object.

For example, here the user is renaming an input port from StringConcatenator_In to simply Input.

Renaming the input and output ports is useful for making the custom transformer object more legible, and for helping the user to understand what data is supposed to connect to the port.

For example, after editing the transformer might look like this:

First Officer Transformer says...

Here the user has simply renamed the ports to Input and Output. However, renaming the input port to "Strings", "Lines", or "Raster" (for example) help guide other users of the transformer as to what data is required as input.

Likewise, the output port could be renamed to illustrate the type of data that will emerge; for example
Adding or Removing Ports

Besides renaming ports, it is also possible to add new ports to a custom transformer.

To do so simply click the tab to display the custom transformer’s definition and select Transformer Input (or Output) from either the canvas context (right-click) menu or the menu bar.

For example, here a user has added ports to handle two streams of input data, and has two output ports (one for the required output, another that handles “bad” features:

This means that each instance of the custom transformer in the main canvas will now have an extra input port, like so:

1 UPDATE

The VectorOnRasterOverlayer transformer in the above screenshot gained a <Rejected> port in FME2016.1
Miss Vector says...

Here are some questions for you.

Q) Which of these is NOT a reason to use Custom Transformers?

1. To make my content available in Quick Add
2. To use advanced functionality like looping
3. To reuse chunks of content in a simple way
4. To tidy and declutter the main workspace canvas

Q) Consider this section of workspace. If I select the three transformers highlighted with arrows, and create a custom transformer, how many input and output ports will it have by default?

1. One Input and One Output port
2. One Input and Two Output ports
3. Two Input and One Output ports
4. Two Input and Two Output ports
A colleague - new to FME - has created a workspace that calculates the population density for neighborhoods in the city of Vancouver, and comments that this technique could be reused for other projects.

We've turned her workspace into a custom transformer as a way of doing this and now need to show how to use it multiple times and apply edits to its definition.

**1) Start Workbench**

Open the workspace containing the embedded custom transformer:
C:\FMEData2016\Workspaces\DesktopAdvanced\CustomTransformers-Ex2-Begin.fmw

**2) Duplicate Custom Transformer**

Notice that we started with two ExpressionEvaluator and now have one ExpressionEvaluator and one custom transformer. Let's place another instance of the custom transformer in place of the ExpressionEvaluator.

Click on the ExpressionEvaluator and press the delete key to delete it.

Click on the DensityEvaluator custom transformer and press Ctrl+D (or right-click > duplicate) to create a duplicate copy of it.
This is the same effect as placing a new instance, but quicker. You could do the same task through Quick Add or the Transformer Gallery if you desired.

Connect the second DensityEvaluator into the workflow, in parallel and not in series:

**3) Set Custom Transformer Parameters**

By creating a second instance of the custom transformer we've started to re-use our content, which is great. However, the second instance is currently processing the wrong data.

Click the cog wheel icon to open the parameters dialog for the second DensityEvaluator. This time set the population parameter to TotalPopulation2011 (not 2001).
4) Run Workspace
Run the workspace and inspect the output to ensure the data is being processed correctly.

One obvious problem with the output from the transformer is that the result is put into an attribute called PopulationDensity2001, regardless of what data is being processed.

This is not useful; for example the 2011 results also get the same name, as would any other scenario where we used this transformer. We should improve this by making the output name more generic.

5) Edit Custom Transformer
Click on the tab labelled DensityEvaluator to switch the canvas to the custom transformer definition. Open the ExpressionEvaluator parameters and change the name of the New Attribute parameter to DensityResult.

If you run the workspace again you’ll notice that DensityResult is the attribute output by both instances of the custom transformer; i.e. one edit has fixed both of them!

6) Rename Ports
One other edit we ought to make is to the port names of the custom transformer. At the moment they are not very elegant.

Still within the DensityEvaluator tab, click the input port object in the custom transformer definition (currently labeled AreaCalculator_Input) and press F2 to edit the name. Change the name to Input.
Now repeat the process for the output port object, renaming it to Output.

Click the Main tab to check back on the main canvas and confirm the changes have been made:

CONGRATULATIONS

By completing this exercise you have learned how to:
- Use multiple instances of a custom transformer
- Make a custom transformer generic for use anywhere
- Rename output ports in a custom transformer
**Custom Transformers and Schema**

Schema Handling is one of the most misunderstood components in Custom Transformers. That’s because there are consequences that arise from allowing the re-use of content, and these consequences are not always apparent to the workspace author.

In brief, the ability to re-use a custom transformer means that it might be used in places where the schema does not match the custom transformer design.

---

**First Officer Transformer says...**

_The best analogy I can come up with is this: I have a laptop computer. I can use it at home and in the office. I can also use it overseas. The difference in power supply (110v against 240v) was considered by the manufacturers and the computer will work on both._

_In much the same way, the author of a custom transformer must be aware of the limitations that might apply if it is used outside of its expected area, and try to adapt to them. Schema is a key consideration._

---

**Handling Attribute Schema**

One part of the schema to be considered is the set of attributes available to the custom transformer.

For example, in this workspace a custom transformer carries out processing on incoming data using an attribute called AddressID as a key field:

![Diagram of Attribute Schema]

However, if that transformer is duplicated and used elsewhere, there is no guarantee that AddressID will exist:

![Diagram of Attribute Schema Variants]
These cases are both flagged red as "incomplete"; the first schema has ADDRESSID (not AddressID) and the other AddressUUID. Without FME's help, the end user would need to edit the transformer definition to fix these.

Therefore it’s vital that there is some form of mechanism for protecting against problems of a mismatched schema of this type. In fact there are two ways this can be handled: FME can automatically take care of the schema, or the workspace author can handle it manually.

Handling User Parameters

Besides attributes, the other part of the schema to be handled is user parameters.

Here, for example, a LabelPointReplacer transformer allows user input for label content and height:

Now let's assume that LabelPointReplacer is incorporated into a custom transformer, and that transformer used in several places in a workspace:

There's no problem with attributes, because the transformer isn't using any. But ideally each instance of the custom transformer will allow different values for the LabelPointReplacer parameters, in which case a single user parameter won't work. We need a mechanism for the user to enter different values.

As with attribute schema, there is an automated method for handling these, you just need to be aware of it in order to make some adjustments.
Automatic Schema Handling

Let's look at how we can handle the schema problems that might arise if a custom transformer is reused.

Automatic Handling of User Parameters

To take the handling of user parameters first, when a transformer with a published parameter is incorporated into a custom transformer, the published parameter is automatically moved from the main workspace to the custom transformer.

![Diagram showing parameter movement from workspace to custom transformer]

This means that the user is no longer prompted for these when the workspace is run! But... those parameters instead become available on the Custom Transformer itself:

![Diagram showing custom transformer interface]

That way the parameters can be set differently for each instance of the custom transformer. If user input is required then these new parameters can be published themselves - and shared if you want them all to have the same value.

Automatic Handling of User Attributes

Now let's look at how attributes are handled. When a custom transformer is created, one of the parameters in the Create Custom Transformer dialog is labelled Attribute References:

![Diagram showing Create Custom Transformer dialog]

Now let's look at how attributes are handled. When a custom transformer is created, one of the parameters in the Create Custom Transformer dialog is labelled Attribute References:
"Handle with Published Parameters" is the automatic way of handling attribute references in the custom transformer. It makes sure that every attribute referenced within the custom transformer is supported outside of the transformer definition.

It does that by creating a published parameter for each attribute:

...and then using that instead of the attribute reference:

Now, when this transformer is used in a place where the schema doesn't match, the transformer will still be flagged red as "incomplete". However... the user is able to use that published parameter to select an attribute that is available.

So (in the above) if AddressID is not available, the user can select ADDRESSID or AddressUUID instead.
First Officer Transformer says...

Referring to my previous analogy, without an adapter I would need to manually open up my laptop’s power supply, and rewire it with a new plug in order to use it overseas. However, this FME solution is like an adapter with a dial I just have to turn to the correct country setting.

This illustrates how FME has automatically solved the attribute reference problem using published parameters. To make the custom transformer more generic, the workspace author can change the prompts on these parameters; for example change the prompt from "AddressID" to "Select an ID Attribute to Process".

Miss Vector says...

What do you think would happen if you changed the parameter from "Handle with Published Parameters" to its other possible value, "Fix Manually (Advanced)"? Pick as many of these answers as you think are correct:

1. The workspace won’t run by default because no attributes are available in the custom transformer
2. There will be no way to pick attributes to use from the main canvas
3. The author will need to manually fix the custom transformer by exposing attributes in its definition
4. The custom transformer won’t work on a different schema unless the exposed attributes are also published
Post-Creation Schema Handling

As we know, Custom Transformers can be edited after they have been created.

The "Handle with Published Parameters" setting handles attributes used in a custom transformer when it is created, but there also needs to be a mechanism for handling edits to a custom transformer (or where the custom transformer is simply created from scratch).

This is achieved using a schema-editing button on the custom transformer.

For example, in this custom transformer definition the user concatenates a string using AddressID. AddressID is exposed in the custom transformer because it was being used when the custom transformer was created (and Handle With Published Parameters was set).

However, now the author wishes to add PostalCode to the concatenated string. By default they won't find PostalCode in the StringConcatenator, because it has not yet been exposed in the custom transformer. So they click the cog-wheel parameters icon on an input port:

This opens a dialog in which the incoming schema can be defined, and here the PostalCode attribute can be exposed:

Now PostalCode becomes available to the StringConcatenator and, additionally, back on the main canvas the custom transformer has a newly exposed parameter ready to accept an attribute selection should PostalCode not be available.

Handling Outgoing Attributes

Besides incoming attributes, there is also the question of what attributes should emerge from the output of a custom transformer.

Best Practice suggests that we really don’t want to output more attributes than are expected by the user. We should hide or remove any attributes that are part of a calculation, or any attributes that are otherwise generated inside the custom transformer but aren’t necessary to the output.

Here a custom transformer is calculating the average area of a number of polygon features. It has renamed ports and a specific output port to deal with bad features, but it is outputting more attributes than I would expect:
Basically, no effort is being made to clear up other attributes that are being used or generated, such as \_area, \_min, and \_max.

The workspace author should clean up this output. They can do this by visiting the custom transformer definition, clicking on the parameters (cog wheel) button for the output port object, and opening a dialog in which the attributes to be output can be defined:
Exercise 3 Re-using and editing a Custom Transformer

<table>
<thead>
<tr>
<th>Data</th>
<th>Neighborhoods (Google KML)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>Edit a custom transformer to use published parameters</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>Published parameters and custom transformers</td>
</tr>
<tr>
<td>Start Workspace</td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\CustomTransformers-Ex3-Begin.fmw</td>
</tr>
<tr>
<td>End Workspace</td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\CustomTransformers-Ex3-Complete.fmw</td>
</tr>
</tbody>
</table>

A colleague - with our help - has created a custom transformer that calculates density for a particular area. However, we need to work on it further to make it more generic - and to expand its capabilities.

This transformer was created using the automatic schema handling parameter and the workspace will already be handling schema properly. So, to an extent we don’t have to worry - but there are improvements we can make.

1) Start Workbench

Open the workspace containing the embedded custom transformer:
C:\FMEData2016\Workspaces\DesktopAdvanced\CustomTransformers-Ex3-Begin.fmw

Notice that, currently there are two instances of the custom transformer. Both produce an attribute with the same name (DensityValue). It would be helpful if the user of the custom transformer could define what the name of that attribute should be. Let’s set up the transformer to allow that.

Click on the tab labelled DensityEvaluator to switch the canvas to the custom transformer definition. Open the parameters dialog for the ExpressionEvaluator. Next to New Attribute click the down-arrow and choose User Parameter > Create User Parameter:

![ExpressionEvaluator Parameters](image)

When prompted, click OK and then OK again to close the dialog. Return to the main tab and check the parameters for each custom transformer instance. There should be the option to set the name of the attribute to output.

You can now move the instances so they are joined in sequence (rather than parallel) and change the two output attribute names to something different (PopulationDensity2001 and PopulationDensity2011):
WARNING
A fault in FME2016.1 means the transformer returns zero instead of the expected value. The problem does not occur in FME2016.0 and is fixed in FME2016.1.1 (build 16597 or greater). Apart from the incorrect result, the exercise is not otherwise affected by this problem.

First Officer Transformer says...
At first glance it might appear that we've simply reverted the custom transformer right back to where we started from. To an extent, that's true. However, the point is that we've now got a solution that can work in other scenarios (i.e. where something other than population density is being calculated).

2) Set Parameter Prompts
Looking at the custom transformer parameters we can also see that the prompt for the attribute to analyze is called "TotalPopulation2001". Obviously this is not very generic.

Return to the custom transformer definition tab and browse the Navigator window to find the related published parameter. Right-click on the parameter and choose Edit Definition.
In the dialog that opens set the parameter name to Density Attribute and the prompt to Attribute to Analyze:

Click OK to close the dialog. Return to the main tab and check the custom transformer parameters to prove the label change worked, and run the workspace to show that the output is still correct.

First Officer Transformer says…

It’s because we chose to handle schema automatically that we can simply change this user parameter name and not worry about where it is used. FME will handle the name change wherever is necessary.

3) Implement Units Selection

At the moment this workspace is calculating the number of items (in this exercise, persons) per square kilometre of land. This works for the original scenario, however, other uses of this transformer might find different units to be more useful.

Therefore we’ll implement a parameter for users to be able to select their units of choice.

In the custom transformer definition, browse the Navigator window and right-click on the entry labelled User Parameters. Select the Add Parameter option:

In the Add/Edit User Parameter dialog, set the following parameters:
Uncheck the check box parameter labelled Optional because the user has to select a value.

Now click the [...] button to the right of the Configuration parameter. This opens a dialog in which to define choices for the user to select from. Make two entries into this dialog:

<table>
<thead>
<tr>
<th>Display Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sq Metres</td>
<td>1</td>
</tr>
<tr>
<td>Sq Kilometres</td>
<td>0.000001</td>
</tr>
</tbody>
</table>

To save you counting, that's five zeros after the decimal place.

Click OK to close that dialog. Back in the Add/Edit User Parameter dialog set Sq Kilometres as the Default Value.

Then click OK to close this dialog and add the published parameter.

4) Implement Parameter

Now we've defined a published parameter that the user can set the units with, but we still have to apply it in the custom transformer.

Open the parameters dialog for the AreaCalculator transformer. For the Multiplier field, click the drop-down arrow and select the newly defined user parameter, DensityUnits.
Click OK to close the dialog.

Back in the main canvas the custom transformer now has a parameter for the end user to select the output density units. Experiment by running the workspace using different units, to prove that the changes were implemented properly.

---

**Advanced Exercise**

*Although it’s not needed for this population density calculation, another useful function for this transformer would be the ability to apply a weighting to the density calculations. If you have time, carry out the following steps to set it up.*

*The weighting will come from an incoming attribute, which means we need to be able to handle this in the custom transformer’s schema.*

---

5) **Add RandomNumberGenerator**

Our source data doesn’t have any fields we could reasonably use for weighting the output. Therefore return to the Main canvas tab and add a RandomNumberGenerator transformer in order to generate a test attribute:

Open the parameters dialog for the RandomNumberGenerator. For the purposes of this exercise set:

<table>
<thead>
<tr>
<th>Minimum Value</th>
<th>0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Value</td>
<td>1</td>
</tr>
<tr>
<td>Decimal Places</td>
<td>1</td>
</tr>
<tr>
<td>Result Attribute</td>
<td>WeightingAttribute</td>
</tr>
</tbody>
</table>
6) **Expose Attribute in Custom Transformer**

Now we have an attribute we need to expose it in the custom transformer, in order to use it.

Return to the DensityEvaluator tab where the transformer is defined. Click on the parameters button on the Input port object to open up a dialog named Edit Transformer Input.

Put a checkmark against the WeightingAttribute attribute and then click OK to close the dialog.

This will cause the attribute to be exposed in the custom transformer definition:

![Edit Transformer Input dialog](image)

It will also cause a user parameter to be created. Locate the parameter in the Navigator window (it should be called WEIGHTINGATTRIBUTE) right-click on it and choose Edit Definition.

Put a checkmark in the Optional field, as this should not be compulsory (the user might not have an attribute to weight the results by).

7) **Duplicate ExpressionEvaluator**

Now we can use the attribute inside the custom transformer.

Make a duplicate copy of the existing ExpressionEvaluator and connect it in parallel to the current one. Then put a Tester in beforehand where the Passed port goes to one ExpressionEvaluator and the Failed port goes to the other:

![ExpressionEvaluator duplication](image)

8) **Set up Tester**

Open up the Tester parameters dialog. Make a test for where WeightingAttribute > 0
9) Adjust Equation

Now that the attribute is exposed in the custom transformer, we can use it in the equation for calculating density. Open the parameters dialog for the ExpressionEvaluator transformer connected to the Tester:Passed port.

Change the equation to:

\@Value(TotalPopulation2001)/(\@Value(NeighborhoodArea)*\@Value(WeightingAttribute))

i.e. multiply the existing NeighborhoodArea attribute by the WeightingAttribute and place parentheses around that part of the expression.

Click OK to close the dialog and run the workspace to check the result. Remember – the results will be different every time because we’re generating the weighting attribute randomly at run time!

Experiment selecting the weighting attribute in the main canvas, and not selecting it. When no attribute is selected then the features should pass through the Failed port and no weighting is used in the calculation:

First Officer Transformer says…

It may seem odd – especially to experienced users – that we would use the attribute in the expression, and not the published parameter. But this is all part of how FME handles this behavior automatically. It avoids the author needing to know about published parameters and how to use them, and uses hidden functionality to replace the attribute with the published parameter wherever necessary.

CONGRATULATIONS

By completing this exercise you have learned how to:
- Publish an FME parameter inside a custom transformer
- Create a new user parameter inside a custom transformer
- Expose attributes inside a custom transformer
- Use an exposed attribute inside a custom transformer
Custom Transformer Types

There are two types of custom transformers.

An **embedded** transformer is one that exists only in the workspace itself. Its definition is stored (embedded) inside the workspace file and it is not available to any other workspace.

A **linked** transformer is one that exists outside of a workspace. Its definition is stored in its own file and it is available to any other workspace, which reference it through a link.

On a workspace canvas, embedded transformers are identified by their green color, while linked transformers are colored cyan:

---

**First Officer Transformer says...**

*It’s really important to know that the transformer type can be changed inside the workspace. You can switch an embedded custom transformer from being embedded to being linked, and switch a linked custom transformer to being embedded.*

---

Linked vs Embedded Transformers

Both type of transformer can be used in an FME workspace, and there are various advantages and disadvantages to each type.

**Embedded Transformers**

Embedded transformers are perhaps easier to understand, need no external files, and their definition is embedded into the workspace. They are particularly useful for tidying a workspace but also work for employing advanced functionality like parallel processing.

However, sharing and content re-use is not as simple with an embedded transformer. The custom transformer cannot easily be shared with other users, unless they are given a copy of the same workspace, and it is not easy to maintain a consistent definition among several users.

**Linked Transformers**

Linked transformers are perhaps a little harder to understand and manage. They exist as a file (.fmx) outside of the workspace which is less convenient, and employing advanced functionality like loops can be more complex.

However, a linked custom transformer is a little easier to edit (you open the .fmx file rather than the .fmw file) and is much easier to share among users. Not only can the file be given to any FME author to use, any number of authors can actually point their FME to the same custom transformer file.

Sharing the same file is useful because any changes made to the definition are automatically propagated to all workspaces that use it.
First Officer Transformer says...

Like embedded transformers, linked transformers also show up in the transformer gallery and Quick Add dialog. Also notice that they have a special icon to signify that you are about to use a linked version, rather than the embedded:
Creating a Linked Custom Transformer

All custom transformers start out as an embedded version. To create a linked version the custom transformer is exported from the workspace.

This is easily done by clicking on the canvas tab for the embedded definition and choosing File > Export as Custom Transformer from the menubar:

---

**First Officer Transformer says…**

It’s best to debug your custom transformers before exporting them, because the Run with Breakpoints tool does not work inside exported transformers, only embedded.

---

At this point a dialog opens in which you can confirm the transformer name and category, plus some other parameters including the save location:
Let's look at some of the different options:

**Name and Category**

It's obvious what these are - the same fields as when the transformer was originally created - but it's interesting to note that you can save it with a different name/category to what was used for the original version.

**Insertion Mode**

The Insertion Mode parameter specifies how the custom transformer can be used. *It is an important parameter when I am the author of a custom transformer, who has created it for other workspace authors to use.*

There are four different modes:

If I export my custom transformer in **Embedded** mode, it means that an author who uses this transformer in his workspace finds it gets embedded (rather than a link to a file).

If I choose **Embedded By Default** it means that the author can choose to make the transformer linked instead. If I choose **Embedded Always** it means that the author is stuck with using that custom transformer in embedded mode.

Similarly, if I export my custom transformer in **Linked** mode, it means that an author who uses this transformer in his workspace finds it gets linked to a file (rather than embedded).

If I choose **Linked By Default** it means that the author can choose to make the transformer embedded instead. If I choose **Linked Always** it means that the author is stuck with using that custom transformer in linked mode.

**Which Mode to Use**
**Embedded Always** is a good choice when the person using the transformer is less experienced with FME; it's easier for them to manage and if they make changes they won't affect other people. Embedded is also a good choice where the custom transformer is intended for use by individuals (i.e. not sharing it as a group).

**Linked Always** is a good choice when the custom transformer is intended to be shared among a group of users. Because it is linked, the users will always receive updates if the transformer definition is changed, and because the definition is shared it becomes a standard that is applied to all users.

Only when the end-user is experienced in FME and can understand the consequences, is it advisable to use a “By Default” setting and allow type switching.

**Password**

The password field allows you to password-protect the custom transformer. This will make it impervious to edits from unauthorized persons. Additionally, the file contents are (very mildly) encrypted so that they cannot be copied by opening the source file in a text editor.

This allows authors to make transformers available for purchase without any fear that their work will be copied or edited. Of course, it's important not to forget or lose the password yourself, in case you wish to make edits!

**Licensing**

Although not part of the export dialog, it's worth mentioning licensing along with passwords. Custom transformers can be licensed so that they cannot be used without the proper registration code. This is of benefit when you want to restrict access (perhaps within your own organization) or you want to license a transformer just like any other item of software (perhaps you made it for general sale to users).

A special transformer – the LicenseChecker – and a license generator tool are provided for authors to implement such a setup:

Here, for example, the LicenseChecker is being used to protect a custom transformer. If the transformer is licensed then it will work as expected. If it is not licensed then it will terminate.

For more information on obtaining licenses for a custom transformer, contact the Safe Software support team.

**Save Location**

FME has a specific installation folder in which custom transformer files can be saved. If a custom transformer is saved in this folder then it becomes available in Workbench and can be used the same as any other transformer. If it is saved elsewhere then FME won't be able to find it unless that path is set under Tools > FME Options > Default Paths.

---

**Miss Vector says…**

*Here is a question for you to investigate: Can you nest custom transformers? That is, can you put one*
**custom transformer inside another?**

1. Yes, with no restrictions
2. Yes, but you can only nest transformers of the same type (Linked or Embedded)
3. Yes, but you cannot nest Linked Custom Transformers
4. Yes, but only a single level of nesting
Switching Custom Transformer Types

When a custom transformer is designed to allow type switching, to switch a custom transformer instance from embedded to linked mode is very straightforward. Simply right-click on the instance and choose Link:

Of course, to be able to switch types requires that you have already exported the custom transformer. If you have not, there will be no option to link the transformer because there will be no file to link to!

When you have done this then the custom transformer definition tab will be closed, and the custom transformer replaced with one that simply references the external fmx file definition.

If there is more than one instance of the Custom Transformer, you will be asked whether you want to switch all of them:

The usual answer to this is yes, because having the same transformer both embedded and linked could be quite confusing!

In a similar manner to above, to switch from Linked to Embedded right-click and choose the option Embed.

First Officer Transformer says…

Switching from Embedded to Linked only works as long as the two versions are the same. In other words, if you embed a linked transformer and then make changes to the embedded definition, you won’t be able to revert to the linked version.
Custom Transformer Versioning

FME includes functionality so that a linked custom transformer can exist as a number of versions. In short, each time a custom transformer definition is edited, a new version can be saved.

In that way a single fmx file can contain multiple versions of the same custom transformer.

Why use Versioning?

One advantage of versioning a custom transformer is that you have a record of previous versions and therefore can revert to a previous one should the need arise. For example, maybe some recent edits were incorrect and you need to switch back to the definition that existed before those edits were made.

However, a more important advantage relates to FME releases and new functionality.

For example, a custom transformer created in FME2015 and shared by many users could be updated to use new behaviour in FME 2016. If that custom transformer is versioned then the 2015 version remains available to users who have yet to update to that version of FME; while users who have updated their FME can also update their custom transformer to take advantage of the new updates.

Creating a Versioned Custom Transformer

Versioning can occur when you edit a linked custom transformer definition (usually an unversioned one) and then attempt to save it. In that situation a dialog will open to ask what you wish to do:

The two options are to overwrite the existing version or to create a new version. Creating a new version does not create a separate fmx file; instead it creates a separate version of the transformer in the same fmx file.

The title bar in Workbench also changes to illustrate that this is now a new version.

First Officer Transformer says…

Subsequent saves don’t update the version number. A new version is only created for a new edit session; i.e. the first time you save the transformer after it is newly opened in Workbench. If you keep on making edits, it just saves to the same version. You need to close and reopen the file to create another new version.
Editing a Specific Transformer Version

Whenever a versioned custom transformer is initially opened in Workbench, you are prompted as to which version you wish to edit, or whether you want to just start with a new version:

This way you are able to:

- Create a new version to make edits to
- Continue making edits to the existing version
- Make edits on an older version (particularly useful when that version is tied to a particular FME release)

Updating a Transformer Version

Whenever a workspace contains a linked custom transformer, you get the same version information (in summary annotation and the like) as a regular transformer:

However, if FME detects that a new version is available, then an option appears on the context menu to allow an update to the new version:

Choosing to upgrade means the summary annotation would display the newest version number.

Miss Vector says...

What do you think would happen in this scenario? You have a workspace with a linked custom transformer (version 1). The author of that transformer makes a series of edits and updates it to version 4. What will the upgrade option do to the custom transformer in your workspace?
1. Upgrade it to version 2
2. Upgrade it to version 3
3. Upgrade it to version 4
4. It depends on what version of FME you and the author are using
**Exercise 4  Custom Transformer Modes**

<table>
<thead>
<tr>
<th>Data</th>
<th>Bicycle Routes (Esri Shapefile)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overall Goal</strong></td>
<td>Create a custom transformer to calculate the average length of a number of linear features</td>
</tr>
<tr>
<td><strong>Demonstrates</strong></td>
<td>Custom Transformer Modes</td>
</tr>
<tr>
<td><strong>Start Workspace</strong></td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\CustomTransformers-Ex4-Begin.fmw</td>
</tr>
<tr>
<td><strong>End Workspace</strong></td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\CustomTransformers-Ex4-Complete.fmw C:\FMEData2016\Workspaces\DesktopAdvanced\AverageLengthCalculator.fmx</td>
</tr>
</tbody>
</table>

You arrive early at the office for a meeting, but it is cancelled at the last minute. Typical! Still, it gives you time to carry out an FME project that has been on your mind: a transformer to calculate the average length of linear features.

1) Open Workspace

Open the workspace C:\FMEData2016\Workspaces\DesktopAdvanced\CustomTransformers-Ex4-Begin.fmw

You’ll see that the workspace is reading a set of bicycle path data, and then doing some minor processing to get it into a reasonable state for use in the custom transformer.

You may want to run the workspace to examine the output and see what data we are dealing with; but remember the custom transformer we will create is to be designed to work on any linear data.

2) Add LengthCalculator

The contents of the transformer will be fairly straightforward and we’ll start out with just two transformers. So, simply add a LengthCalculator and a StatisticsCalculator transformer to the workspace.

![Diagram](image)

.1 UPDATE

The LengthCalculator transformer in the above screenshot gained a `<Rejected>` port in FME2016.1

3) Create Custom Transformer

Select the two newly placed transformers and turn them into a Custom Transformer called AverageLengthCalculator. Make sure the attribute references are handled automatically, although at the moment there aren’t any references to handle.
4) Edit Custom Transformer

Now we have a new custom transformer, let’s tidy it up and make it functional.

Firstly rename the input port object to Lines (thus identifying what geometry is expected), then add an output port object (if you don’t have one already) and rename it to Output. It should be connected to the StatisticsCalculator:Complete port:

Open the StatisticsCalculator parameters. Set Attributes to Analyze to _length. Rename the Mean Attribute result to AverageLength

Finally, open the parameters for the Output port object. Change Attributes to Output to ‘Specified Attributes Only’ and ensure that AverageLength is output, but _length and any other StatisticsCalculator attributes are not:
5) Run Workspace
Run the workspace (unless you need to reattach an Inspector transformer, you don’t even have to return to the Main tab to do this). Inspect the output to ensure everything is working as expected.

6) Export Custom Transformer
Now let’s experiment with different transformer modes.

Select File > Export as Custom Transformer from the menubar. In the Export as Custom Transformer dialog make sure the Insert Mode option is set to Linked by Default. Make sure the Save Location is the default for storing custom transformers (C:\FME\Transformers):

Click OK to close the dialog. The custom transformer is saved (as AverageLengthCalculator.fmx) and this file opened up in a new instance of FME Workbench.

7) Examine Workspace
Go back to the instance of FME Workbench where the original workspace is open. The custom transformer is now a cyan color to denote that it is now a linked transformer (it’s linked because we chose “Linked By Default”):

Notice that you can right-click and choose to embed the transformer, and then switch back to the linked version. In a real-life scenario, which you choose would usually depend on whether you are planning to share the transformer.

In embed mode, right-click the transformer and choose Edit. You’ll find that you can no longer change back to Linked mode, because the two definitions are now considered different!

Delete the embedded transformer. You’ll be prompted whether you wish to delete the definition too. Click Yes.
8) Examine Custom Transformer
Place a new instance of the custom transformer in the workspace (it will be linked by default, which is fine). It you hover the mouse cursor over the transformer the pop-up text will show that it is version 1.

Go back to the instance of Workbench where the fmx file is open. Move one of the objects about to activate the save button. Then save the file. Notice that you aren’t prompted to save a new version. That’s because you’re still in the same session. Any edits you make here will go towards the same version, until you close and reopen the file.

So, leaving Workbench open, close the fmx file. Then go to the start tab and select it from the recent Files list. Now it is reopened, any changes we make will go towards a new version.

9) Update Custom Transformer
Rather than just jiggling objects about to prove a point, let’s make a real update to this transformer. One thing we could do is filter data by geometry, so we aren’t trying to measure the length of a point feature, or similar.

So, add a GeometryFilter transformer, in front of the LengthCalculator:

Open the parameters and select Line and Arc as the geometries to filter by. Then click OK to close the dialog.

Adjust the feature mapping so that the Line and Arc ports are directed into the LengthCalculator. Add a second output port object by right-clicking on the canvas and selecting Insert Transformer Output. Call the newly placed port Rejected and connect the data to it, like so:
Now click the save button to save the custom transformer. You’ll be prompted whether you want to create a new version. Click the button labelled New Version to do so.

10) Update Workspace
Go back to the instance of FME Workbench where the original workspace is open. Click on the refresh button on the Transformer Gallery in order for FME to scan all custom transformers and discover the new version we’ve just created.

Now right-click on the AverageLengthCalculator custom transformer and there should be an option to upgrade to the latest version. Choose this option:

The transformer will be refreshed and updated, which you can tell by the presence of a Rejected port.

First Officer Transformer says…

Now you have this custom transformer you have two options to share it.

1. You can put the transformer into a shared folder and then have other users use Tools > FME Options > Default Paths to link their FME to that shared folder.

2. You can send (email) the fmx file to other users and have them install it in their FME. They can install it either by double-clicking the file or saving it to their default FME resource folder.

CONGRATULATIONS
By completing this exercise you have learned how to:

- Export a custom transformer
- Switch the mode of a custom transformer instance
- Delete an embedded custom transformer
- Edit and create a new version of a linked custom transformer
- Update a linked custom transformer instance to the latest version
Custom Transformers and Parallel Processing

Parallel Processing is a way to improve performance on high-end machines.

As noted in the Performance chapter, some FME transformers have the option to allow parallel processing. However, custom transformers have a special mechanism to do this. Whereas not all FME transformers allow parallel processing, you can apply this technique to ANY custom transformer that you like!

Setting up Custom Transformer Parallel Processing

Parallel processing for a custom transformer is set up in the Navigator window.

Each custom transformer has a set of transformer parameters that specifically relate to parallel processing. Here you can determine the level of parallel processing, and the attribute that is going to be used to define the processing groups:

By default, these are set to not carry out parallel processing. However, when the author sets a level of parallelism then the Parallel Process By parameter becomes active and a user parameter is automatically created:

First Officer Transformer says…

Because of how parallel processing works in a custom transformer, you can’t use an attribute for the Parallel Process By parameter. Instead you have to make use of a user parameter that references an attribute.
In short, you can’t select an attribute in this dialog, only user parameters.

The published parameter means that the end user is able to set the attribute to group-by for parallel processing. For example, here the custom transformer is creating a separate process for each different park feature:

If, as an author, I don’t want the end user to be setting the group-by, then what I can do is locate that published parameter, edit its definition, and unset the Published parameter:

First Officer Transformer says…

Are you using raster data?

Raster is an oddity in FME as most of the transformers do very little to the data. For example, the RasterResampler doesn’t actually resample the data; it just tags it as being resampled. The actual
resampling is carried out when the data is written.

On the one hand this is great. It means – for example – if you resample then clip some raster data, FME knows to resample only data that falls inside the clip boundary, as the rest is ultimately going to be discarded.

On the other hand, it does mean that parallel processing doesn’t help performance that much, as most work occurs in the Writers. That’s why few raster transformers have parallel processing options, and why it’s not worth doing in a custom transformer.
Exercise 5  Parallel Processing with Custom Transformer

<table>
<thead>
<tr>
<th>Data</th>
<th>3D Point Clouds (ASPRS Lidar Data Exchange Format (LAS))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>Create a custom transformer to parallel process data</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>Custom Transformers and Parallel Processing</td>
</tr>
<tr>
<td>Start Workspace</td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\CustomTransformers-Ex5-Begin.fmw</td>
</tr>
<tr>
<td>End Workspace</td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\CustomTransformers-Ex5-Complete.fmw</td>
</tr>
</tbody>
</table>

The city has recently started collecting point cloud data and now it is ready for sharing with different departments. You have been asked to create a solution that converts the point clouds to a vector format that other departments can use.

You quickly create a great workspace that nicely tiles and thins the data too so the destination datasets aren’t overwhelming in terms of size.

However... the workspace takes longer to run than you like. Because it will be run on a daily basis it would be useful to speed up the translation using parallel processing.

Since none of the transformers used has a parallel processing parameter, you’ll have to create a custom transformer to do this.

1) Open Workspace
Open the workspace C:\FMEData2016\Workspaces\DesktopAdvanced\CustomTransformers-Ex5-Begin.fmw

As you’ll see, this workspace processes some incoming point cloud data. Inspect the data to see what we’re dealing with. If you run the workspace as-is it will take approximately three minutes. To make it run a little faster you can increase the Thinning Interval parameter in the PointCloudThinner (say to 25).

Open a task manager (process manager) tool for your operating system. Run the workspace. You’ll see a single FME engine process running (fme.exe):

<table>
<thead>
<tr>
<th>Image Name</th>
<th>User Name</th>
<th>CPU</th>
<th>Memory (K)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>explorer.exe</td>
<td>imark</td>
<td>00</td>
<td>41,992K</td>
<td>Windows Explorer</td>
</tr>
<tr>
<td>fme.exe #32</td>
<td>imark</td>
<td>25</td>
<td>55,440K</td>
<td>FME EXE</td>
</tr>
<tr>
<td>fmedatadinspector.exe #32</td>
<td>imark</td>
<td>25</td>
<td>231,404K</td>
<td>FME Data Inspector</td>
</tr>
</tbody>
</table>

First Officer Transformer says...

You’ll also see an fmeworkbench.exe process, which is the process running the Workbench interface. This isn’t responsible for processing the workspace; the two are completely separate processes.

2) Create Custom Transformer
Now select the PointCloudThinner and PointCloudCoercer transformers and turn them into a custom transformer.

It’s important you don’t include the Tiler transformer, as this is creating the tiles that we’ll be using as a way to parallel process.

You can call the transformer something like PointCloudProcessing. It doesn’t matter what attribute reference handling you choose.

The transformer definition should look something like this:
3) **Set Parallel Processing**

In the Navigator window (of the custom transformer definition) locate and expand the section of custom transformer advanced parameters.

Double-click the Parallel Processing Level parameter to set it. Set the processing level to Moderate.

Click OK to close the dialog and you’ll notice the Parallel Process By parameter is now published.

4) **Set Process By**

Return to the main canvas and click on the parameters button for the custom transformer instance. Select both `_column` and `_row` as the attributes to process by:

This means that each unique combination of `_column` and `_row` (i.e. each tile) will be run under a separate process, up to a maximum of one process per core processor.
5) Run Workspace

Run the workspace, again with a task manager window open. Once the tiling is complete and the rest of the workspace is being processed, you’ll notice a number of FME worker processes (fmeworker.exe).

In moderate mode, you’ll see up to one fmeworker process for each core. This time the translation should be complete in nearly half the time, approximately one minute and thirty seconds.

6) Experiment with Parallel Processing Level

If you have time, re-run the workspace with a different processing level, say Aggressive. Does it run any quicker than the Moderate processing level? If not, why might that be? Does adjusting the number of tiles make it better or worse?

CONGRATULATIONS

By completing this exercise you have learned how to:
- Create and use parallel processing in a custom transformer
- Confirm with the task manager that FME is launching worker processes
Custom Transformers and Loops

Loops are a way to carry out a process that repeats a section of transformers.

What is a Loop?

A loop is a programming structure that allows an action to be repeatedly executed.

Often this is used to carry out iteration; where a process repeats to gradually narrow down the process to a desired result. Usually a loop is linked to a condition; i.e. the action continues until a certain condition is met.

In FME, loops are only permitted inside a custom transformer.

First Officer Transformer says...

I often get into a loop when flying. I have to circle the airport again and again (the action), until I have clearance to land (the condition). Users who have flown into London Heathrow will understand what I mean!

Why Use a Loop?

As you know, FME processes one feature at a time. Therefore, when you create a loop, each feature is being sent round the loop individually.

However, the essence of a loop is that each iteration is slightly different. Why else would you want to repeat the same action again and again, if the result was going to be no different?

So, to be worthwhile, each iteration of the loop must either fetch new data (for example, it reads another entry from a list) or it repeats the same process but using the results of the previous loop.

NB: In the following screenshots, the processing section is a single transformer labelled "ProcessData", in order to give a generic view of how a loop is created. In reality, this process must be doing something to the data in order to be worthwhile carrying out at all.

First Officer Transformer says...

It’s stating the obvious I know, but you only use a loop to repeat an action inside the custom transformer. One-off actions should take place outside of a loop. For example, here is my landing sequence:

The “wheels down” part comes outside the circling loop, because I don’t want to raise/lower the wheels every time I circled the airport. That would be very inefficient.
Setting up a Custom Transformer Loop

A loop in a custom transformer requires two components: the start and end point of the loop.

Loop Start Points

The start of the loop is identified by an Input port object. Although it can be the same input port as used for features to enter by, this does not have to be the case. For example here there is an input port for features to arrive into, and another one for the start of the loop:

This allows the loop point to be other than the beginning of the custom transformer.

By default, this second input port also appears on the transformer itself, like this:

If you don’t require this, then you simply have to open the Input port’s parameters and ‘unpublish’ it:

Loop End Points

The end of a loop is identified by a Loop object. You can insert one by selecting it from the canvas context menu in a custom transformer:
When you place a loop object you are asked which Input object is to be looped to:

And then the loop is complete:

Of course, this example is an infinite loop. The action is repeated but there is no condition being tested to stop it. FME won't let an infinite loop run forever - it will recognize the problem and stop it - but we should set up something to force an ending.

**Loop Conditions**

There are two general types of condition we can test for. Firstly we can loop a set number of times. Secondly we can loop until a specific condition is met.

Here is a custom transformer that loops a set number of times:
Notice that we have an attribute that is a counter for the number of times we have looped, and an attribute that tells us the maximum number of loops to carry out. In each loop the counter attribute is incremented by 1. When \( \text{LoopCounter} < \text{MaxLoops} \), then we loop back and process the data again. When \( \text{LoopCounter} = \text{MaxLoops} \), then we exit the transformer.

Instead of a simple count, another method is to replace the counter with a measure of data quality.

**First Officer Transformer says…**

*For an excellent example of looping in an FME workspace, check out [this customer story](#)*

In that example, the user uses a loop to place trees (the action) until a certain density is reached (the condition). Notice that the loop is not tied to a specific counter - it continues until the data quality required is met.

**Loops and Transformer Types**

As you should already know, transformers that operate on one feature at a time are called Feature-Based, and transformers that operate on multiple features at a time are called Group-Based.

We can also call a loop "Feature-Based" because it only processes one feature at a time. Unfortunately, that means there are issues around using a group-based transformer inside a (feature-based) custom transformer loop.

If you attempt to create a loop inside an embedded custom transformer, when it includes a group-based FME transformer, then you will receive an error message. Group-based transformers are only permitted inside a loop in a linked custom transformer. There are technical reasons for this that we won't go into right now.

This is the error message you will get:

So, inside a linked custom transformer definition, you'll see a particular parameter (in the Navigator window) called Enable Blocked Looping:
When set to Yes then other parameters are exposed to set the number of iterations and an attribute that will hold that value. Notice how parallel processing is turned off (the parameters are removed) for custom transformers that are being looped, and the Insert Mode is automatically changed to "Linked Only".

**Miss Vector says…**

Which of these statements about loops are true?

1. Loops are only permitted inside a custom transformer
2. A loop without a condition will continue processing until manually stopped
3. Test conditions are built into the loop end point parameters
4. Nested loops (a loop within a loop) are permitted
Exercise: Looping

Exercise 6: Looping in a Custom Transformer

<table>
<thead>
<tr>
<th>Data</th>
<th>Address Data (Esri Geodatabase (File Geodb API))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>Create a custom transformer to calculate a check digit</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>Custom Transformers and Loops</td>
</tr>
<tr>
<td>Start Workspace</td>
<td>None</td>
</tr>
<tr>
<td>End Workspace</td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\CustomTransformers-Ex6-Complete.fmw</td>
</tr>
</tbody>
</table>

Due to a horrible mistake (don't worry, it wasn't yours) a city workcrew were given an incorrect address and accidentally demolished the house of the city's police chief!

You've been asked to come up with ideas to help prevent this from reoccurring, and the quickest method you can think of is to add a check digit to all address ID numbers.

Having taken FME training(!) you realize this can be done using a custom transformer loop.

Wikipedia says...

A check digit is a form of redundancy check used for error detection on identification numbers (e.g. bank account numbers) which have been input manually. It is analogous to a binary parity bit used to check for errors in computer-generated data. It consists of a single digit (sometimes more than one) computed by an algorithm from the other digits (or letters) in the sequence input.

A looping transformer is useful here because it can take each digit in an ID number, one at a time, and add its value to a running count. The last digit of the final count will be the check digit. Looping is particularly good because it can be applied to any length of ID attribute.

1) Start Workbench

Start FME Workbench. Use Readers > Add Reader to add a Reader with the following specifications:

<table>
<thead>
<tr>
<th>Reader Format</th>
<th>Esri Geodatabase (File Geodb API)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reader Dataset</td>
<td>C:\FMEData2016\Data\Addresses\Addresses.gdb</td>
</tr>
</tbody>
</table>

When prompted, the only table you need to select is PostalAddress

2) Add StringLengthCalculator

We want FME to handle attributes with published parameters. It doesn't make a lot of difference, but it's slightly easier to create the custom transformer using at least one transformer that uses the main attribute (OBJECTID)

So, place a StringLengthCalculator transformer and use it to calculate the length of the OBJECTID attribute:
3) Create Custom Transformer
Now select the StringLengthCalculator and create a new Custom Transformer. You can call it CheckDigitCalculator.

Edit it to make sure it has properly named input and output ports:

4) Initialize Loop
Now let's set up some attributes to initialize the loop. Inside the custom transformer definition place an AttributeCreator transformer and use it to create two new attributes: loopCounter and digitSum. Set both of their initial values to 0 (zero).

loopCounter will be the position of the digit we are processing, and digitSum will be the total of the digits processed.

5) Add SubstringExtractor
This is where we start to process the data. Add a SubstringExtractor transformer to fetch the next digit of the ID string. The Start and End Index parameters will be set to the value of loopCounter:
This way, as the loop is incremented, we fetch subsequent characters from the ID string.

6) Add ExpressionEvaluator
This is the other key transformer of the processing part. Place an ExpressionEvaluator transformer after the SubstringExtractor.

Open the parameters dialog. Set up the transformer to add the value of the extracted substring to the current digitSum value:

7) Add ExpressionEvaluator
The main part of the processing is done. Now all we need to do is implement the conditional part of the looping. So, add a second ExpressionEvaluator and use it to increment the loopCounter attribute by 1:

8) Add Tester
Next we need to test whether we have reached the end of the ID string. Add a Tester transformer and set it up to test whether loopCounter = _length (i.e. has the transformer looped n times, where n is the length of the ID string).

The Tester:Passed port can be linked to the custom transformer Output port.
9) Add Loop Start Point
Before we can add the loop "object" we need to define where to loop back to. In this case we want to loop back to where the processing takes place, just after the loop count initialization.

So, add a new Custom Transformer Input port. Call it loopInput and connect it to the SubstringExtractor transformer input port:

Open the properties dialog and uncheck the published button to hide this port from the main canvas.

10) Add Loop End Point
Now let's add the loop end point object. Right-click on the canvas and choose to add a Transformer Loop. When prompted, choose loopInput as the input port to loop back to:

Connect the loop object to the Tester:Failed port:

11) Add SubstringExtractor
The result of this translation will be a number that is a total of the digits in the ID attribute. For example, if OBJECTID = 5621 then digitSum will be 14 (5+6+2+1).

However, a check digit is normally a single digit - the last digit of our sum (in the above example, 4).

So, add a second SubstringExtractor transformer and connect it between the Tester:Passed and Output ports:
Open the parameters dialog and set it up to extract just the final digit of digitSum and create a new attribute called CheckDigit:

-1 means the final digit in the string.

12) Clean Up Attributes
One final thing to do: clean up the attributes emerging from the transformer. Open the properties dialog for the Output port and set it up to output the CheckDigit attribute, but no others.

13) Run Workspace
Add an Inspector to the main canvas and run the translation. You could also optionally add a transformer to concatenate the old ID (OBJECTID) and the check digit (CheckDigit) to give a new ID.

The output should be something like this:

The final digit of the new address ID field will always be the sum of the preceding digits, meaning there is more ability to check for mistakes before anything bad happens!

CONGRATULATIONS

By completing this exercise you have learned how to:
- Create a loop object in a custom transformer
- Create and increment a loop counter in a custom transformer
- Use a loop counter to loop through content in a custom transformer
- Add an unpublished input port and use it for a loop target
- Prevent excess attributes exiting a custom transformer
Module Review

This chapter investigated Custom Transformers and how they can be used to improve your FME workspaces

What You Should Have Learned from this Module

The following are key points to be learned from this session:

Theory

- A custom transformer is a sequence of standard transformers condensed into a single transformer.
- A custom transformer lets you tidy your workspace, re-use sequences of transformers, and apply some advanced functionality like looping.
- Handling schema in a custom transformer is very important, and it can be done either automatically or manually.
- Custom transformers can be either embedded or linked.
- Custom transformers can be used to implement parallel processing.
- Loops in FME can only be implemented in a custom transformer.

FME Skills

- The ability to create, edit, and reuse a custom transformer.
- The ability to handle schema in a custom transformer
- The ability to embed or link transformers and to share them with colleagues
- The ability to apply parallel processing in a custom transformer
- The ability to use custom transformer loops

Further Reading

For further reading why not browse articles relating to Custom Transformers on our blog?
Questions

Here are the answers to the questions in this chapter.

Miss Vector says...

Which of these is NOT a reason to use Custom Transformers?

1. To make my content available in Quick Add
2. To use advanced functionality like looping
3. To reuse chunks of content in a simple way
4. To tidy and declutter the main workspace canvas

Admittedly it does become available in Quick Add, but that's not a specific reason to create a custom transformer. The reason you want it in Quick Add is so that you can reuse the content.

Miss Vector says...

Consider this section of workspace. If I select the three transformers highlighted with arrows, and create a custom transformer, how many input and output ports will it have by default?

1. One Input and One Output port
2. One Input and Two Output ports
3. Two Input and One Output ports
4. Two Input and Two Output ports

It will have two input and two output ports, proving that FME will automatically create multiple ports where required. There are two inputs because there are two connections entering the set of transformers, and two outputs because there are two connections exiting the set of transformers.
Miss Vector says...

Why do you think that we left the CSMapReprojector transformer out of our custom transformer (in the exercise)?

It's because the CSMapReprojector is not a vital part of the process and wouldn't be needed in all cases. Only where the coordinate system wasn't compatible with our area measurements would we need it. For that reason we'll leave it out of the custom transformer - but maybe add a note to the usage setting to say that data needs to be in a specific coordinate system to use this custom transformer.

Miss Vector says...

What do you think would happen if you changed the parameter from "Handle with Published Parameters" to its other possible value, "Fix Manually (Advanced)"? Pick as many of these answers as you think are correct:

1. The workspace won't run by default because no attributes are available in the custom transformer
2. There will be no way to pick attributes to use from the main canvas
3. The author will need to manually fix the custom transformer by exposing attributes in its definition
4. The custom transformer won't work on a different schema unless the exposed attributes are also published

Yes, when the parameter is set to manual, it really means manual! All four of these are true, meaning you'll have your work cut out if you don't let FME take care of attribute references for you.

Miss Vector says...

Can you nest custom transformers? That is, can you put one custom transformer inside another?

1. Yes, with no restrictions
2. Yes, but you can only nest transformers of the same type (Linked or Embedded)
3. Yes, but you cannot nest Linked Custom Transformers
4. Yes, but only a single level of nesting

Yes you can embed any type of custom transformer inside any other type of custom transformer, to multiple levels of nesting.

Miss Vector says...

You have a workspace with a linked custom transformer (version 1). The author of that transformer makes a series of edits and updates it to version 4. What will the upgrade option do to the custom transformer in your workspace?

1. Upgrade it to version 2
2. Upgrade it to version 3
3. Upgrade it to version 4
4. It depends on what version of FME you and the author are using

It will upgrade the custom transformer to the latest version that is compatible with the FME version you
are using. If you are using the same version of FME, then it would upgrade the custom transformer to version 4. But if you are using different versions of FME then it could be version 2 or 3 instead (or maybe there would be no available updates at all!)

Miss Vector says...

Which of these statements about loops are true?

1. **Loops are only permitted inside a custom transformer**
2. A loop without a condition will continue processing until manually stopped
3. Test conditions are built into the loop end point parameters
4. **Nested loops (a loop within a loop) are permitted**

Yes, loops only work in a custom transformer. An endless loop will not continue forever (FME will stop it after a time). Conditions need to be checked with transformers (like the Tester). And Nested Loops are permitted. The following is a screenshot of a nested loop custom transformer:

Notice that there are now two count attributes (one for each loop). The first (inner) loop counter is reset to zero every time the second (outer) loop counter is incremented.
Advanced Reading and Writing
Zip File Handling

The Basic Desktop course covered how data can be read directly from a zip file. However, it is also possible to write data to a zip file.

Zip files are a convenient way to write output datasets that need to be handled as a single unit.

For example, a single Shape feature type consists of several files; shp, shx, dbf, prj, etc. A Shape dataset can consist of multiple feature types. So, in a scenario where the output data needs to be post-processed – uploaded to a web site, say – it’s more convenient to handle a single zip file than multiple data files.

Zip File Writing

The simplest way to create a zipped output is to simply change the file extension to .zip in the output dataset field:

A shortcut button for setting a zip extension exists on most Writer dialogs:

Once set, a small icon in the dataset field indicates the zipped status:

When the workspace is run the log file reports the zip creation:

Finished updating output zip file `C:\FMEData2016\Output\Parks.zip'

And the output is, indeed, a zipped dataset:
Sister Intuitive says…

I’m Sister Intuitive from the order of Perpetual Translations. I’ll provide you with spatial guidance throughout this chapter.

Some users may want to zip data in order to upload it to a different location as a single entity. A user parameter can be used in a TCL or Python script to find the name of the file just written, and the FeatureWriter transformer (new in FME2016) also provides the name of the dataset as an attribute.
Fanouts

Fanouts are one of the most powerful pieces of functionality within FME, capable of producing impressive results with very little effort.

What is a Fanout?

Fanouts are a way for the workspace author to divide data up into groups of features in the output dataset. The groups are defined by either a single attribute or a string constructed from a combination of attributes and fixed strings.

For example, here an author is "fanning-out” a set of data into multiple outputs depending on a feature’s elevation attribute:

Because a fanout occurs as the data is being written, it does not require multiple flows of data inside the workspace. Therefore this technique makes it easy to create groups with minimal impact on the workspace canvas.

Another major benefit of a fanout is the high degree of flexibility – and freedom from fixed-layer schemas – in return for minimal effort.

There are two types of fanout: Feature Type Fanout and Dataset Fanout.
**Feature Type Fanout**

A Feature Type Fanout delivers data to multiple feature types (layers) within a single dataset. Taking the elevation example, here the output is a different feature type for each elevation value:

This results in a DXF dataset containing multiple layers of data.

**Setting a Feature Type Fanout**

A feature type fanout is defined in the Feature Type Properties dialog by selecting an attribute like so:

The Feature Type Name then changes to match what has been selected:
In this case, each feature with a different park name will be written to a different layer of the GML output dataset.

Additionally, the Text Editor can be used to construct a fanout string:

The result of this translation - as shown in the FME Data Inspector - is a series of layers, each with the park ID and name included:

Two things to notice in that result are:

- Parks 1 and 4 did not have a ParkName attribute (or it was empty) and so the layer name reflects that.
- The : (colon) and - (dash) characters in the fanout string were rendered as _ (underscore) characters instead. Presumably this is a limitation of the GML output format.
Sister Intuitive says…

FME2016 (in fact, FME2015.1) introduced the above technique as an easier way for setting a feature type fanout, losing the "fanout" label at the same time. But, regardless of the label, it's still the same fanout as in previous versions.
Dataset Fanout

A Dataset Fanout delivers data to the same feature type, but in multiple datasets. Taking the elevation example again, here the output is a different dataset for each elevation value:

This results in a series of DXF datasets, each of which has one elevation’s worth of contours on one layer.

Setting a Dataset Fanout

A Dataset Fanout is defined in the Navigator window in Workbench, just below the Writer’s dataset parameter:

Double-clicking the Fanout Dataset parameter opens a dialog in which to define the folder to write to and the Fanout Expression to use:

The Fanout Expression can - like the Feature Type fanout - be a simple attribute, like so:
In this case, each feature with a different park name will be written to a different GML output dataset (whereas a feature type fanout would write to different layers in the same dataset).

Additionally, the Text Editor can be used to construct a fanout string:

```
The result of this translation is a series of GML datasets, each with the park ID and name included:
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Date modified</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Park1.gml</td>
<td>12/01/2016 3:36 PM</td>
<td>GML File</td>
<td>2 KB</td>
</tr>
<tr>
<td>Park1.xsd</td>
<td>12/01/2016 3:36 PM</td>
<td>XSD File</td>
<td>2 KB</td>
</tr>
<tr>
<td>Park2-Rosemary Brown Park.gml</td>
<td>12/01/2016 3:36 PM</td>
<td>GML File</td>
<td>2 KB</td>
</tr>
<tr>
<td>Park2-Rosemary Brown Park.xsd</td>
<td>12/01/2016 3:36 PM</td>
<td>XSD File</td>
<td>2 KB</td>
</tr>
<tr>
<td>Park3-Tee Swamp Park.gml</td>
<td>12/01/2016 3:36 PM</td>
<td>GML File</td>
<td>3 KB</td>
</tr>
<tr>
<td>Park3-Tee Swamp Park.xsd</td>
<td>12/01/2016 3:36 PM</td>
<td>XSD File</td>
<td>2 KB</td>
</tr>
<tr>
<td>Park4.gml</td>
<td>12/01/2016 3:36 PM</td>
<td>GML File</td>
<td>2 KB</td>
</tr>
<tr>
<td>Park4.xsd</td>
<td>12/01/2016 3:36 PM</td>
<td>XSD File</td>
<td>2 KB</td>
</tr>
<tr>
<td>Park5-Morton Park.gml</td>
<td>12/01/2016 3:36 PM</td>
<td>GML File</td>
<td>2 KB</td>
</tr>
<tr>
<td>Park5-Morton Park.xsd</td>
<td>12/01/2016 3:36 PM</td>
<td>XSD File</td>
<td>2 KB</td>
</tr>
<tr>
<td>Park6-McBride Park.gml</td>
<td>12/01/2016 3:36 PM</td>
<td>GML File</td>
<td>2 KB</td>
</tr>
<tr>
<td>Park6-McBride Park.xsd</td>
<td>12/01/2016 3:36 PM</td>
<td>XSD File</td>
<td>2 KB</td>
</tr>
<tr>
<td>Park7-Granville Park.gml</td>
<td>12/01/2016 3:36 PM</td>
<td>GML File</td>
<td>2 KB</td>
</tr>
<tr>
<td>Park7-Granville Park.xsd</td>
<td>12/01/2016 3:36 PM</td>
<td>XSD File</td>
<td>2 KB</td>
</tr>
</tbody>
</table>

Two things to notice in that result are:

- The : (colon) character is not supported in the filename (a Windows limitation) - but this time it had to be removed before running the workspace
- The file extension (.gml) is necessary as part of the fanout string. FME won't add it automatically.
**Miss Vector says...**

*Fanouts are an important part of writing data with FME, so tell me, which of these statements are true?*

1. You can have both a Feature Type Fanout and a Dataset Fanout in the same workspace
2. You can use a Feature Type Fanout with a database format, but not a Dataset Fanout
3. A fanout expression can be an attribute, or a constructed string, but not a user parameter
4. A fanout cannot be based on a format attribute such as fme_color
Exercise 1  Fanning Out Development Zones

<table>
<thead>
<tr>
<th>Data</th>
<th>Zoning Data (MapInfo TAB, Esri Shapefile)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>Create a separate Shape dataset for each type of development zone</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>Feature Type Fanouts and Zipped Datasets</td>
</tr>
<tr>
<td>Start Workspace</td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\ReadWrite-Ex1-Complete.fmw</td>
</tr>
<tr>
<td>End Workspace</td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\ReadWrite-Ex1-Complete-Advanced.fmw</td>
</tr>
</tbody>
</table>

You’ve been given a dataset of development zones and asked to separate each zone type into a separate Shapefile and send it back with everything zipped together in a single file.

The requester thinks this will be a difficult task; but with FME you should be able to do it in about two minutes.

1) Inspect Source Data
Inspect the source dataset for this translation in the Data Inspector. The source data is a MapInfo TAB dataset:

<table>
<thead>
<tr>
<th>Reader Format</th>
<th>MapInfo TAB (MITAB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reader Dataset</td>
<td>C:\FMEData2016\Data\Zoning\Zones.tab</td>
</tr>
</tbody>
</table>

Check the geometry type used and notice that there is a field called ZoneName. We need the first characters of this field (up to any "–" character) for our fanout.

2) Start Workbench
Start Workbench and generate a workspace to translate the MapInfo source data to Esri Shapefile.

By default the workspace will include a GeometryFilter and multiple output feature types. However, we know the data is polygon only (because we inspected it first, right?) so we can remove much of this.

So, delete the GeometryFilter transformer and all of the Writer feature types except Zones_polygon. You’ll end up with something that looks like this:

3) Add StringReplacer Transformer
To remove everything after the "–" character in the ZoneName field, place a StringReplacer transformer into the workspace, between the Reader and Writer feature types.

4) Set Parameters
Open the parameters dialog for the StringReplacer. Set the following parameters:

- **Attributes**: ZoneName
- **Text to Match**: -(.+)$
- **Replacement Text field**: leave empty
- **Use Regular Expressions**: Yes (this is very important)
This regular expression will search out the dash character – and anything after it – and replace it with nothing (i.e. delete it). Click OK to close the dialog.

5) Set Fanout
Open the Feature Type Properties dialog for the Writer feature type. Click the drop-down arrow next to the Shapefile name parameter and select Attribute Value > ZoneName:

Click OK to close the dialog.

6) Save and Run Workspace
Save the workspace. Run the workspace with the Prompt option turned on. When prompted manually change the Destination Directory to: C:\FMEData2016\Output\Zones.zip

**NB:** Manually enter the name directly into the field. Don’t click the browse button first. You cannot enter a filename in the browse dialog.

Locate the output folder in a file browser. You should see the file Zones.zip:
If you open it up there will be inside a Shapefile dataset for every zone type.

**Sister Intuitive says…**

A feature type fanout results in multiple Shapefile datasets because each Shapefile is a layer (feature type). As an advanced task, repeat the exercise but write a separate DWG file for each zone type. In that scenario you’ll need to use a Dataset Fanout instead. Be sure to add ".dwg" as a suffix to the fanout.

**CONGRATULATIONS**

By completing this exercise you have learned how to:
- Use a feature type fanout to write separate feature types
- Use a .zip extension to create zipped output datasets
- (Advanced) Use a dataset fanout to write separate datasets
The Generic Reader/Writer

The Generic Reader and Writer allow FME workspaces to be freed from format restrictions.

Where all other Readers and Writers are tied to a specific format of data, the Generic Reader and Generic Writer are not. The Generic Reader is capable of reading almost any format of data, and the Generic Writer is capable of writing almost any format of data.

In that way, a single workspace can be used to process different data formats without being specifically set up for that format.
The Generic Reader

A Generic Reader is used in the same way as any other Reader; by specifying the format in the new Reader dialog:

Here the workspace is being created by reading a MapInfo TAB dataset, but it could then be set to GML when run, like so:

FME will examine the extension of the file chosen to identify the format of data and then read it just as if it were a true GML Reader.

Sister Intuitive says…

You're thinking there must be a catch, right? Well it's true. Firstly this technique only works with file-based formats (it won't work on a database or web format). Secondly, the Generic Reader is not immune from the Unexpected Input Remover, so switching datasets - regardless of format - only works with a compatible schema.

Generic Reader Feature Types

The Unexpected Input Remover is the function in FME that filters incoming data against the list of feature types (layers) that are defined in the workspace. If the incoming data is stored on a layer that is not defined in the workspace, then it will be dropped from the translation:
So, although the Generic Reader allows you to read datasets of different formats, the limitation is that each dataset must have its layers defined as feature types in the workspace, unless you want them to be dropped.

Of course, an easy way to allow all layers to pass is to set a Merge Feature Type in the Feature Type Properties dialog:

```plaintext
Merge Feature Type
Merge Filter: =
Filter Type: Wildcard

<All>
```

With that setup, any layer of data can be passed into the workspace, regardless of format. But even then you need to be careful about assuming what attributes will be available!

---

**Generic Reader Parameters**

All Readers in a workspace have a number of parameters that can be used to control how that Reader operates. Each format has its own set of specialized parameters.

However, the Generic Reader has very few parameters, the main one being to set the format of data being read:

```plaintext
By default this is set to determine data format from the file extension, but it can be set manually to a specific format.
```

---

**Sister Intuitive says…**

*The Input Format parameter is useful when the file extension is - like .mdb - one used by multiple formats. For example, you might turn it into a user parameter for use in an FME Server data upload service. Then a single workspace can read any format of data, but the end-user is able to tell you what format that is.*

---

However, let's say you wish to use the Generic Reader, but apply a particular GML Reader parameter when a GML dataset is being read. If the Generic Reader has no parameters, then how can this be done?

In brief, the solution is to add a dummy GML Reader:
Here, for example, the author has added a GML Reader but then deleted all its feature types. The data will be read and then dropped. But, all of the parameters in this Reader will get applied to the Generic Reader when reading GML data.

Sister Intuitive says…

*In fact, the little-known Resource Reader is the best solution, because it applies parameters without reading any data. You can add a Resource Reader using Readers > Add Reader as Resource. There’s more info on Resource Readers later in this chapter.*
The Generic Writer

A Generic Writer is used in the same way as any other Writer; by specifying the format in the new Reader dialog:

Unlike the Generic Reader, the Generic Writer cannot determine the format to write by a file extension, so it has to use a Format Parameter to specify the output format:

Whatever format is chosen will be the format the data is written in. The same parameter is available in the Navigator window once the workspace is created.

This parameter is one of those that FME automatically creates a linked user parameter for. That way the end-user can choose at runtime which format to write to.

The Destination Dataset parameter, like all dataset parameters, is also linked to a user parameter. Note that the destination for this writer is always a folder, even when the selected format is file-based.

Sister Intuitive says…

It’s important to remember that FME sometimes transforms output data to fit the definitions and rules of the destination format. Therefore, the same Generic Writer may produce slightly different results for different data formats.

Generic Writer Feature Types

Feature Types are less of an issue for the Generic Writer (than the Generic Reader) because they are already pre-defined in the workspace.
The main limitation will be if you wish to have both a Generic Reader and Generic Writer, with the Reader accepting any layers and the Writer writing ones to match.

In that scenario, you could use a feature type fanout with the Generic Writer, based on the format attribute fme_feature_type. Then the destination dataset will have the same layers as the source – even if that varies from translation to translation!

**Generic Writer Parameters**

Like the Generic Reader, the Generic Writer has only one or two parameters of its own (output format being one of them). To apply a particular format parameter you need to add a dummy Writer of the same format.

The dummy Writer does not need to have any feature types defined, or any data sent to it; in fact it should not as this would only slow the translation.

Here, for example, the author has added a dummy GML Writer (with no feature types) in order to use the parameters for GML writing.

**Sister Intuitive says…**

Generic Readers and Writers by nature only deal with a flexible format, but can also be set up to be flexible with layers.

However, each dataset being read must have the same attribute schema, and each dataset being written will end up with the same attribute schema. This part is not flexible.

Flexible attribute schemas require the use of either Automatic Attribute Definitions or a Dynamic Translation.

**Miss Vector says…**
Now tell me which of these statements about Generic Reading/Writing are true?

1. Because you select an output **folder**, the Generic Writer won't write to a **file** format like AutoCAD DWG
2. If the feature types of the chosen format are limited to a single geometry, the Generic Writer will drop all features except a single geometry type
3. The Generic Writer does not support either type of Fanout
4. The ParameterFetcher transformer can be used to retrieve the format of data being Read/Written in order to route features in a specific way through the workspace
Exercise 2  Community Mapping Data Translation Project

<table>
<thead>
<tr>
<th>Data</th>
<th>Community Mapping (Esri File Geodatabase)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>Create a workspace to translate Community Mapping data to a format of the end-user's choice</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>Generic Writer</td>
</tr>
<tr>
<td>Start Workspace</td>
<td>None</td>
</tr>
<tr>
<td>End Workspace</td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\ReadWrite-Ex2-Complete.fmw</td>
</tr>
</tbody>
</table>

As resident FME expert you are often asked to translate data (particularly the community map) between formats. You realize that it would be way simpler if you created a workspace to do this - regardless of format - and let the end-users carry out the translation themselves. In the future this would make an excellent use for an FME Server Data Download service, but for now we'll let the users simply run the workspace in FME Workbench.

1) Start Workbench
Start FME Workbench and begin with an empty canvas. Select Readers > Add Reader from the menubar and add the following:

| Reader Format | Esri Geodatabase (File Geodb API) |
| Reader Dataset | C:\FMEData2016\Data\CommunityMapping\CommunityMap.gdb |
| Workflow Options | Single Merged Feature Type |

By selecting the single merged feature type option we will have a workspace that is nice and compact, plus it will allow the user to select which tables they want to read from the source.

Click OK to close the dialog and add the Reader.

2) Add Writer
Select Writers > Add Writer from the menubar and add a Generic Writer:

| Writer Format | Generic (Any Format) |
| Writer Dataset | |
| Writer Parameters | Output Format: Esri Shapefile |
| Add Feature Types | Feature Type Definition: Automatic |

You don’t have to select an output location, but will you have to open the parameters dialog and set an original output format; so do that and select a format like Esri Shape:

In the "Add Feature Types" section of the dialog, select Automatic for feature type definitions.
The Feature Type Properties dialog for the new Writer will open automatically. Set the Allowed Geometries field to fme_any. This will allow any data to be written to this feature type.

Click OK to close the dialog and add the new feature type. Connect it to the source feature type. When you make the connection the attribute schema will automatically be updated to match the connected Reader feature type:

3) Check User Parameters

Look in the Navigator window at the user parameters that were created automatically with the Reader and Writer:
The parameter for source dataset is something we won’t ever need (this translation will always use the same dataset) so delete it.

Another automatically created parameter is called Feature Types to Read. This is very useful because when the user runs the workspace they will be prompted to select which tables to read from the source Geodatabase, so keep this parameter.

Similarly keep the Destination Dataset parameter.

The Output Format parameter is interesting. Double-click on it as if you were going to set a value. Notice that the "More Formats..." option in the drop-down list opens up the full FME formats list.

It wouldn’t be fair to the end-user to expose so many formats, when they don’t really need to see or select most of them. It would be better to restrict this list. So, delete this user parameter and we’ll create a new - more restrictive - one.

4) Add User Parameter

Add a new User Parameter by right-clicking on User Parameters and selecting Add Parameter. In the dialog that opens set the following:

<table>
<thead>
<tr>
<th>Type</th>
<th>Choice with Alias</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>OutputFormat</td>
</tr>
<tr>
<td>Published</td>
<td>Yes (checked)</td>
</tr>
<tr>
<td>Optional</td>
<td>No (unchecked)</td>
</tr>
<tr>
<td>Prompt</td>
<td>Select Output Format</td>
</tr>
</tbody>
</table>

For the configuration field click the browse button to open a new dialog. In that dialog, select Import > Writer Formats. Select a handful of the most common formats like Shape, AutoCAD DWG, GML, and MapInfo TAB; then click OK.

Then click OK twice more until all the dialogs are closed.

5) Link Parameter

Now, in the Navigator window, expand the parameters for the Generic Writer. Locate the Output Format parameter. Right-click it and choose Link to User Parameter.

Select the newly created OutputFormat parameter and click OK. Now when the workspace is run, the choice of output format will be between these few:

6) Expose Format Attribute

The other, final, task we can do here is to output the features to their original table. To do this we need to know where they came from, and that is obtained from a format attribute called fme_feature_type.

Open the properties dialog for the Reader feature type and click the Format Attributes tab.
Put a checkmark next to fme_feature_type and click OK to close the dialog:

![Feature Type Properties](image)

**.1 UPDATE**

In FME2016.1 you can skip the above step: fme_feature_type is automatically exposed whenever the source feature type has a merge filter set.

7) Set Feature Type Fanout
Now open the properties dialog for the Writer feature type. Set a fanout by choosing fme_feature_type as the attribute supplying the feature type name.

![General Parameters](image)

8) Save and Run Workspace
Save the workspace and then run it with the prompt option set. When prompted, select some source tables to read (include at least the GarbageSchedule plus one other) and set an output folder. Set Esri Shapefile as the format to write.

Examine the output folder. The selected tables have been written back to Shapefile format:
Now you have a solution that almost anyone will be able to open and run for themselves. Also, if you published the workspace to FME Server it would automatically create a web page that matched the different user parameters.

Sister Intuitive says…

Did you notice that FME handled the different geometry types and output the files with the geometry as part of the name? It’s a Shape format thing. FME can never – and will never – write more than one geometry type to the same Shape file.

The one drawback is that each output Shape file has **all** of the attributes of **all** of the source tables. To avoid that you would need to use a dynamic translation, as we shall see shortly.

CONGRATULATIONS

By completing this exercise you have learned how to:
- Add a Generic Writer to a workspace
- Set the initial format of a Generic Writer
- Create a Choice with Alias parameter that prompts for a Writer format
- Use a Choice with Alias parameter for the Generic Writer output format
- Use a feature type fanout on fme_feature_type to return features to their original layer
Dynamic Translations

Dynamic Translations are a way to create “schema-less” workspaces.

What are Dynamic Translations?

Most translations - and everything this training has covered so far - involve a schema being defined within the workspace. In other words, the source and destination schema reflect the structure of the source data (what we have) and the structure of the destination data the user requires (what we want).

The layout of a dynamic translation does not reflect either the source or destination schema. It’s a universal layout that is designed to handle data regardless of what schema is being used.

Sister Intuitive says…

For this section it’s useful to think of a schema being comprised of a trinity of objects: feature types, attributes, and geometry type.

Dynamic Readers

On the Reader side of things, a dynamic workspace is very similar to using Merge Parameters; feature types are given free entry to a workspace, regardless of whether they are yet defined in there.

Data is also read regardless of attributes and/or geometry type.

Dynamic Writers

The Writer side of a dynamic workspace mimics the Reader part; feature types are written to the destination dataset, regardless of whether they have been defined in the workspace.

Additionally, all attributes and geometries are also written, regardless of whether they too have been predefined in a Writer feature type.
Miss Vector says…

It’s important to get the concepts correct here, and I have some more statements only some of which are true:

1. A Dynamic workspace will read/write any format of data
2. A Dynamic workspace will read/write any feature types in the source data
3. A Dynamic workspace will read/write any attributes in the source data
4. A Dynamic workspace will read/write any geometry in the source data
Creating a Dynamic Translation

When an author creates a translation using the Generate Workspace dialog, there are two options for what is called workflow: static schema and dynamic schema.

The Static Schema option is the default for a workspace including schema. Choosing the Dynamic Schema option creates a schema-less workspace with dynamic Readers and Writers.

It is, however, possible to also create a workspace where only the Readers are dynamic, or only the Writers.

Dynamic Reader Only

The Add Reader dialog has similar options for static and dynamic; however in this case we try to make them more user-friendly by labelling them Individual Feature Types and Single Merged Feature Type:

In essence, a dynamic Reader is similar to just setting the Merge Feature Type option.

Dynamic Writer Only

The Add Writer dialog has options for how feature types and their attributes will be defined. The most commonly used ones are Manual and Automatic. There is also an option that will add a Writer in dynamic mode:
Sister Intuitive says…

Let’s clarify Automatic vs Dynamic. Automatic attributes take their definition from whatever is connected to them. If the Source Dataset parameter is changed, it will have no effect.

Dynamic attributes are different. If the Source Dataset parameter is changed, the attribute definition comes from whatever source data gets read, regardless of what is connected to it.

How Does a Dynamic Translation Look?

Both dynamic readers and dynamic writers each have a single Feature Type, regardless of the schema of the Reader datasets:

Notice that there is only a single feature type, regardless of whether the data is made up of several layers or tables.

Also notice that the sole Reader Feature Type is named <All> (which provides a clue to what is happening here) and that the sole Writer Feature Type is named <Dynamic>.

When the workspace is run, all of the source data is read through a single feature type. On the Writer side, although there is only one output type, the data will be dynamically divided back into its component layers, keeping its original attributes and geometry type.

With this workspace you can switch the source dataset to anything (of the correct format) and the output will be a mirror image. There is no need to worry about unexpected input or unsupported geometry types. Plus, if you used the Generic Reader/Writer, it could read any dataset, of any format and create a duplicate output of it!
Exercise: Dynamic Community Map Translation

<table>
<thead>
<tr>
<th>Exercise 3</th>
<th>Dynamic Community Mapping Data Translations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Community Mapping (Esri File Geodatabase)</td>
</tr>
<tr>
<td></td>
<td>Addresses (Esri File Geodatabase)</td>
</tr>
<tr>
<td>Overall Goal</td>
<td>Create a dynamic workspace to translate any Geodatabase dataset to a format of the end-user's choice</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>Dynamic Formats</td>
</tr>
<tr>
<td>Start Workspace</td>
<td>None</td>
</tr>
<tr>
<td>End Workspace</td>
<td>C:/FMEData2016/Workspaces/DesktopAdvanced/ReadWrite-Ex3-Complete.fmw</td>
</tr>
</tbody>
</table>

In the previous exercise, a workspace was generated to translate a Geodatabase dataset into a number of formats using the Generic Writer.

However, that workspace had a limitation with the output attributes (every output dataset got all of the source table attributes) and you also feel it would be useful if that workspace could handle any source Geodatabase, not just the community maps dataset.

So, let’s create a new workspace to handle that scenario.

1) Start Workbench
Start FME Workbench and begin by generating a workspace as follows:

<table>
<thead>
<tr>
<th>Reader Format</th>
<th>Esri Geodatabase (File Geodb API)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reader Dataset</td>
<td>C:/FMEData2016/Data/CommunityMapping/CommunityMap.gdb</td>
</tr>
<tr>
<td>Writer Format</td>
<td>Generic (Any Format)</td>
</tr>
<tr>
<td>Writer Dataset</td>
<td>C:/FMEData2016/Output/Training</td>
</tr>
<tr>
<td>Workflow Options</td>
<td>Dynamic Schema</td>
</tr>
</tbody>
</table>

2) Inspect Workspace
Inspect the newly created workspace:

There is one Reader feature type and one Writer feature type. The Reader feature type shows a list of attributes, but the Writer feature type doesn’t. It is, however, labelled Dynamic.

Again, there will be a user parameter for the Feature Types to Read and the output format.

If you wish, create a more-limited version of the output format parameter, by following steps 3-5 in the previous exercise; although this isn’t totally necessary for what we’re doing here.

But don’t delete the Source Dataset user parameter; we’ll need that shortly.
3) Run Workspace
Run the workspace with the prompt option set.

When prompted, select some source tables and set the output format. The workspace will run to completion. Check the output to ensure it is all correct.

4) Re-Run Workspace
Now run the workspace again.

This time click the browse button for the source Geodatabase and browse to C:\FMEData2016\Data\Addresses\Addresses.gdb

Choose the feature types to read and this time you will be presented with a list of feature types from the newly selected Geodatabase.

Click OK to run the workspace again. Inspect the output. Notice that the output feature types are all as listed in the original data. Also notice that the attributes are as in the original too!

From this we can see that a dynamic workspace is capable of handling any source schema and writing it out to a new dataset just as it was in the source data.

CONGRATULATIONS

By completing this exercise you have learned how to:
- Create a dynamic workspace
- Use the Generic Reader in a dynamic workspace
Schema Handling in Dynamic Translations

Opening feature type properties dialogs for a dynamic translation reveals the checkboxes that turn on this behavior.

For a Reader, all that is really happening is the merge feature type setting is turned on:

- **Merge Feature Type**
  - Merge Filter: *
  - Filter Type: Wildcard
  - Case-sensitive Filter

Unchecking that box turns off the full behavior and there are not many parameters to adjust. However, for a Writer, the dialog is a bit more complex:

- **General Parameters**
  - Shapefile Name: fme_feature_type
  - Table Qualifier:
  - Writer: Output [ESRIShape]
  - Geometry: From Schema Definition

- **Dynamic Properties**
  - Schema Definition
    - Schema Sources: "Zones [MITAB]"
    - Schema Definition Name: Default from Shapefile name above
    - Attributes to Remove:

The three components of schema - 1) feature type, 2) geometry, and 3) attributes - all have different ways in which they can be set.

---

**Sister Intuitive says…**

*By default* the Writer schema in a dynamic translation is defined *not* in the workspace, but by the source dataset. So whatever dataset is chosen as input defines the chosen output structure. Simple.

However, the parameters in a Writer feature type let us alter how that schema is defined. We can choose to take the structure from an entirely different dataset to the source. Or, we can individually define each component of our schema (Feature Types, Attributes, Geometry) in a variety of ways.

So, we'll start out by looking at how to use a schema from a different dataset, then we'll look at each schema component separately.
Schema Sources

The Writer feature type has a dynamic parameter labelled Schema Sources:

This parameter defines where the destination schema is going to be obtained from. By default, this parameter is set to whatever source dataset is being read. That way the output schema is always a duplicate of the input.

However, it can be set to use any Reader dataset – in any format – as the source for the outgoing schema.

For example, here the workspace author is converting zoning data, but has chosen a PostGIS database as the required structure for the output dataset.

Presumably, the corporate database includes a zoning table, and the structure of that table will be used as the schema of the output in this workspace.

Sister Intuitive says...

In this scenario the “Dynamic” part of the translation is the destination schema being fetched at run time. For example, if the corporate schema (above) were to change, the workspace feature types would not need updating. They would be automatically ‘updated’ when the workspace is run.

Resource Readers

The Schema Source parameter can be set to point to any Reader as the source of a dynamic schema. However, in most cases all we need from the dataset is the schema, not the data.

This is where Resource Readers can be used.
A Resource Reader is a Reader that returns the schema of a dataset, but no data. One is added using Readers > Add Reader as Resource on the menubar:

Here the user adds a PostGIS database as a resource and it appears in the Navigator window:

Once available the resource Reader can be used as the source of a schema for a dynamic Writer.

Non-Matching Data

Sadly, the ability to pick any schema for the Writer doesn't mean it will handle just any type of data. The incoming data must have the same feature types as defined in the Writer schema, otherwise they will be dropped.

When features are dropped, the FME log reports that fact like this:

<table>
<thead>
<tr>
<th>Features With No Schema defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zones</td>
</tr>
<tr>
<td>Total Features NOT Written</td>
</tr>
</tbody>
</table>

Consider this behavior a sort of “Unexpected Output Remover”. In this case, data was read from a feature type called Zones. However, no table called Zones is defined in the PostGIS database used as the Writer schema. Therefore the Zones data was dropped from the translation.
Why Use an External Schema?

The main reason for using an external dataset schema is to adhere to a fixed standard. Perhaps the most useful aspect of this is that if the schema of the dataset changes, then the workspace makes use of it automatically. There's no need to manually update the workspace because the output requirements have changed.

However, as noted above, the data being written must match that standard or risk being dropped. So there is always likely to be some data transformation required in the workspace to coax the input data into the required output schema.

The SchemaMapper transformer is useful for reconciling data with the required schema - that's because the SchemaMapper too can use an external lookup table, meaning that the dynamic workspace can be changed to meet any required output schema without having to actually make edits in Workbench!
Dynamic Feature Types

In all Writer feature types, the feature type name can be defined by an attribute (or string constructed from attributes). When a Writer is set up as dynamic, by default the attribute used is an FME attribute called fme_feature_type:

![General Parameters](image)

fme_feature_type stores the name of the original feature type on incoming features. It makes sense to default to this attribute because then all data is written to the same feature type as it came from, and we get an output that is a duplicate of the input.

However, should you wish, there's no reason why a different attribute couldn't be used, in order to define a different set of output feature types:

![General Parameters](image)

Here, for example, the author is using ZoneCategory to supply the name of the layers to create.

Schema Requirements

However! Changing the output feature type names is not as simple as that. Because this is a dynamic workspace (not just a fanout) the feature type names chosen must match a layer that exists in the source schema dataset.

Failure to do so will lead to the data being dropped with the following log messages:

```
Features With No Schema defined
---------------------------------------------
Commercial                                  51
Comprehensive Development                  237
Historic Area                               5
Industrial                                  14
Light Industrial                            19
Multiple Family Dwelling                    45
One Family Dwelling                         19
Two Family Dwelling                         26

Total Features NOT Written                  416
```

---

246
What the author must ensure is that the schema used actually contains these layers. Then the translation will proceed as expected.
Dynamic Attributes

The Dynamic Properties part of the feature type dialog includes a setting for Schema Source and underneath that, a setting for Schema Definition Name:

The Schema Sources setting we know can point to a different dataset, one that defines a schema we wish to use. The Schema Definition Name defines which feature type to use in that dataset to define our attributes.

By default the feature type name defined in the General Parameters is used to define both the layers to be created and the attributes to be used for those layers.

However, the Schema Definition Name can be set to define attributes from different feature types than those that are being used to define output layers.

Because it is possible to set this parameter using an attribute, each different feature can point to a different attribute definition.

Here, for example, the user has set up a custom set of feature types to be written by adding a new schema source and setting the output feature type name to point to feature types in that dataset.

However, the output will NOT contain the attributes from that dataset. Instead the user is choosing to create attributes from another Reader feature type, called Parks.

Sister Intuitive says…

Yes, it's very complex and sometimes hard to grasp what all this means.
Adding or Deleting Attributes

Besides specifying which set of attributes to use, sometimes - even in a dynamic translation - you need to add or delete specific attributes. This is very simple to do.

Adding a New Attribute

Adding a new attribute to all output on a dynamic feature type is just a case of editing the feature type definition to add that attribute.

In other words, any attribute you add to the feature type definition will get added to all features output through there – regardless of source or resource schemas.

For example, I might add an attribute if an AreaCalculator transformer was in the workspace and I wished to store the result.

Deleting an Attribute

Deleting an existing attribute is done through the dynamic Schema Definition dialog. At the foot of that dialog is a field for removing attributes:

The edit [...] button opens a dialog in which to select attributes that are in the source schema but that you don’t want in the output:
You can select existing attributes to remove or you can manually enter attribute names. That way you can choose to remove attributes even when the schema is coming from outside of the workspace.
Dynamic Geometry

The geometry used in an output dataset depends on the format used. Only when the format does not allow a mix of geometries in a single feature type can there be a problem.

For example, a standard (static) Shape Writer feature type allows you to pick the geometry allowed in that file:

In a dynamic workspace this changes. The geometry type permitted depends on that defined in the chosen schema:

By default - when the Schema Source is unchanged as that being read by the Reader - then the permitted geometry will be a duplicate of the source data.

When the Schema Source is changed to another dataset, then the permitted geometry is what is defined in that dataset. If the geometry of the source dataset is different, then it would be dropped instead of written.

Additionally, the workspace author may edit the schema definition for permitted geometry:

A specific geometry type will set the permitted geometry to that type. From Schema Definition, as we know, sets the permitted geometry to whatever is defined in the Source Schema.

The other option is First Feature Defines Geometry Type.
**First Feature Defines Geometry Type**

The geometry type in a dynamic translation can be difficult to handle once the feature types are being handled dynamically (i.e. either the Schema Source has changed or the Feature Type name is set to other than fme_feature_type).

That's because two features with the same Feature Type name might have different geometry types. In that situation FME can't tell which should get priority.

If First Feature Defines Geometry Type is the chosen method of handling geometry, then the first feature to arrive at the Writer gets to set the geometry type. Subsequent features destined for the same feature type will be refused if they do not have the correct geometry.
Exercise 4: Dynamic Community Mapping Updates

<table>
<thead>
<tr>
<th>Data</th>
<th>Community Mapping (Esri File Geodatabase)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>Create a workspace to generate a new Community Mapping dataset using an existing schema</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>Dynamic Schemas</td>
</tr>
<tr>
<td>Start Workspace</td>
<td>None</td>
</tr>
<tr>
<td>End Workspace</td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\ReadWrite-Ex4-Complete.fmw</td>
</tr>
</tbody>
</table>

Previous exercises have involved translating the Community Mapping dataset, used by the planning department for various community mapping tasks.

However, as time has moved on the Community Mapping dataset has become out of date. The planning department therefore wants to start building a new community map dataset. The dataset will have new data, but use the existing schema where possible. They also – in order to use an open standard – want a format change to GML.

At the moment two source datasets have been identified as being required in the new community mapping "database". They are fire halls (source format: GML) and city parks (source format: MapInfo TAB).

So, let's create a new workspace to handle that scenario.

1) Inspect Data
Inspect the two source datasets in the FME Data Inspector, to become familiar with them.

<table>
<thead>
<tr>
<th>Format</th>
<th>GML (Geography Markup Language)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset</td>
<td>C:\FMEData2016\Data\Emergency\FireHalls.gml</td>
</tr>
<tr>
<td>Format</td>
<td>MapInfo TAB (MITAB)</td>
</tr>
<tr>
<td>Dataset</td>
<td>C:\FMEData2016\Data\Parks\Parks.tab</td>
</tr>
</tbody>
</table>

There was already parks data in the community mapping, but this time it is polygons, not points. The FireHalls data is entirely new for community mapping.

2) Start Workbench
Let's get started by generating a workspace as follows:

| Reader Format | GML (Geography Markup Language) |
| Reader Dataset | C:\FMEData2016\Data\Emergency\FireHalls.gml |
| Writer Format | GML (Geography Markup Language) |
| Writer Dataset | C:\FMEData2016\Output\Training\NewCommunityMap.gml |
| Workflow Options | Dynamic Schema |

The dynamic option is chosen so that we can use a schema other than that of the dataset being translated.

3) Add Reader
So far, so good. Now let's add a Reader for the new parks data.

Select Readers > Add Reader from the menu bar. Add a Reader as follows:
4) Add Resource Reader
One of the requirements was to use the existing community mapping schema where possible. With the firehalls it isn’t possible, since that never existed in the Community Mapping Geodatabase. However, the parks dataset did exist in that Geodatabase, so we’ll need to use that schema.

So, select Readers > Add Reader as Resource.

When prompted, enter the following info:

<table>
<thead>
<tr>
<th>Reader Format</th>
<th>Esri Geodatabase (File Geodb API)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reader Dataset</td>
<td>C:\FMEData2016\Data\CommunityMapping\CommunityMap.gdb</td>
</tr>
</tbody>
</table>

Click OK and the Reader is added as a Resource:

5) Adjust Dynamic Parameters
Now we need to make sure that resource is being used.

Open the properties dialog for the Writer feature type. Click the Schema Sources browse button.

Put a checkmark against CommunityMap and ensure Parks is NOT checked.
Click OK.

Since we are writing both points and polygons, for some formats we might have to change the Geometry setting. But GML can cope with both geometry types and so this section is greyed-out.

Click OK to close the Feature Type Properties dialog.

6) Save and Run Workspace

Save the workspace and then run it.

Inspect the output. There should be two layers: one for fire halls, the other for parks. The parks dataset should have the schema from the Geodatabase (not the MapInfo parks), including attributes for ParkName, ParkAddress, and ParkURL (even if there is no data to fill some fields yet):

<table>
<thead>
<tr>
<th>ParkName</th>
<th>ParkAddress</th>
<th>ParkURL</th>
<th>OBJECTID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanier Park</td>
<td>&lt;missing&gt;</td>
<td>&lt;missing&gt;</td>
<td>&lt;missing&gt;</td>
</tr>
<tr>
<td>Sutcliffe Park</td>
<td>&lt;missing&gt;</td>
<td>&lt;missing&gt;</td>
<td>&lt;missing&gt;</td>
</tr>
<tr>
<td>Charleson Park</td>
<td>&lt;missing&gt;</td>
<td>&lt;missing&gt;</td>
<td>&lt;missing&gt;</td>
</tr>
<tr>
<td>Granville Loop Park</td>
<td>&lt;missing&gt;</td>
<td>&lt;missing&gt;</td>
<td>&lt;missing&gt;</td>
</tr>
<tr>
<td>Willow Park</td>
<td>&lt;missing&gt;</td>
<td>&lt;missing&gt;</td>
<td>&lt;missing&gt;</td>
</tr>
<tr>
<td>Coopers’ Park</td>
<td>&lt;missing&gt;</td>
<td>&lt;missing&gt;</td>
<td>&lt;missing&gt;</td>
</tr>
<tr>
<td>Helmcken Park</td>
<td>&lt;missing&gt;</td>
<td>&lt;missing&gt;</td>
<td>&lt;missing&gt;</td>
</tr>
<tr>
<td>Roundhouse Turntable Plaza</td>
<td>&lt;missing&gt;</td>
<td>&lt;missing&gt;</td>
<td>&lt;missing&gt;</td>
</tr>
</tbody>
</table>

Notice that it also has OBJECTID, which came from the Geodatabase and we don’t really need.

7) Delete Attribute

Re-open the properties dialog for the Writer feature type. Under Attributes to Remove type OBJECTID into the first row. You
won’t be able to select it from the drop-down list because it comes from a resource reader, not a real reader:

But don’t close the Feature Type Properties dialog yet.

8) Add Attribute
A last-minute request is to add an attribute – LastUpdatedBy – to all tables in the output.

So, click on the User Attributes tab and add this new attribute. Make it a 30 character string.

As you can see, there is no need to change the attribute definition mode - it should stay as dynamic.

9) Re-Run Workspace
Save the workspace and run it again.

Inspect the output. Notice that OBJECTID will not appear as an attribute. LastUpdatedBy does appear, albeit that it doesn’t have a value yet.

CONGRATULATIONS

By completing this exercise you have learned how to:
- Create a dynamic workspace with multiple Readers
- Add a Resource Reader
- Change the source of a dynamic workspace schema
- Add and remove hard-coded attributes in a dynamic workspace
Exercise: Creating a new Community Map Dataset
Advanced Dynamic Attribute Schemas

In general, the schema for a dynamic translation comes from either the source dataset itself, or from a different dataset (such as the database table the data is being written to).

However, there are two other scenarios for providing the output schema: it can come from a text file (or spreadsheet) in which the definition is stored; or it can be actually defined dynamically as a list of attributes in a workspace.

Table-Based Schemas

In this scenario, the output schema is stored as some form of table in a text file or spreadsheet; for example:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FeatureType</td>
<td>AttributeName</td>
<td>AttriDataType</td>
<td>GeometryType</td>
</tr>
<tr>
<td>2</td>
<td>FireHalls</td>
<td>HallNumber</td>
<td>fme_char(30)</td>
<td>fme_point</td>
</tr>
<tr>
<td>3</td>
<td>FireHalls</td>
<td>Name</td>
<td>fme_char(30)</td>
<td>fme_char(30)</td>
</tr>
<tr>
<td>4</td>
<td>FireHalls</td>
<td>Address</td>
<td>fme_char(30)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>FireHalls</td>
<td>PhoneNumber</td>
<td>fme_char(30)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>FireHalls</td>
<td>LastUpdatedBy</td>
<td>fme_char(30)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Parks</td>
<td>ParkName</td>
<td>fme_char(30)</td>
<td>fme_polygon</td>
</tr>
<tr>
<td>8</td>
<td>Parks</td>
<td>ParkAddress</td>
<td>fme_char(50)</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Parks</td>
<td>ParkURL</td>
<td>fme_char(100)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Parks</td>
<td>LastUpdatedBy</td>
<td>fme_char(30)</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Zones</td>
<td>ZoneName</td>
<td>fme_char(30)</td>
<td>fme_polygon</td>
</tr>
<tr>
<td>12</td>
<td>Zones</td>
<td>ZoneCategory</td>
<td>fme_char(30)</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Zones</td>
<td>LastUpdatedBy</td>
<td>fme_char(30)</td>
<td></td>
</tr>
</tbody>
</table>

Here the author has listed a series of feature types, attributes, and geometry types that define the output schema. In FME they would use this schema by adding a Resource Reader. The format of the Resource Reader would be Schema (From Table):

In the parameters dialog for this Reader, there are parameters that specify which fields in the table represent which parts of the schema:
Geometry type is optional, but used in this example.

Attribute sequence is another optional parameter. It defines a field in the table that records the order that attributes should appear in.

Then, of course, this Reader must be used as the source for the output schema:

As always, the incoming attributes must be mapped to the outgoing schema. The best way here is the SchemaMapper transformer, since it too can use a lookup table to create its mappings.

Sister Intuitive says…

The great advantage of this method is that you don’t need to edit the workspace, or edit a dataset, to make schema changes. Once you change the output schema in the table, then that is automatically applied in the FME translation. That’s heavenly!
**Constructed Attribute Schemas**

This scenario is a way to construct an attribute schema using lists in FME. The schema is defined by using attributes in the list, for example:

![Transformer](image)

The Writer is told to use this schema in preference to any others by selecting it as the Source Schema:

![Select 'Schema Sources' Items](image)

**FME Data Types**

Both of the two preceding tools allow the user to define attribute type in an output schema. There are a set of valid datatypes in FME, which are as follows:

<table>
<thead>
<tr>
<th>General Field Type</th>
<th>Specific Field Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character Fields</td>
<td>fme_vARCHAR(width), fme_CHAR(width), fme_CHAR</td>
</tr>
<tr>
<td>Integer Fields</td>
<td>fme_UINT8, fme_INT16, fme_UINT16, fme_INT32, fme_UINT32, fme_INT64, fme_UINT64</td>
</tr>
<tr>
<td>Numeric Fields</td>
<td>fme_DECIMAL(width, decimal), fme_REAL32, fme_REAL64</td>
</tr>
<tr>
<td>Date-Time Fields</td>
<td>fme_DATETIME, fme_TIME, fme_DATE</td>
</tr>
<tr>
<td>Other Fields</td>
<td>fme_BUFFER, fme_BOOLEAN</td>
</tr>
</tbody>
</table>

**Miss Vector says…**

*The ability to construct a dynamic schema from attributes in a workspace has lots of possibilities. In fact,*
one of these FME transformers automatically creates dynamic schema attributes specifically so you can create a new schema. Which is it?

1. SchemaMapper  
2. AttributePivoter  
3. PythonCaller  
4. Clipper
In a previous exercise you created a new community map dataset for the planning department using a dynamic schema. At the time only two tables were defined, but now another one is required and the planning department wants you to update the workspace.

Rather than having to make changes each time they add more datasets, you figure that you can simply create an Excel spreadsheet containing the schema definition, so the planning team can edit it themselves and do the same for all future updates.

1) Inspect Spreadsheet
Open and examine the spreadsheet at C:\FMEData2016\Resources\CommunityMapSchema.xlsx.

If you don’t have Excel then open it in the FME Data Inspector and switch to Table View.

<table>
<thead>
<tr>
<th>FeatureType</th>
<th>AttributeName</th>
<th>AttrDataType</th>
<th>GeometryType</th>
<th>AttrOrder</th>
</tr>
</thead>
<tbody>
<tr>
<td>FireHalls</td>
<td>HallNumber</td>
<td>fme_char(30)</td>
<td>fme_point</td>
<td>1</td>
</tr>
<tr>
<td>FireHalls</td>
<td>Name</td>
<td>fme_char(30)</td>
<td>&lt;missing&gt;</td>
<td>2</td>
</tr>
<tr>
<td>FireHalls</td>
<td>Address</td>
<td>fme_char(30)</td>
<td>&lt;missing&gt;</td>
<td>3</td>
</tr>
<tr>
<td>FireHalls</td>
<td>PhoneNumber</td>
<td>fme_char(30)</td>
<td>&lt;missing&gt;</td>
<td>4</td>
</tr>
<tr>
<td>FireHalls</td>
<td>LastUpdatedBy</td>
<td>fme_char(30)</td>
<td>&lt;missing&gt;</td>
<td>5</td>
</tr>
<tr>
<td>Parks</td>
<td>ParkName</td>
<td>fme_char(30)</td>
<td>fme_polygon</td>
<td>1</td>
</tr>
<tr>
<td>Parks</td>
<td>ParkAddress</td>
<td>fme_char(30)</td>
<td>&lt;missing&gt;</td>
<td>2</td>
</tr>
<tr>
<td>Parks</td>
<td>ParkURL</td>
<td>fme_char(100)</td>
<td>&lt;missing&gt;</td>
<td>3</td>
</tr>
<tr>
<td>Zones</td>
<td>ZoneName</td>
<td>fme_char(30)</td>
<td>fme_polygon</td>
<td>1</td>
</tr>
<tr>
<td>Zones</td>
<td>ZoneCategory</td>
<td>fme_char(30)</td>
<td>&lt;missing&gt;</td>
<td>2</td>
</tr>
<tr>
<td>Zones</td>
<td>LastUpdatedBy</td>
<td>fme_char(30)</td>
<td>&lt;missing&gt;</td>
<td>3</td>
</tr>
</tbody>
</table>

The table has schema definitions for Firehalls, Parks, and Zones feature types.

2) Start Workbench
Start FME Workbench. Open the workspace from the previous exercise or the begin workspace listed above.
3) Delete CommunityMap Resource Reader
Because we are using a spreadsheet to define our output schemas, the CommunityMap Resource Reader is no longer needed. Locate it in the Navigator window, right-click on it, and choose Delete.

![Navigator window showing CommunityMap Resource Reader](image)

When prompted, click OK to confirm that all references relating to this dataset will also be removed.

4) Add Excel File as Reader Resource
Now select Readers > Add Reader as Resource. In the dialog that opens choose:

<table>
<thead>
<tr>
<th>Reader Format</th>
<th>Schema (From Table)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reader Dataset</td>
<td>C:\FMEData2016\Resources\CommunityMapSchema.xlsx</td>
</tr>
</tbody>
</table>

Click the parameters button (if you don’t you will be prompted to anyway). This dialog is where we can define how the table maps to the required schema.

Check the Reader parameters at the top. They should show the dataset is an Excel format file. Select Sheet1 as the table to use:
The first row should get used as the field names. If this is not the case, then click the parameters button above and set the values properly:

Next select the appropriate fields to match to the required parameters (for example Feature Type = FeatureType):

Click OK to close the dialog and again to add the Reader.

5) Set Dynamic Parameters
Now open the feature type properties dialog for the Writer feature type.

Under the User Attributes tab remove the LastUpdatedBy attribute, as we’ve added this to the spreadsheet definition for each type and no longer need it in here.

In the General tab click the Schema Sources edit button. Uncheck FireHalls and check CommunityMapSchema [SCHEMA_FROM_TABLE]:

Exercise: Creating a table-driven Community Map Dataset
Click OK and OK again to close these dialogs.

6) Add Reader
If you noticed, the schema spreadsheet included an entry for the Zones dataset, so add a Reader (not a Resource – we really want the data this time) as follows:

<table>
<thead>
<tr>
<th>Reader Format</th>
<th>MapInfo TAB (MITAB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reader Dataset</td>
<td>C:\FMEData2016\Data\Zoning\Zones.tab</td>
</tr>
</tbody>
</table>

Once added, connect its Reader feature type to the dynamic Writer feature type.

7) Save and Run Workspace
Save the workspace and then run it.
Inspect the output. Notice that all three feature types have been written, and that their attribute schema matches what was defined in the Excel spreadsheet; including the LastUpdatedBy field for each one.

Advanced Exercise
If you have the ability to edit the Excel spreadsheet then let’s do a couple more advanced steps (alternatively you can convert the spreadsheet to a CSV dataset and work in there).

8) Update Spreadsheet - 1
The planning team have decided they should rename some attributes, so open the spreadsheet and rename the following attributes
for the FireHalls feature type:

- Name to HallName
- Address to HallAddress
- PhoneNumber to HallPhone

Save the spreadsheet.

9) Update Spreadsheet - 2
If you run the workspace now it will run to completion, but there will be no values in the renamed fields. That’s because FME has no way to tell how to map the source data to the new schema.

We could simply add an AttributeRenamer transformer to handle this change, but the better way is to use the SchemaMapper. That way it can be made a little more dynamic.

So, in sheet 2 of the spreadsheet, enter:

<table>
<thead>
<tr>
<th>OldAttrName</th>
<th>NewAttrName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>HallName</td>
</tr>
<tr>
<td>Address</td>
<td>HallAddress</td>
</tr>
<tr>
<td>PhoneNumber</td>
<td>HallPhone</td>
</tr>
</tbody>
</table>

Then save the spreadsheet.

10) Add SchemaMapper

Add a SchemaMapper transformer to the workspace, with both output ports connected to the output feature type:

Open the parameters dialog. For the format select the edited Excel file. In the next dialog select Sheet 2 as the table to use.

In the final dialog, select Add > Attribute Map.
When prompted, select OldAttrName as the source field and NewAttrName as the Destination field. Check the box to Remove the original attributes (i.e. this is a renaming, not copying):

Click OK to close this dialog, then click Finish. Now save and run the workspace again.

This time the output will have its attributes properly mapped. So now the planning department can translate their data, decide on the output schema, and map source to destination attributes dynamically; all by editing this one Excel spreadsheet.

CONGRATULATIONS

By completing this exercise you have learned how to:
- Set up a spreadsheet for use as a dynamic translation schema
- Use the Schema (from Table) Reader to read a schema from a spreadsheet
- Map attributes in a dynamic workspace using a SchemaMapper transformer
Module Review

This chapter looked at advanced techniques for reading and writing datasets with FME.

What You Should Have Learned from this Module

The following are key points to be learned from this session:

Theory

- A Fanout is a way to divide data into a number of layers or a number of datasets, according to the value of a user-defined attribute.
- The Generic Reader and Writer allow a workspace to be dynamic in terms of format; it can read any format of data and write any format of data.
- Schema is composed of Feature Types, Attributes, and Geometry. A Dynamic translation is a universal workspace that can handle any combination of schema.
- The Schema in a dynamic workspace does not need to come from the source datasets; it can come from another dataset entirely, come from a lookup table, or can be constructed as list attributes inside the workspace.

FME Skills

- The ability to write output datasets to a zip file
- The ability to use a Feature Type Fanout and a Dataset Fanout
- The ability to use the Generic Reader and Writer
- The ability to create a simple Dynamic workspace
- The ability to add a Resource Reader
- The ability to use a Resource Reader or Lookup Table as the schema source in a dynamic translation.

Further Reading

For further reading why not browse articles tagged with Dynamic Schema or Fanout on our blog?
Questions

Here are the answers to the questions in this chapter.

Miss Vector says...

Fanouts are an important part of writing data with FME, so tell me, which of these statements are true?

1. You can have both a Feature Type Fanout and a Dataset Fanout in the same workspace
2. You can use a Feature Type Fanout with a database format, but not a Dataset Fanout
3. A fanout expression can be an attribute, or a constructed string, but not a user parameter
4. A fanout cannot be based on a format attribute such as fme_color

It's true you can have both types of fanout in the same workspace, but a database Writer won't do a Dataset Fanout (it can create multiple tables, but not multiple databases). A Fanout Expression can be any attribute (including FME attribute or format attribute) or string, and it can be a user parameter too!

Miss Vector says...

Now tell me which of these statements about Generic Reading/Writing are true?

1. Because you select an output folder, the Generic Writer won't write to a file format like AutoCAD DWG
2. If the feature types of the chosen format are limited to a single geometry, the Generic Writer will drop all features except a single geometry type
3. The Generic Writer does not support either type of Fanout
4. The ParameterFetcher transformer can be used to retrieve the format of data being Read/Written in order to route features in a specific way through the workspace

Most of these are untrue. The Generic Writer will create files, even though you only pick the output folder, and it will create multiple feature types if it's necessary to split data because of clashing geometry types. On a similar vein, fanouts are supported.

And, yes, the ParameterFetcher transformer will fetch the value of the format parameter letting you redirect data in the workspace (for example you might route data through a KMLStyler if KML is the chosen output format).

Miss Vector says...

It's important to get the concepts correct here, and I have some more statements only some of which are true:

1. A Dynamic workspace will read/write any format of data
2. A Dynamic workspace will read/write any feature types in the source data
3. A Dynamic workspace will read/write any attributes in the source data
4. A Dynamic workspace will read/write any geometry in the source data

Feature Types, Attributes, and Geometry are the three schema parts handled by dynamic workspaces.
However, format is not handled. This is the role of the Generic Reader/Writer. If you don't use the Generic Reader/Writer your dynamic workspace will not handle any format.

Basically, the term **Generic** means “any format”, while **Dynamic** means “any schema”. A workspace may be generic, or dynamic, or both, or neither!

---

**Miss Vector says...**

The ability to construct a dynamic schema from attributes in a workspace has lots of possibilities. In fact, one of these FME transformers automatically creates dynamic schema attributes specifically so you can create a new schema. Which is it?

1. SchemaMapper  
2. **AttributePivoter**  
3. PythonCaller  
4. Clipper

The AttributePivoter creates a whole new series of attributes that are completely different to the source schema; therefore it also generates a dynamic schema to assist you in writing the data. Check it out and you will see it creates a whole series of attribute_name{} and attribute_data_type{} list attributes.
Advanced Attribute Handling

Handling attributes is one of the most common operations carried out in FME. Users need to set attributes, edit them, and use them within most workspaces.

Professor Lynn Guistic says…

Good afternoon. I'm here to guide you through the technical aspects of attribute handling in FME. Don't worry, it's not rocket science, and it doesn't include the SchemaMapper!
Constructing Attributes

Besides constant attribute values FME also allows you to construct values using string manipulation and arithmetic calculations. This is achieved using the menu opened by clicking on the arrow in the Attribute Value field:

This is very useful because the attribute now no longer is a fixed value: it can be constructed from a mix of existing attributes and parameters. The two main methods are the Text Editor and Arithmetic Editor.

The string functions are mostly based around Tcl, the arithmetic functions around C.

Professor Lynn Guistic says…

Remember that besides existing to construct attributes, the text and arithmetic editors can be applied on most FME parameters. You wouldn’t, for example, have to use the AttributeManager arithmetic editor to create an attribute to then use in the 3DForcer transformer; you would just use the arithmetic editor directly in the 3DForcer itself.

Miss Vector says...

Do you know which transformers can be used to create attributes? Select all that apply:

1. AttributeCopier
2. AttributeCreator
3. AttributeManager
4. AttributeRenamer
Text Editor

The text editor - as you would expect - allows you to construct a text value. It includes all the usual string-handling functionality you would need, such as concatenation, trimming, padding, and case changing.

The text editor looks like this:

![Text Editor Image]

Here the user is constructing a simple address string by concatenating various existing attributes. Notice the menu on the left hand side. Existing attributes are listed here and were added into the string by double-clicking them.

Also notice the other menu options. The most important (for a text editor) are the String Functions:
These are the functions that can be used to manipulate the strings being used. For example, here the user is making sure the attributes being used are trimmed when used:

```plaintext
@Trim(@Value(PSTLADDRESS)), @Trim(@Value(PSTLCITY))
@Trim(@Value(PSTLPROV)), @Trim(@Value(COUNTRY))
```

## Arithmetic Editor

The arithmetic editor is much the same as the text editor, except that whatever is entered into the dialog will be evaluated as an arithmetic expression and a numeric result returned:

Here the user is calculating a resident's tax using a simple equation containing property area and a tax code. As with the text editor, existing attributes and arithmetic functions were obtained from the menu on the left-hand side.
FME Feature Functions

One other item in the menu of both text and arithmetic editors is FME Feature Functions:

These are functions that reach into the very heart of FME's core functionality. They are the building blocks that transformers are built upon; basic functionality that can return values to the editor.

For example, the @Area() function returns the area of the current feature (assuming it is a polygon). @Timestamp() returns the current time.

Some functions return strings, others return numeric values; therefore the available functions vary depending on whether the text or arithmetic editor is being used. In the screenshot above, the text editor functions are on the left and the arithmetic editor functions on the right.

FME Feature Functions are useful because they allow you to build processing directly into attribute creation, instead of using a separate transformer.

---

Professor Lynn Guistic says…

You might also be wondering why there are math functions inside the text editor and how you can use them.

Well, sometimes you need to calculate a numeric value as part of a string. If so, you can use the FME function @Evaluate() inside a text editor to carry out a mathematical calculation. This function understands all numerical operations, which is why they are included in the text editor dialog.

---

Replacing Other Transformers

Integrated text and arithmetic editors provide a great benefit for workspace creation. They allow attribute-creating functions to be carried out directly in a single transformer.
For example, the AttributeManager text editor can be used as a direct replacement for the StringConcatenator and ExpressionEvaluator transformers.

The AttributeManager could also replace the StringPadder and AttributeTrimmer transformers, albeit with a little less user-friendliness. If FME Feature Functions are used inside the editor, this transformer could also technically replace transformers such as the AreaCalculator, LengthCalculator, CoordinateCounter, TimeStamper, and many more.

This is usually a good thing. Workspaces will be more compact and well-defined when as many peripheral operations as possible are directly integrated into a single transformer. However, because it’s possible for an AttributeManager to be carrying out one of many, many operations, it is also more important to use Best Practice and ensure it has proper annotation.

If an AttributeManager is not properly annotated, it isn’t possible to determine from looking at the Workbench canvas what action it is carrying out!
## Exercise 1  Taxation Report Project

| Data                  | Building Footprints (AutoCAD DWG)  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Zoning Data (MapInfo TAB)</td>
</tr>
<tr>
<td>Overall Goal</td>
<td>Create a taxation report for each building</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>Constructing attribute values</td>
</tr>
<tr>
<td>Start Workspace</td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\Attributes-Ex1-Begin.fmw</td>
</tr>
<tr>
<td>End Workspace</td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\Attributes-Ex1-Complete.fmw</td>
</tr>
</tbody>
</table>

The annual property tax reports are due to be calculated and it is decided to use FME to carry out the processing. You must set up a workspace that calculates tax values for each building (based on size and zoning) and create a plain text file containing the results.

### 1) Inspect Source Data

Inspect the two source datasets you will be working with. They are:

<table>
<thead>
<tr>
<th>Reader Format</th>
<th>MapInfo TAB (MITAB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reader Dataset</td>
<td>C:\FMEData2016\Data\Zoning\Zones.tab</td>
</tr>
<tr>
<td>Reader Format</td>
<td>Autodesk AutoCAD DWG/DXF</td>
</tr>
<tr>
<td>Reader Dataset</td>
<td>C:\FMEData2016\Data\Parcels\BuildingFootprints.dwg</td>
</tr>
<tr>
<td>Reader Parameters</td>
<td>Group Entities By: Attribute Schema</td>
</tr>
</tbody>
</table>

### 2) Open Workspace

Open the workspace C:\FMEData2016\Workspaces\DesktopAdvanced\Attributes-Ex1-Begin.fmw. This is what you have created so far in order to set up Readers and Writers and a couple of preparatory transformations:
Open the parameters dialog for each transformer in turn.

The AttributeValueMapper is mapping from zone category to a TaxMultiplier value:

<table>
<thead>
<tr>
<th>Source Value</th>
<th>Destination Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comprehensive Development</td>
<td>0.9</td>
</tr>
<tr>
<td>Historic Area</td>
<td>1.1</td>
</tr>
<tr>
<td>Light Industrial</td>
<td>0.7</td>
</tr>
<tr>
<td>Multiple Family Dwelling</td>
<td>0.7</td>
</tr>
<tr>
<td>One Family Dwelling</td>
<td>1.2</td>
</tr>
<tr>
<td>Two Family Dwelling</td>
<td>1.0</td>
</tr>
<tr>
<td>Commercial</td>
<td>0.8</td>
</tr>
<tr>
<td>Industrial</td>
<td>0.5</td>
</tr>
</tbody>
</table>

The GOIDGenerator is merely creating a unique ID for each building footprint.

3) Add SpatialFilter
The first thing to do is transfer zoning information onto the building footprints. Add a SpatialFilter transformer to the workspace, with the zones as the Filter features and the buildings as the Candidates:

Open the parameters dialog. Under Tests to Perform, select Intersects and Contains.

The other parameters can remain as their default; but just check that Filter Type = Multiple Filters and that Pass Criteria = Pass Against One Filter

You may want to add an Inspector and run the workspace to ensure that the building features do - as expected - emerge from the SpatialFilter:Passed output port.
4) Add AttributeManager

Add an AttributeManager connected to the SpatialFilter:Passed output port and open the parameters dialog.

The first task is to create a numeric value for the taxation amount. So add a new attribute called TaxAmount and click the drop-down arrow to the right and choose the option for Open Arithmetic Editor:

![AttributeManager with TaxAmount option](image)

5) Calculate Tax Amount

The calculation for the tax amount is:

\[
\text{Building Footprint} \times \text{Tax Multiplier} \times \text{Tax Rate}
\]

...where Building Footprint is the area in square metres, Tax Multiplier is the value relating to the Zone Type, and Tax Rate is a value that changes each year and so should be provided by the user.

The result should also be rounded off to two decimal places.

So... start out by locating Area under FME Feature Functions and double-clicking it to add the area of the building footprint to the equation:

![Area function](image)

Next add a multiplication symbol, and locate the TaxMultiplier attribute and double-click it to add it to the equation:
Add another multiplication symbol. Now we need a user parameter to accept input from the user. We haven't defined it yet, but we can still do that inside the editor dialog.

Locate the published parameters section and click on <Create Published Parameter>

When prompted create a new parameter called TaxRate of type Float. Enter a default value of 0.2 and ensure the Optional checkbox is **not** checked:

Click OK and the parameter is added to the calculation:

```
@Area()*@Value(TaxMultiplier)*$(TaxRate)
```

Now we just need to round the result to two decimal places. Unfortunately the round function in this editor rounds only to an integer, so we must multiply the result by 100 first, then divide by 100 after.

So, add a multiplication symbol and the fixed value 100:

```
@Area()*@Value(TaxMultiplier)*$(TaxRate)*100
```

Now enclose all of the current statement inside a round function:
Finally add a division symbol and divide the whole statement by 100:

```plaintext
@round(@Area()*@Value(TaxMultiplier)*$(TaxRate)*100)/100
```

Click OK to close this dialog but keep the AttributeManager parameters open.

**6) Create Tax Report String**

The final task is to create a string of text that we can write to a report file. In this case the end-users of the information wish to receive a plain text file in the following structure:

```
Property: <PropertyID>
Tax Value: <TaxValue>
Dated: <Current Date>
```

So, inside the AttributeManager create a new attribute called `text_line_data` - this will match the output schema. Then click the drop-down arrow and open the text editor dialog.

Enter fixed values and add the attributes in the appropriate places to get the correct output:

```
Property: @Value(PropertyID)
Tax Value: @Value(TaxAmount)
```

Then use the Timestamp function with the format `/^d-^m-^Y` to create a date in the correct format:

```
Property: @Value(PropertyID)
Tax Value: @Value(TaxAmount)
Dated: @Timestamp(/^d-^m-^Y)
```

To get carriage returns in the output we need to specifically add those characters to the editor. To see such characters, select Options > Show Spaces/Tabs

Then locate Carriage Return (``) in the Special Characters menu and add one for each line:
Now click OK to close the dialog and OK again to close the AttributeManager dialog.

7) Connect Schema, Run Workspace
Connect the AttributeManager:Output port to the Text File Writer feature type:

Save and then run the workspace. The result should be a text file that looks like this:

```
Property: 47B666A49E4048E121F13A8E3E000043
Tax Value: 22.81
Dated: 12-03-2016

Property: 47B666A4D9ED88A021F13A8E41000104
Tax Value: 14.5
Dated: 12-03-2016

Property: 47B666A4C8A3901E21F13A8E410002D1
Tax Value: 13.02
Dated: 12-03-2016

Property: 47B666A4CC89898521F13A8E4100031D
Tax Value: 25.01
Dated: 12-03-2016

Property: 47B666A5998983E221F13A8E410004BA
Tax Value: 24.93
```
CONGRATULATIONS

By completing this exercise you have learned how to:

- Construct numeric values with the arithmetic editor
- Construct strings with the text editor
- Write data to a plain text file
Conditional Attribute Values

Conditional Attribute Values are a tool that can be used to replace many existing transformers of the same type.

Transformer-Based Attribute Mapping

Features can be divided within a workspace using transformers in a process called Conditional Filtering, and attributes can be set or created based on these divisions:

But this is just too many AttributeCreator transformers. Imagine if there were 100 values to handle! The workspace would be enormous!

One solution might be to use a simple AttributeValueMapper transformer. But that transformer only permits a simple condition. If a more advanced set of conditions is required, then we need to use Conditional Attribute Values.

What are Conditional Attribute Values?

Conditional attribute values are when the author sets an attribute value inside a transformer, but also sets a number of conditions that must be first applied to the data.

The option for conditional attributes is found in the drop-down dialog on most transformer parameters. In the AttributeManager, it appears like so:
The Conditional Definition dialog (with values filled in) looks like this:

Like the AttributeValueMapper, a series of conditions (left) map to different values (right). However, in contrast to the AttributeValueMapper, this dialog allows much more complex conditions than a simple 1:1 mapping. That’s because full test capabilities are built into this dialog.

The conditions are defined by double-clicking in the Test Condition field to open up a Tester-style dialog. The output value can also be set within this Tester dialog:
Conditional Values

Professor Lynn Guistic says...

Just like attribute construction, conditional values apply not just to attributes but to most FME parameters.

For example, I can create labels conditional upon certain tests using the Labeller transformer itself. I don't have to create the labels in an AttributeManager and then apply them in the Labeller as a separate task.

When the conditions are set then the original dialog – in this case an AttributeManager – looks like this, with the number of conditions defining the number of possible values:

When to use Conditional Attribute Values?

Conditional attribute values are great for when you need to map (or set) an attribute in relation to the value of an existing attribute, and when the conditions are more complex than can be handled in a simple AttributeValueMapper (or RangeValueMapper) transformer.

In essence, conditional values are like a combination of TestFilter and AttributeCreators in the range of functionality that they include.

Professor Lynn Guistic says...
For example, if you’re using the `?:` operator in an arithmetic editor, then you can stop being such a show-off and use conditional values instead.

**Miss Vector says...**

The output attribute "value" in a conditional setup can be which of these (select all that apply):

1. A simple value like a string or number
2. A value constructed from a text or arithmetic editor
3. No Action (i.e. the value will remain what it was)
4. A command to FME to terminate the translation
### Exercise 2 Flood Risk Project

<table>
<thead>
<tr>
<th>Data</th>
<th>Parks (MapInfo TAB)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overall Goal</strong></td>
<td>Assess flood risk for addresses based on elevation and distance from shore</td>
</tr>
<tr>
<td><strong>Demonstrates</strong></td>
<td>Conditional attribute values</td>
</tr>
<tr>
<td><strong>Start Workspace</strong></td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\Attributes-Ex2-Begin.fmw</td>
</tr>
<tr>
<td><strong>End Workspace</strong></td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\Attributes-Ex2-Complete.fmw</td>
</tr>
</tbody>
</table>

A colleague is working on a workspace to calculate the tsunami flood risk for all addresses in the city. The risk is adjudged to be a combination of closeness to the shoreline and elevation above sea level, and is calculated using this table:

<table>
<thead>
<tr>
<th>Distance from Shoreline (metres)</th>
<th>0-10m</th>
<th>10-25m</th>
<th>25-60m</th>
</tr>
</thead>
<tbody>
<tr>
<td>100m</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>200m</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>300m</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Your colleague has created the workspace up until the point at which each address has an elevation and distance from shoreline. Now the calculations need to start and he has asked for your assistance in finishing the project.

#### 1) Start Workbench

Open the workspace C:\FMEData2016\Workspaces\DesktopAdvanced\Attributes-Ex2-Begin.fmw. This is the workspace as your colleague has created it so far.

To find out what data we are dealing with, add Inspector transformers throughout the workspace and then run it.

You’ll probably want to inspect the source feature types (or the Reprojector, since the CDED data is a different coordinate system). You’ll also want to inspect the AttributeRenamer output.

Don’t forget you can set a Group-By in an Inspector’s parameters, which may be of use in visualizing which addresses are in which zone.
You’ll see how the addresses are assigned a zone denoting their distance from the shoreline, and also possess an elevation.

Ms Analyst says...

Before we get onto a full set of instructions for the exercise, try to consider how you might go about this task. You’ll need to think about:

- Can the data be mapped directly to flood risk, or does it need filtering first?

The zone is fairly easy to map, because it is a fixed value (100, 200, and 300). However, elevation is trickier because they are not fixed values; elevation could be any single value from 0 to 60.

- Which transformers would you use?

To filter the data the Tester and TestFilter transformers are the most obvious candidates, with maybe the AttributeRangeFilter; to map data, either the AttributeValueMapper or AttributeRangeMapper. Or why not just use the AttributeManager with conditional attributes?

- Should you combine methods?

Perhaps a combination method would work best, where you filter the data partially and then map it? If so, which data do you filter by and using which transformers?

- Which will produce the most aesthetic workspace?

Best Practice should always play a part in any workspace. If there are several solutions, then which produces the most aesthetic (good-looking) workspace? Are fewer transformers always better, or will that impact on the need to maintain the workspace?

Miss Vector says...

On consideration, I see three likely ways to carry out this project. I call them:

- Simple Filtering
- Complex Filtering
• **Conditional Values**

Simple Filtering filters the data and then maps it to the required values; as such it is a two-step process. It will require more transformers but will be simpler to understand and set up.

Complex Filtering filters the data in a single step, so no further mapping is required. It’s a single-step process, but – because the data is being filtered – needs more transformers than perhaps necessary. It is moderately complex.

Conditional Values will set the values directly depending on a set of inbuilt conditions. All the work can be done in a single transformer, so it is compact, but the setup and maintenance are considerably more complex.

On the following pages we’ll detail how to set up each method. Pick which one you want to try and follow the instructions. Alternatively, do each in turn – then you’ll be able to compare the different methods and decide which you think is the best.
This simple filtering method is a two-step process involving an AttributeFilter and several AttributeRangeMapper transformers. You should already have the start workspace open.

1) Place AttributeFilter

Place an AttributeFilter connected to the AttributeRenamer:

Open the parameters dialog. Select Zone as the attribute to filter by. In the Attribute Values field enter the values 100, 200, and 300:

You could use the Import function, but for so few values it’s hardly worth it.

Click OK to close the dialog and you’ll see a new output port added for each value you specified.

2) Add AttributeRangeMapper

Add an AttributeRangeMapper transformer and connect it to the 100 output port of the AttributeFilter:
Open the parameters dialog. As you’ll see this is a lookup table that involves ranges. We should be able to map the elevation range to a final flood risk using the information in the original table.

So, select Elevation as the Source Attribute. Enter FloodRisk as the Output Attribute:

In the Range Lookup Table, enter the From-To values as follows:

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Output Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>25</td>
<td>2</td>
</tr>
<tr>
<td>25</td>
<td>60</td>
<td>3</td>
</tr>
</tbody>
</table>

If an elevation falls exactly on one value (for example 25) it will be counted in the lower band (i.e. 10-25). Enter 999 as the Default, so that any features whose elevation does not match, for whatever reason, is flagged appropriately:

Click OK to close the dialog.

Miss Vector says...

_Hopefully you see that the Output Value numbers come from the flood risk table in the introduction to this exercise!_
3) Duplicate AttributeRangeMapper
Now we need to do the same thing for each of the other AttributeFilter output ports. Rather than set them up manually – as above – the easiest method is to copy the AttributeRangeMapper transformer that we just set up.

So, click on the existing AttributeRangeMapper and press Ctrl+D to duplicate it. Repeat and connect each duplicate to a different AttributeFilter output port.

The workspace will now look like this:

Now open the parameters dialog for each of the new AttributeRangeMapper transformers in turn and set up the correct Output Values in accordance with the original table of calculations.

The values will be:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>100m Zone</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200m Zone</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>300m Zone</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4) Add Inspector
Place a single Inspector transformer and connect each AttributeRangeMapper output to it.

Open the Inspector parameters dialog and under Group-By select the newly created attribute called FloodRisk.

**.1 UPDATE**

In 2016.1 the quickest way to do this is select all of the AttributeRangeMapper transformers and then add an Inspector using Quick Add. This will automatically connect all three transformers and is a new piece of functionality for this version of FME. You'll still need to manually set the group-by setting on the Inspector.

5) Save and Run Workspace
Save and run the workspace. You should see each address colored to match its flood risk. You can also turn off each zone in turn to see which addresses are most/least at risk.
Professor Lynn Guistic says…

If you’re sharp today, you’ll have noticed you could do this process in the reverse order. Instead of handling zone then elevation, you could handle elevation then zone using a combination of AttributeRangeFilter and AttributeValueMapper transformers.

CONGRATULATIONS

By completing this exercise you have learned how to:
- Filter data with a simple test in order to subdivide it for attribute mapping
- Map attribute values
Flood Risk Project: Complex Filtering Method

<table>
<thead>
<tr>
<th>Start Workspace</th>
<th>C:\FMEData2016\Workspaces\DesktopAdvanced\Attributes-Ex2-Begin.fmw</th>
</tr>
</thead>
<tbody>
<tr>
<td>End Workspace</td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\Attributes-Ex2b-Complete.fmw</td>
</tr>
</tbody>
</table>

This slightly more complex method is also a filtering process, but the filtering is all done in a single step - for both zones and elevation - with a TestFilter.

1) **Place TestFilter**

Place a TestFilter connected to the AttributeRenamer:

![Filter diagram]

What we want to get here is a separate output port for each flood risk value. So we’ll need to incorporate all of the tests into this one transformer.

2) **Set First TestFilter Condition**

Open the parameters dialog. See that there are fields for Test Condition and Output Port. Double-click the first Test Condition field and a Tester-like dialog will open:

![Test Condition dialog]

This can be the test for FloodRisk=1 (the highest). According to the table of calculations, this can occur only when Zone=100 and Elevation <= 10. So, set up the conditions to test for Zone = 100 AND Elevation <= 10. The important part here is to set up the test as an AND (i.e. both clauses) must be true.
Enter 1 into the Output Port parameter at the foot of the dialog:

Now click OK to close this part of the dialog.

The main TestFilter dialog now looks like this:

3) Set Second TestFilter Condition

Now double-click the next Test Condition to set up the condition for FloodRisk=2

According to the table, there are two conditions for FloodRisk=2. They are when:

- Zone = 200 AND Elevation <= 10
- Zone = 100 AND Elevation <= 25

So, enter four clauses; one each for Zone=100, Zone=200, Elevation<=10, Elevation<=25.

Now change the test type to Composite. In the Composite Expression field, enter:

- (1 AND 4) OR (2 AND 3)
Of course the composite expression field will depend on the order you entered the clauses in. If you entered them in a different order then you will need to adjust this field.

Enter 2 into the Output Port parameter and click OK to close this dialog. The main TestFilter dialog now looks like this:

4) Set Remaining TestFilter Conditions
Now repeat the above steps for each of the other flood risk values. There will be five conditions in all (one for each flood risk).

It may seem complicated, but it should be easy to get into a routine. Additionally, make use of the Duplicate buttons in these dialogs to speed up the process.

The final dialog will look like this:
The final test will be similar to the very first, with only two conditions, so it will be an AND rather than a Composite test.

It is very important to keep these in the correct order; otherwise a feature may pass the tests in the wrong order and be given a lesser risk than expected.

5) Add AttributeCreator

Add an AttributeCreator (or AttributeManager) connected to each TestFilter output port. Use the AttributeCreator to create the correct FloodRisk attribute (and value) for each output port (i.e. Port 1: FloodRisk = 1):

In fact, it’s probably easier to place one AttributeCreator and duplicate it for each port, editing the FloodRisk value each time.

6) Add Inspector

Place a single Inspector transformer and connect each AttributeCreator/Manager output to it. Open the Inspector parameters dialog and under Group-By select the newly created attribute called FloodRisk.

.1 UPDATE

In 2016.1 the quickest way to do this is select all of the AttributeCreator/Manager transformers and then add an Inspector using Quick Add. This will automatically connect all five transformers and is a new piece of functionality for this version of FME. You’ll still need to manually set the group-by setting on the Inspector.
7) Save and Run Workspace
Save and run the workspace. You should see each address colored to match its flood risk. You can also turn off each zone in turn to see which addresses are most/least at risk.

CONGRATULATIONS

*By completing this exercise you have learned how to:*
* Filter data with a complex test in order to subdivide it for attribute mapping*
Flood Risk Project: Conditional Values Method

<table>
<thead>
<tr>
<th>Start Workspace</th>
<th>C:\FMEData2016\Workspaces\DesktopAdvanced\Attributes-Ex2-Begin.fmw</th>
</tr>
</thead>
<tbody>
<tr>
<td>End Workspace</td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\Attributes-Ex2c-Complete.fmw</td>
</tr>
</tbody>
</table>

This is a one-step process involving an AttributeManager transformer.

1) Place AttributeManager
Place an AttributeManager transformer and connect it to the AttributeRenamer:

![Diagram of AttributeManager transformer and AttributeRenamer](image)

2) Set First AttributeManager Condition
Open the parameters dialog. Ignoring all of the existing attributes, scroll to the bottom of the dialog and in the <Add New Attribute> field enter FloodRisk.

In the Attribute Value field click the drop-down arrow and choose Conditional Value:

![Conditional Value dialog](image)

This opens a new dialog very similar to a Tester/TestFilter transformer. There are fields for Test Condition and Output Value. Double-click the first Test Condition field and a Tester-like dialog will open:
This can be the test for FloodRisk=1 (the highest). According to the table of calculations, this can occur only when Zone=100 and Elevation <= 10. So, set up the conditions to test for Zone = 100 AND Elevation <= 10. The important part here is to set up the test as an AND (i.e., both clauses) must be true.

Enter 1 into the Output Value parameter at the foot of the dialog:

Now click OK to close this part of the dialog.

The main Conditional Definition dialog now looks like this:
3) Set Second AttributeManager Condition

Now double-click the next Test Condition to set up the condition for FloodRisk=2

According to the table, there are two conditions for FloodRisk=2. They are when:

- Zone = 200 AND Elevation <= 10
- Zone = 100 AND Elevation <= 25

So, enter four clauses; one each for Zone=100, Zone=200, Elevation<=10, Elevation<=25.

Now change the test type to Composite. In the Composite Expression field, enter:

- (1 AND 4) OR (2 AND 3)

Of course the composite expression field will depend on the order you entered the clauses in. If you entered them in a different order then you will need to adjust this field.

Enter 2 into the Output Value parameter and click OK to close this dialog. The main Conditional Definition dialog now looks like this:
4) Set Remaining TestFilter Conditions

Now repeat the above steps for each of the other flood risk values. There will be five conditions in all (one for each flood risk).

It may seem complicated, but it should be easy to get into a routine. Additionally, make use of the Duplicate buttons in these dialogs to speed up the process.

The final dialog will look like this:
It is very important to keep these in the correct order; otherwise a feature may pass the tests in the wrong order and be given a lesser risk than expected.

The main AttributeCreator dialog now looks like this:

5) Add Inspector
Place a single Inspector transformer connected to the AttributeCreator. Open the Inspector parameters dialog and under Group-By select the newly created attribute called FloodRisk.

6) Save and Run Workspace
Save and run the workspace. You should see each address colored to match its flood risk. You can also turn off each zone in turn to see which addresses are most/least at risk.

CONGRATULATIONS

By completing this exercise you have learned how to:
- Use Conditional Attributes to map data according to a set of complex conditions
Adjacent Feature Attributes

Normally a feature in FME is self-contained. It might get processed as a group at some point, but other than that it doesn’t have any sort of relationship to other features in the workspace.

However, in some cases the ability for a feature to access the attributes of other features would be quite useful.

For example, take a tabular dataset of coordinates that are recorded as follows:

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0.0</td>
<td>+3.0</td>
</tr>
<tr>
<td>+3.2</td>
<td>+0.0</td>
</tr>
<tr>
<td>-3.2</td>
<td>+0.0</td>
</tr>
<tr>
<td>+0.0</td>
<td>+3.4</td>
</tr>
<tr>
<td>+4.2</td>
<td>+0.0</td>
</tr>
</tbody>
</table>

In this case each row is not an absolute coordinate; instead it is an offset from the previous one. Therefore, to calculate the true coordinates, each feature would need to know the coordinates of the previous feature, so that it could apply the offset.

This sort of scenario is catered for by Adjacent Feature Attributes in FME.

Adjacent Feature Functionality

Adjacent Feature functionality is activated by checking the box labelled Enable Adjacent Feature Attributes in an AttributeCreator or AttributeManager transformer:

This opens up a section of dialog in which the author can specify how many features preceding the current feature, or how many features that succeed it, should be made available. In the above screenshot attributes from the previous and subsequent two features will become available.

So that's how to expose adjacent attributes, and using them is almost as simple.

Using Multiple Feature Attributes
The simplest way to make use of the attributes retrieved from prior/subsequent features is through the text or arithmetic editors, where the list of feature attributes will have an expandable section for prior and subsequent features:

Notice above how attributes are available not only for the current feature, but also for the previous/subsequent two features. As with the current attribute, double-clicking an adjacent attribute adds it to the expression window:
In the above screenshot the user is creating a string composed of an address, and that of its neighbors (they won’t be neighbors in a spatial sense unless the data is already sorted that way).

You can see that prior and subsequent attribute values can be accessed simply by using feature[x]. where x is a positive or negative number that refers to a subsequent or prior feature.

**Professor Lynn Guistic says…**

*This is a first-class piece of functionality, excellent to the highest degree.*

*However... be aware that extra system resources are used for storage of adjacent features. Therefore translation performance will take a (fairly minor) hit when using these capabilities, the degree of which depends on the number of attributes being retained.*

**Missing Values**

The AttributeCreator and AttributeManager also have an option to specify what should happen if the attributes being used in a string are missing:

When the transformer tries to use a value that is missing (or null or empty) this option lets the user choose a replacement value, or to carry out no substitution.

Notice that this setting applies to attributes of the current feature, just as much as attributes of adjacent features.

**Miss Vector says...**

*My AttributeManager sets up NewAttribute = OldAttribute + feature[+1].OldAttribute*

*There are 100 features in my dataset. Given that feature[101].OldAttribute doesn’t exist, what will the value of NewAttribute be for the 100th feature?*

1. No value at all (empty attribute)
2. The same as feature[100].OldAttribute
3. It depends on the Substitute Value parameter
4. FME will crash and explode your computer*
You're working on a project mapping monthly precipitation (rainfall) in the city. You have been given a dataset like so:

<table>
<thead>
<tr>
<th>Month</th>
<th>Precipitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>168</td>
</tr>
<tr>
<td>Feb</td>
<td>273</td>
</tr>
<tr>
<td>Mar</td>
<td>387</td>
</tr>
<tr>
<td>Apr</td>
<td>476</td>
</tr>
<tr>
<td>May</td>
<td>541</td>
</tr>
<tr>
<td>Jun</td>
<td>595</td>
</tr>
<tr>
<td>Jul</td>
<td>631</td>
</tr>
<tr>
<td>Aug</td>
<td>668</td>
</tr>
<tr>
<td>Sep</td>
<td>719</td>
</tr>
<tr>
<td>Oct</td>
<td>840</td>
</tr>
<tr>
<td>Nov</td>
<td>1029</td>
</tr>
<tr>
<td>Dec</td>
<td>1191</td>
</tr>
</tbody>
</table>

Unfortunately, the numbers are a cumulative amount and you wanted to map individual figures for each month.

Rather than reaching into your desk drawer for a calculator, you decide to use FME to do the calculations!

1) Create Workspace
Create a workspace to translate the data as follows:

<table>
<thead>
<tr>
<th>Reader Format</th>
<th>Microsoft Excel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reader Dataset</td>
<td>C:\FMEData2016\Data\ElevationModel\Precipitation.xlsx</td>
</tr>
<tr>
<td>Writer Format</td>
<td>Microsoft Excel</td>
</tr>
<tr>
<td>Writer Dataset</td>
<td>C:\FMEData2016\Output\Training\MonthlyPrecipitation.xlsx</td>
</tr>
</tbody>
</table>

When creating the workspace, check the parameters for the Reader to ensure FME is recognizing the headers at the top of each column.

2) Add AttributeManager
To calculate precipitation for any given month you just need to subtract the previous month's cumulative total from the current month's cumulative total.

With FME you can use the Adjacent Feature Attribute functionality to fetch the previous month's number.
So, place an AttributeManager transformer between the Reader and Writer feature types:

![Diagram of transformer placement](image)

### 3) Set AttributeManager Parameters 1

Open the AttributeManager parameters dialog.

Expand the advanced set of attributes and check the box marked Enable Adjacent Feature Attributes. In the fields provided enter 1 for the Number of Prior Features to be kept.

Next set the parameter Substitute Missing, Null and Empty by: to Default Value and enter 0 into the Default Value field:

![AttributeManager parameters dialog](image)

#### Professor Lynn Guistic says...

*The substitution parameter is more important than perhaps most people recognize.*  
*Think about it: the first feature to be processed can’t have a prior feature, and the last feature to be processed won’t ever have a subsequent one. Therefore you always have to be careful in what you have set here.*  
*In this exercise we’re calculating a numeric value; therefore it makes sense to use 0 (zero) as the default replacement.*

### 4) Set AttributeManager Parameters 2

Now let’s calculate the new precipitation value.

In the Attribute Value field for the precipitation attribute, click the drop-down arrow and open the Arithmetic Editor:
In the arithmetic editor dialog use the menu on the left to select:

- The FME Feature Attribute: Precipitation
- The Math Operator – (minus)
- The FME Feature Attribute: Precipitation for feature[-1]

All of which should leave you with an expression looking like this:

```
@Value(Precipitation) - @Value(feature[-1].Precipitation)
```

Now you can see why it was so important to set the substitution field, because it's uncertain what result would occur from the above when feature[-1].Precipitation is missing!

Click OK to close the Arithmetic Editor dialog, and then click OK again to close the main AttributeManager dialog.

5) Save and Run Workspace
Save the workspace and then run it. Inspect the output.

The numbers start out looking correct, but quickly become wrong. Not even in Vancouver (I mean, Interopolis) does it rain 623mm in a single month!

The problem is this: unlike other occasions in FME, here we can’t simply overwrite the attribute we are working with. That’s because it skews the next calculation. i.e. the calculation for March needs to operate on February's original number, but instead it receives the value we've just overwritten it with!

The only way to solve this is by creating a new attribute.

6) Adjust Workspace
Return to the workspace. Edit the Writer schema by renaming the destination attribute: Precipitation to MonthlyPrecipitation:
Now return to the AttributeCreator and change it to create an entirely new attribute called MonthlyPrecipitation:

<table>
<thead>
<tr>
<th>Input Attribute</th>
<th>Output Attribute</th>
<th>Attribute Value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Month</td>
<td>Month</td>
<td></td>
<td>Do Nothing</td>
</tr>
<tr>
<td>Precipitation</td>
<td>Precipitation</td>
<td>@Value(Precipitation) @Value(feature...)</td>
<td>Do Nothing</td>
</tr>
<tr>
<td>MonthlyPrecipitation</td>
<td></td>
<td></td>
<td>Set Value</td>
</tr>
</tbody>
</table>

It's a pain to have to do, but blame me for leading you in the wrong direction at first! You can't even just rename Precipitation to MonthlyPrecipitation since, whatever you call it, it still fetches an incorrect value.

In fact, it might be quicker to just remove the transformer and replace it with a new one, instead of trying to copy the calculation to a new attribute.

**WARNING**

This should solve the problem. But! You should check the attribute mapping that results from this operation. In my workspace it looked like this:

That's obviously incorrect and you need to make sure it is MonthlyPrecipitation that is being mapped, not Precipitation:
7) Re-Run Workspace.

Save the workspace.

Before you re-run the workspace, check the Writer Parameter called "Overwrite Existing File" in the Navigator window.

Set it to Yes – if it isn’t already – so the output overwrites the destination dataset and doesn’t just append this data onto the same spreadsheet.

Also make sure the file you are writing to is not already open in Excel (or any other editor).

Re-run the workspace.

Inspect the output. This time the numbers should be correct:

```
Month       MonthlyPrecipitation
-----------  --------------------
 1  Jan       168
 2  Feb       105
 3  Mar       114
 4  Apr       89
 5  May       65
 6  Jun       54
 7  Jul       36
 8  Aug       37
 9  Sep       51
10 Oct       121
11 Nov       189
12 Dec       162
```

CONGRATULATIONS

By completing this exercise you have learned how to:
- Expose adjacent feature attributes
- Use adjacent feature attributes
- Handle missing values in attribute manipulation
Null Attributes

Null attributes are a relatively new, but very important, part of FME’s attribute handling.

What is a Null Value?

In general, a null attribute value is the equivalent of nothing. However, it’s important to be precise in our terminology because there are many ways to represent nothing:

- An attribute has a value that indicates nothingness (null)
- An attribute exists but has no value (empty)
- An attribute doesn’t exist (missing)
- A numeric attribute is NaN (Not a Number)
- A numeric attribute has a value of zero

In fact, Safe Software’s developers have identified fifteen (15) different ways for “nothing” to be represented in spatial and tabular data.

Professor Lynn Guistic says…

In case you are wondering, yes, our developers were the subject of many jokes for having spent six months “working on nothing”!

So when we talk about null, it has a particular meaning. For us, a null is an actual value that is deliberately set to signify that the information does not exist. It tells us that the lack of information is not a mistake – as a missing or empty value might.

Because there are so many different methods, this section will discuss ways to handle “nothing” attribute values, but with a particular emphasis on Null values.

How does FME Represent Nothing?

FME’s internal engine has its own state to represent null. However, when presented to the user, a null value is usually represented as <null>.

For example, this feature in the Logger has <null> for the ParkName attribute:

```
Parks: Feature is: 'Parks_LOGGED'
+----------------------------------------------------------+
| Feature Type: 'Parks_LOGGED'                               |
| Attribute(encoded: fme-system) : 'DogPark' has value 'N'   |
| Attribute(encoded: utf-8)   : 'NSStreet' has value ''     |
| Attribute(encoded: fme-system) : 'NeighborhoodName' has value 'West End' |
| Attribute(string)           : 'ParkId' has value '71'      |
| Attribute(string)           : 'ParkName' is <null>        |
| Attribute(string)           : 'RefParkId' has value '-9999'
```

Similarly, the FME Data Inspector will depict nulls as <null>:
Notice how it differentiates between states by also displaying `<missing>` (when an attribute does not exist, like for EWSStreet) or leaving the column blank (when an attribute exists but is empty, like NSStreet).

**Professor Lynn Guistic says…**

`<missing>` is an interesting concept. You might be asking, “how do we know when an attribute is missing”? But a better question is “how do we know that the attribute should exist”?

We know it should exist because it appears in the schema defined in a Reader. For example, in the above screenshot, NSStreet appears in the schema, but for some reason these features do not have that attribute.

Another common scenario where this happens, is when two streams of data are connected together. The attributes available to transformers will be the combination of the two streams, but each stream will only possess its own set of attributes.

Here, for example, the VertexCreator is rejecting 416 Zones features because the X/Y attributes it is using are `<missing>` from them!
Handling Nothing

Besides representing all forms of nothing in its interface, FME also allows nothing to be a condition in various tests, let's users set nothing values, and allows bulk updates from one form of nothing to another.

Recognizing Null Values

Various formats have various ways to represent nothing. But, if they support the concept of null, then FME will read any null attributes with a <null> value.

For a workspace to check for incoming nulls, the Tester transformer has specific operators to test for null, empty, and missing values:

Because the Tester interface is incorporated into many facets of FME (such as the TestFilter transformer) you can test for nulls wherever you find that interface.

NEW

New for FME2016 is the "Attribute Has a Value" test. This test returns true when an attribute is not null, is not empty, and is not missing - saving you the inconvenience of having to use those three tests separately.

Other transformers, such as the Matcher, also allow testing for nulls:
In the case of the Matcher the parameter decides whether null, empty, and missing values should be treated as different values. If set to No, then an attribute that is <null> is treated the same as an attribute that is <missing>. Otherwise, a match only occurs when <null> = <null>
Setting Null Values

The usual way to set an attribute value is with the AttributeCreator or AttributeManager, and these have an option in their drop-down menu to set a value to null:

The Conditional Attributes functionality in the AttributeManager also supports setting <null> values:

When you set an attribute to null, and send it to a Writer, then what happens depends upon the data format.

If the format supports nulls then the destination dataset will contain <null> attribute values.

If the format doesn’t support nulls, then FME will automatically convert the data to the closest representation that is supported.

Bulk Null Updates

The way to handle bulk updates of attributes is with the NullAttributeMapper transformer.

The NullAttributeMapper transformer allows the author to check values for any or all attributes on a feature, and convert them in bulk to or from null.

For example, here the author is checking for attributes that are either missing or empty, and converting them to nulls:
Here the author is checking a specific attribute for existing null values. If the value is set to null then it gets replaced with a zero. Presumably this must be a numeric field. If it was a text field perhaps instead the author would set it to an empty string:

Professor Lynn Guistic says...

It's good to be aware of nulls and test workspaces when updating from FME versions prior to FME2014. Some formats and transformers may now be producing null values where before they would have been either empty or missing.

Miss Vector says...

My Reader format supports nulls and includes known null values in the data. My Writer format is a simple text format that does not support nulls. What must I do to get my workspace to work correctly?

1. Delete the attributes with the AttributeRemover
2. Set the advanced Reader parameter “Read Nulls as Empty” to Yes
3. Use the NullAttributeMapper to convert all <null> values to <empty>
4. Nothing, the Writer will convert the values as necessary
Exercise 4  Parks Dataset Sorting

<table>
<thead>
<tr>
<th>Data</th>
<th>Parks (MapInfo TAB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>Sort parks into alphabetical order</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>Null attribute handling</td>
</tr>
<tr>
<td>Start Workspace</td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\Attributes-Ex4-Begin.fmw</td>
</tr>
<tr>
<td>End Workspace</td>
<td>C:\FMEData2016\Workspaces\DesktopAdvanced\Attributes-Ex4-Complete.fmw</td>
</tr>
</tbody>
</table>

In this workspace a colleague is trying to write out a list of parks to a Geodatabase dataset. It’s important to them that the parks are in alphabetical order – according to their name – and that features with no park names are written as null and appear last in the dataset.

However, the workspace they have does not seem to be doing what they need. The parks are sorted alphabetically, but un-named parks always appear first.

1) Start Workbench

Open the workspace C:\FMEData2016\Workspaces\DesktopAdvanced\Attributes-Ex4-Begin.fmw

Inspect the source dataset by clicking the source feature type and choosing the pop-up inspection button.

In the Data Inspector examine the data in the Table View window. You’ll see that the data is in order of ID, not name and that there are <missing> values scattered throughout:

<table>
<thead>
<tr>
<th>ParkId</th>
<th>RefParkId</th>
<th>ParkName</th>
<th>NeighborhoodName</th>
<th>EWStreet</th>
<th>NSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>&lt;missing&gt;</td>
<td>Kitsilano</td>
<td>&lt;missing&gt;</td>
<td>&lt;my</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>208 Rosemary Brown Park</td>
<td>Kitsilano</td>
<td>W 11th Avenue Vine</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>141 Tea Swamp Park</td>
<td>Mount Pleasant</td>
<td>E 15th Avenue Sop</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>&lt;missing&gt;</td>
<td>Strathcona</td>
<td>&lt;missing&gt;</td>
<td>&lt;my</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>202 Morton Park</td>
<td>West End</td>
<td>Morton Avenue Dev</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>&lt;missing&gt; Mclniere Park</td>
<td>Kitsilano</td>
<td>&lt;missing&gt;</td>
<td>&lt;my</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>&lt;missing&gt; Granville Park</td>
<td>Fairview</td>
<td>&lt;missing&gt;</td>
<td>&lt;my</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>&lt;missing&gt;</td>
<td>Mount Pleasant</td>
<td>&lt;missing&gt;</td>
<td>&lt;my</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>15 Creekside Park</td>
<td>Mount Pleasant</td>
<td>Terminal Avenue Que</td>
<td></td>
</tr>
</tbody>
</table>
We know the values in here are currently listed as <missing> so set the “If Attribute Value Is” parameter to Missing (Selected Attributes Only)

We want to map these to a value that will appear at the bottom of any alphabetically sorted list, so change “Map To” to New Value and enter ZZZ as the new value.

Click OK to close the dialog.

3) Add NullAttributeMapper
Now add a second NullAttributeMapper; this time it should be connected after the Sorter.

Open the dialog and, once again, ensure “Map” is set to Selected Attributes and select the ParkName attribute. This time turn the ZZZ values back to nulls:

Technically we could just turn them back into <missing>; the Geodatabase Writer will write them out as nulls. However, assuming we didn’t know that, null is the safer option and bound to give us what we want.

4) Save and Run Workspace
Save the workspace and then run it. Inspect the output. This time the data should be sorted by ParkName, but with all null values at the end of the dataset:
5) Fix RefParkId
Your colleague now asks you to fix the RefParkId field. You'll have noticed that a lot of the values are -9999. That's the MapInfo equivalent of "nothing" but for Geodatabase it would be better to set these to proper nulls.

**Miss Vector says...**

That's very simple to do. Take a moment to think about how before you look at the instructions!

To do this open the parameters dialog for the first NullAttributeMapper. Add RefParkId to the list of attributes being processed. Then add -9999 to the Or If Attribute Value Is field:
Now open the second NullAttributeMapper and add RefParkId to the list of attributes being processed.

Now these values will get mapped to ZZZ with the missing ParkName values. Then they will be turned into true nulls by the second NullAttributeMapper.

CONGRATULATIONS

By completing this exercise you have learned how to:
- Identify null and missing attribute values
- Set null and missing attribute values
Module Review

This chapter looked at advanced techniques for handling attributes in FME.

What You Should Have Learned from this Module

The following are key points to be learned from this session:

Theory

- Attributes can be constructed either with a series of transformers or inside a single transformer using Text and Arithmetic editors.
- Constructing an attribute as a series of transformers is more self-documenting, but a single transformer is more elegant and reduces canvas clutter.
- Conditional Values are when an author constructs an attribute according to a number of test conditions.
- Multiple Feature Attributes is a technique where any feature can access the attributes of previous or subsequent features.
- Null attributes are those whose state indicates lack of a value.

FME Skills

- The ability to construct attributes with either the Text or Arithmetic editors.
- The ability to apply conditional attribute values, and the knowledge to know when this is a good choice of approach.
- The ability to use Multiple Feature Attributes.
- The ability to use test for null values, and change values to or from null.

Further Reading

For further reading take a look at articles tagged with Null on our blog.
Questions

Here are the answers to the questions in this chapter.

**Miss Vector says...**

*Do you know which transformers can be used to create attributes? Select all that apply:*

1. AttributeCopier
2. AttributeCreator
3. AttributeManager
4. AttributeRenamer

The AttributeCreator is obvious. The AttributeManager is new for 2016. The AttributeRenamer did allow you to create attributes, but that ability was removed in 2016. There should be no need for it with the new AttributeManager.

**Miss Vector says...**

*The output attribute "value" in a conditional setup can be which of these (select all that apply):*

1. A simple value like a string or number
2. A value constructed from a text or arithmetic editor
3. No Action (i.e. the value will remain what it was)
4. A command to FME to terminate the translation

Yes, all of these are valid. You can type in a simple value or construct one with an editor, or even set the value to a user parameter. But the Output Value field also does not need to be a "value" at all! It can be any action on the usual dropdown menu, including Null, No Action, or Stop Translation.
Miss Vector says...

\[ \text{My AttributeManager sets up } \text{NewAttribute} = \text{OldAttribute} + \text{feature}[+1].\text{OldAttribute} \]

There are 100 features in my dataset. Given that \text{feature}[101].\text{OldAttribute} doesn't exist, what will the value of \text{NewAttribute} be for the 100th feature?

1. No value at all (empty attribute)
2. The same as \text{feature}[100].\text{OldAttribute}
3. It depends on the Substitute Value parameter
4. FME will crash and explode your computer

You as the author get the choice of what happens when a value is missing, using the Substitute Value parameter, and that includes values that are missing because they are out of range.

---

Miss Vector says...

\[ \text{My Reader format supports nulls and includes known null values in the data. My Writer format is a simple text format that does not support nulls. What must I do to get my workspace to work correctly?} \]

1. Delete the attributes with the AttributeRemover
2. Set the advanced Reader parameter “Read Nulls as Empty” to Yes
3. Use the NullAttributeMapper to convert all <null> values to <empty>
4. Nothing, the Writer will convert the values as necessary

If a format doesn’t support nulls then the Writer will write the data in a format as close to null as possible for that format. Sometimes it will be an empty value, other formats have a specific value for null (like -9999). You only need to be cautious if your Writer was created in an FME version prior to FME2014.
Course Wrap-Up

Although your FME training is now at an end, there is a good supply of expert information available for future assistance.
Product Information and Resources

Safe Software Web Site

The Safe Software web site is the official information source for all things FME. It includes information on FME products, Safe Software services, FME solutions, FME support and Safe Software itself.

Safe Support Team

Behind FME are passionate, fun, and knowledgeable experts, ready to help you succeed, with a support team philosophy built on the principle of knowledge transfer.
Your Local Partner

Safe Software has partners and resellers around the world to provide expertise and services in your region and your language.

You can find a list of official partners on the Safe Software web site.

Safe Software Blog

The Safe Software blog provides technical information about FME, articles about customers' use cases, and general thoughts on spatial data interoperability.
FME Manuals and Documentation

Use the Help function in FME Workbench to access help and other documentation for FME Desktop. Alternatively, look on our web site under the Knowledge Center section.
Community Information and Resources

Safe Software actively promotes users of FME to become part of the FME Community.

The FME Knowledge Center

The FME Knowledge Center is our community web site - a one-stop shop for all community resources, plus tools for browsing documentation and downloads.

Knowledge Base

The FME Knowledge Base contains a wealth of information; including tips, tricks, examples, and FAQs. There are sections on both FME Desktop and FME Server, with articles on topics from installation and licensing to the most advanced translation and transformation tasks.

Q&A Forum

FME community members post FME-related messages, ask questions, and share in answering other users’ questions. Members earn “reputation” and “badges” and there is a leaderboard of the top-participating users. Come and see how they can help with your FME projects!

Ideas Exchange

FME development is very much user-driven. The Ideas Exchange gives users the chance to post their ideas for new FME functionality, or improvements to existing functionality, and allows everyone to vote on the proposed ideas. The more votes an idea gets, the more likely it is to be implemented!
The FME Channel

This FME YouTube channel is for those demos that can only be properly appreciated through a screencast or movie. Besides this there are a host of explanatory and helpful movies, including recordings of most training and tutorials.
Feedback and Certificates

The format of this training course undergoes regular changes prompted by comments and feedback from previous courses.

Course Feedback

Miss Vector says...

There’s one final set of questions – and this time you’ll be telling me if the answers are correct or not!

Safe Software greatly values feedback from training course attendees and our feedback form is your chance to tell us what you really think about how well we’re meeting your training goals.

You can fill in the feedback form now, but you’ll also be reminded by email shortly after your course. Safe Software's partners who carry out training may ask that you fill in a separate form, but you can also use the official Safe Software form if you wish.

Certificates

Mr. E. Dict, (Attorney of FME Law)says...

In order to prove you have taken this training course, a certificate will be emailed automatically to anyone who was logged on for the duration of Safe Software hosted courses.
Thank You

Thank you for attending this FME training course.

All of us at SAFE SOFTWARE wish you good luck and lots of fun as you create exciting new FME INVENTIONS.
Congratulations!

As a reward for reading this far, here's a little quiz for you to try out.

As you may know, if you opt in, FME records information about what formats and transformers you are using. It lets us create general statistics and shows us where to focus our development work.

The following questions are based on these statistics for FME2016 beta (from the time it first appeared on our web site, up until December 2015). They refer to the number of times a transformer or format was added to a workspace, by a user.

Each question has a number of correct answers. Your job is to pick one of these answers, but get the LOWEST score you can; i.e. identify a transformer or format that was used as little as possible during the beta period!

The ultimate answer is one that is pointless (i.e. with no score!)

So, here goes...

Miss Vector says...

*Without looking in Workbench (that would be cheating) identify a transformer that is in the Filters category in the transformers gallery in FME2016.*

One answer is very high (you can probably guess which). But there are two transformers that were not used at all in the 2016 beta phase and so will score zero points. Will the transformer you select be one of those?!

*If the transformer you select is incorrect - i.e. it is not a transformer in the Filters category - then you will score 100 points!*

Dr Workbench says...

*Again without looking in Workbench, identify a Reader in FME2016 that includes the word "XML" in its full FME name.*

I don’t just want an XML-based format. For example “GeoRSS/RSS Feed” might be XML-based, but it doesn’t have the string “XML” in its name, so it doesn’t count. There are nine different formats to choose from, and four of them will score you zero points, so your chances are quite good!

Sister Intuitive says...

*I’ll be kind to you. All I need is a transformer whose name begins with the word "Attribute".*

There are lots of these - and I’m including aliases - so you shouldn’t get a completely wrong answer. However, it will take a lot of meditating to choose the one transformer that is a zero score.

Police Chief Webb-Mapp says...
So! You think you’re good, do you? Well here is a really tough question to sort out the good guys from the bad.

The What's Great in 2015 article on the FME blog listed 18 transformers that were new in 2015 (or a 2014 Service Pack). All you have to do is name one.

That shouldn’t be too hard, should it? Not many were used much in the 2016 beta period, so if you can get it right, you will score well!

---

Miss Vector says...

Have you written down your answers? And are you sure you didn’t cheat? Good.

To find out what your score is, visit this page: [http://www.safe.com/training/quiz/](http://www.safe.com/training/quiz/)

Good luck!