FME® Desktop Advanced Training Manual

Contents

Document and Copyright Information

Document and Copyright Information

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 (data.vancouver.ca). 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

(http://en.wikipedia.org/wiki/Vancouver_Fire_and_Rescue_Services), used under a Creative Commons Attribution-ShareAlike license

Stanley Park GPS Trail: Used with kind permission of VancouverTrails.com. See http://www.vancouvertrails.com/trails/stanley-park/.

Copyright

© 2005–2015 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: services@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 trademarks of Safe Software Inc. All brand or product names are trademarks or registered trademarks of their respective companies or organizations.

Document Information

Document Name: Desktop Advanced Training Manual

Updated: April 2015

Chapter 0 - Welcome to Safe Software

Course Details

Course Details

Course Structure
Training Philosophy
Prerequisites

FME Desktop Advanced Training 2015

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

Course Structure

This training material covers advanced topics for creating and running FME workspaces.

It is comprised of the following sections:

1. Advanced Parameter Use
2. Performance Considerations
3. Custom Transformers
4. Advanced Reading and Writing
5. Advanced Attribute Handling

The content may be spread over several days, or your instructor may choose particular sections to be covered.

Training Philosophy

This training 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 workspace authors who wish to take their FME skills to the next level.

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
- Managing Readers, Writers and feature types in a workspace
- Data filtering and attribute management techniques in FME Workbench

About the Manual

About the Manual

Icons

FME Desktop Advanced Training 2015

The FME Desktop Advanced training manual includes detailed material to help you remember your training.

During training we suggest you pay close attention to your instructor; they may cover the course content in a different order than the manuals, and will skip or add new content to better customize the course to your needs.

You'll find most of the instructor's PowerPoint slides reproduced in your manuals, as above. This will help you relate the manual content to the topic the instructor is discussing.

Icons

In the training manual you may see the following icons...

Tip	*Additional advice to help apply the knowledge you have learned.*
!	*A warning where misuse of FME could lead to difficulties.*
	Extra challenges for students who are quick to finish and want to take an exercise a little bit further.
New	*A feature new to, or significantly changed in, the most recent version of FME.*
Q A	*Questions about the course content and how it is applied.*

Also, people from the city of Interopolis will also appear from time-to-time to give you advice and dispense FME-related wisdom.

Course Resources

Course Resources

On Your Training Computer
Amenities

FME Desktop Advanced Training 2015

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

On Your Training Computer

The following applications may already be installed, licensed, and located on your training computer (real or virtual):

- FME Desktop Version 2015
- Adobe Reader
- Google Earth

A collection of sample datasets and workspaces are available in C:\FMEData2015.

Workspaces for this course are located in: C:\FMEData2015\Workspaces\DesktopAdvanced.

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

For more information refer to: C:\FMEData2015\readme.txt.

Amenities

For in-person training, your instructor will explain the facilities and amenities available to you during the course.

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.

Introduction

Introduction

FME Version
Sample Data

FME Desktop Advanced Training 2015

This document is an advanced training manual for FME Desktop 2015

FME Version

This training material is designed specifically for use with FME2015. You may not have some of the functionality described if you use an older version of FME.

Sample Data

This sample data required for the exercises in this document can be obtained from:

www.safe.com/fmedata

Chapter 1 - Advanced Parameter Use

Parameters and FME

Parameters control how FME operates, and can be set by either the workspace author or the end user.

What are Parameters?

Parameters, in basic terms, 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 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

FME parameters are those built into FME. 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.

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 the user may have no experience of Workbench and not 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.
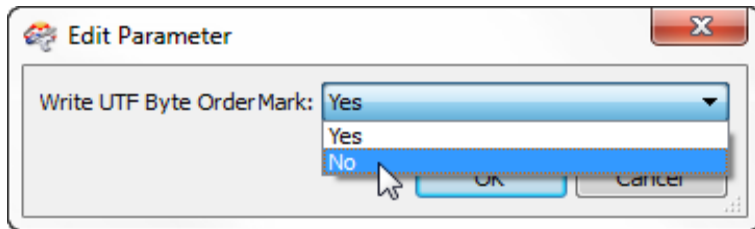

They double-click the parameter to open a dialog in which they can change the parameter value.

User Parameters

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

User parameters appear in a special section of the Navigator window, labelled User Parameters. Here, for example, two user parameters are defined.
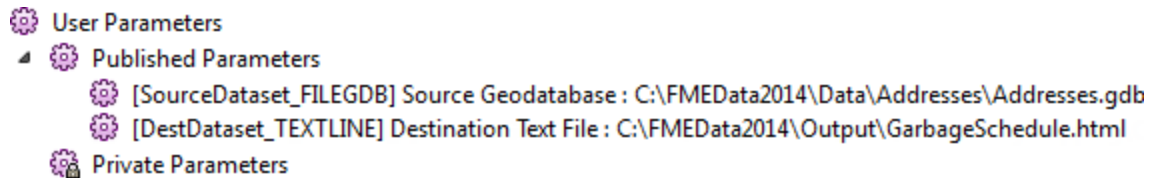
Each user parameter allows the end-user of a workspace to enter information. This may be through a simple dialog (in Workbench or the FME Quick Translator) or it might be through a web page on FME Server.

Ms. Analyst says…

"We used to use the term Published Parameters. It sort of still applies – you can see it used in the screenshot above – but there's some confusion, as we'll see. It's better to use the term User Parameters."
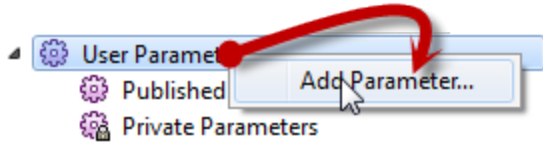
A user parameter is created by a workspace author by – in most cases – right-clicking on the User Parameters label in the Navigator window and choosing Add Parameter:
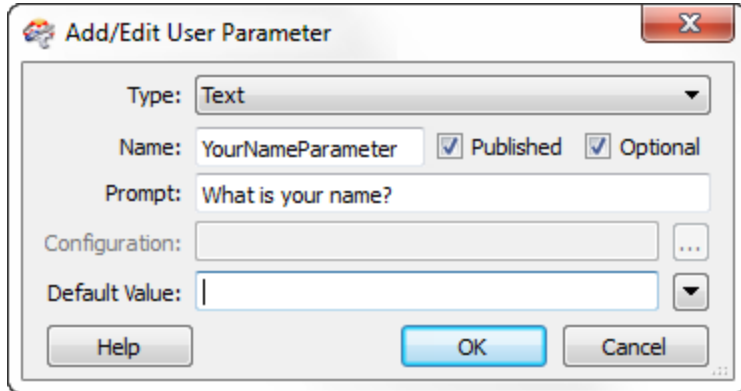
A dialog appears in which the author can define the parameter.

In this case they are creating a parameter in which the user can enter their name:
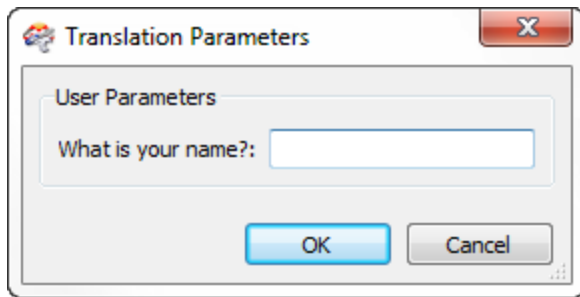
Now when the end-user runs the workspace, they will be prompted to enter their name:

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 an attribute in a workspace.

For example, in this workspace a source dataset undergoes transformation and the output is flagged up with the name of the person who carried out the transformation (i.e. who ran the workspace);
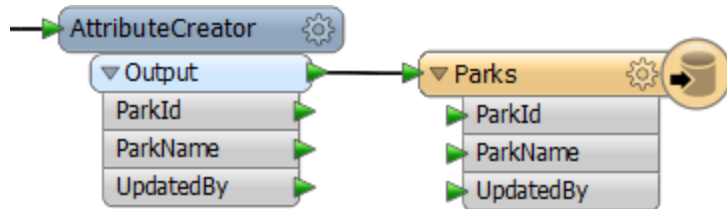
The workspace author creates a user parameter for the user to enter their name (as in the previous screenshots) and then adds an AttributeCreator transformer to create the required output attribute.

In the FME parameters for the AttributeCreator, the author sets the value of the attribute to that provided by the user parameter:
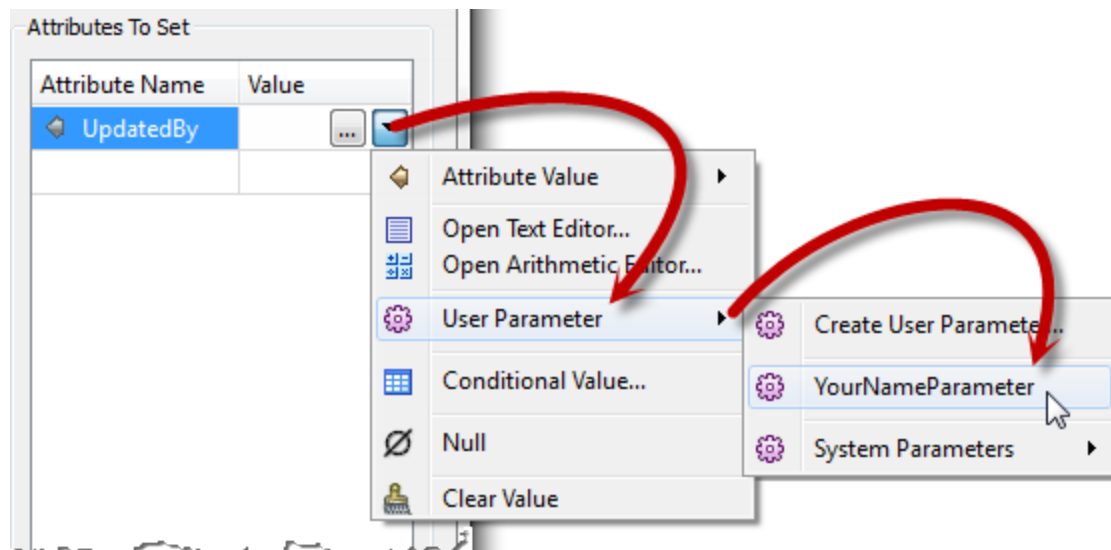
Now when the workspace is run, the end user can enter their name into a text field, and have it entered into an attribute in the output.

Exercise 1a Basic Parameters	
Scenario	FME author; City of Interopolis
Data	Parks (MapInfo Tab)
Overall Goal	Add user input to metadata fields
Demonstrates	Use of FME Parameters. Creation and Use of User Parameters
Starting Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise1a-Begin.fmw*
Finished Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise1a-Complete.fmw*

Here we have a workspace that was created to translate a parks dataset from MapInfo to KML, plus also write an XML metadata file to show who translated the data and when.

At the moment, all of the XML metadata fields are hard-coded in an AttributeCreator transformer. We'll need to create user parameters to take the place of these hard-coded values.

1) Start Workbench

Open the workspace *C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise1a-Begin.fmw*.

Notice the transformers in the workspace.

The Sampler ensures that only one record is written to the output metadata, by discarding all but one feature.
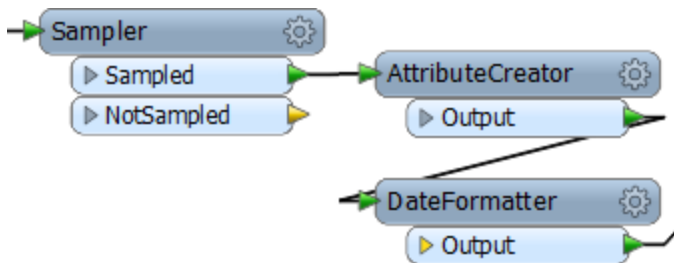
The AttributeCreator creates a set of attributes and the DateFormatter formats the date attribute to an XML-compatible ISO format.

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 are the parameters for the DateFormatter:
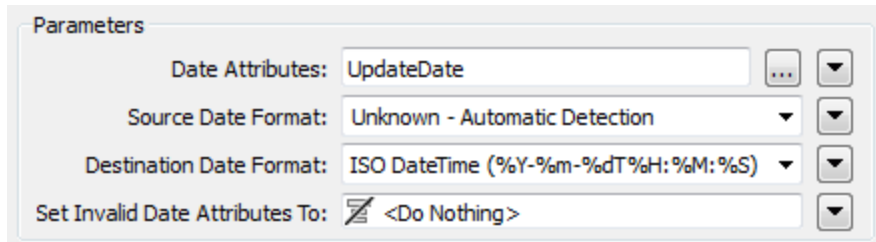
You can also find the same parameters in the Navigator window:
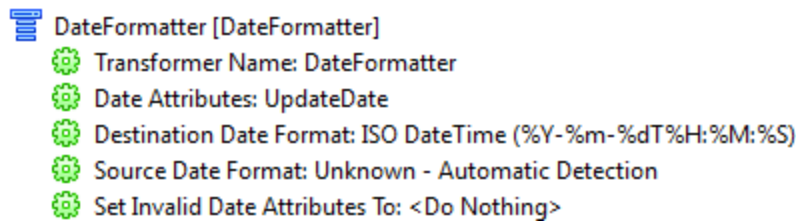
2) Change XML Writer Parameter

As a workspace author we need to change one of the Writer parameters. We don't want the end-user to be setting this, so we'll set it ourselves and not create a user parameter.

In the Navigator window locate the XML Writer. Expand the parameters list. Locate the parameter labelled Pretty Print and double-click on it.
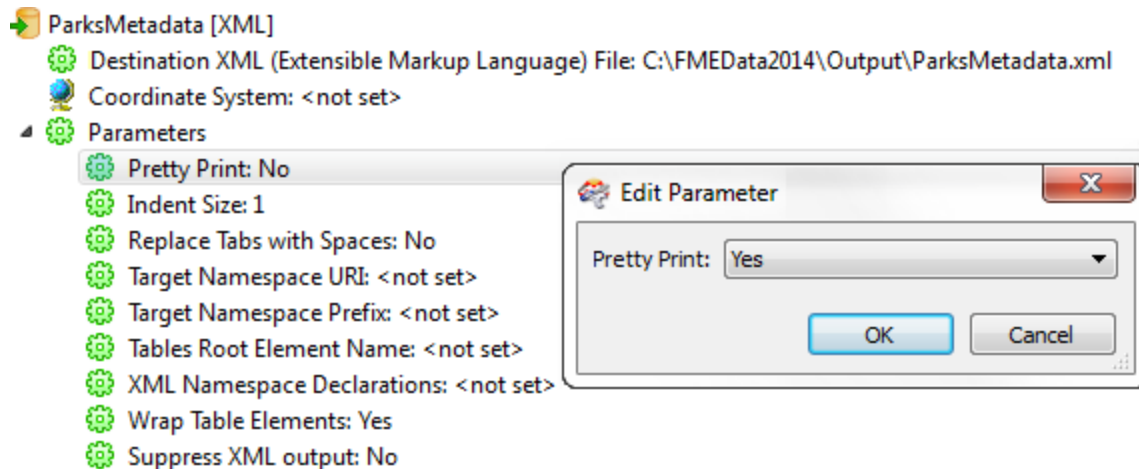
In the dialog that opens, change the value to Yes and then click OK to close the dialog. We have now changed an FME parameter.

3) Create User Parameter

We now need to create some user parameters for the end-user to enter information into the workspace.

Firstly, locate the User Parameters section of the Navigator window, right-click on it, and choose the option to Add Parameter:
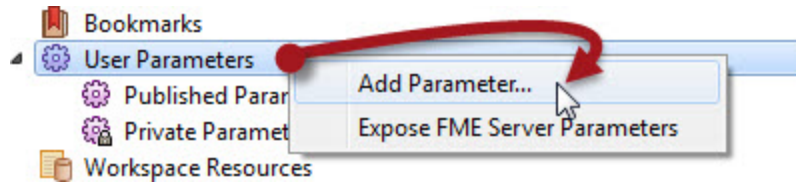
In the new dialog, select Text as the type of parameter to create (there will be more on parameter types in the next section). Each parameter needs a name, so call this one UserNameParam. Now enter a prompt, such as "Enter your name."

Click OK to close the dialog and create the parameter, which will now appear in the Navigator window.
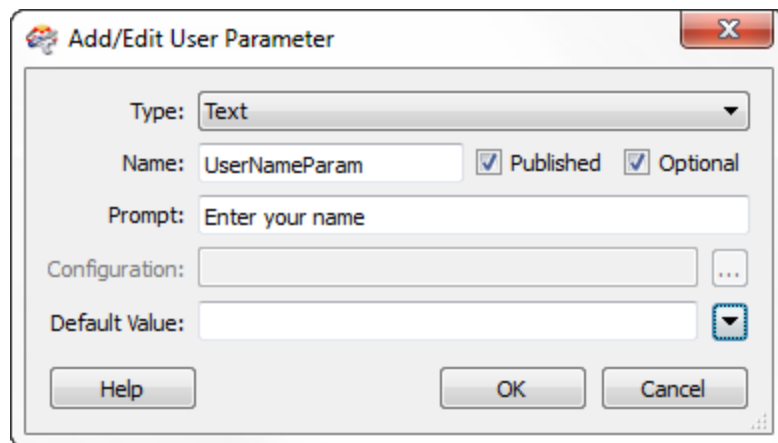
4) Create Remaining User Parameters

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

There are a number of ways to extract the value from a user parameter into a workspace. 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:

Attribute Name	Value
AuthorName	$(UserNameParam)
AuthorEmail	*k* mark.ireland@safe.com
AuthorCompany	*k* Safe Software
UpdateDate	*k* TODAY

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:
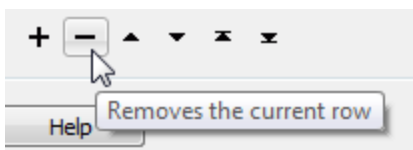
> *Ms. Analyst says…*
>
> *"Did you notice the UpdateDate attribute is set to a value of "TODAY"?*
> *When processed by the DateFormatter it will be turned into whatever the current date is.*
>
> *Similar relative date values can also be used, such as "Yesterday," "Tomorrow," or "Last Thursday"*

6) Use User Parameter – Method 2

A second way to extract the value from a user parameter is with a ParameterFetcher transformer.
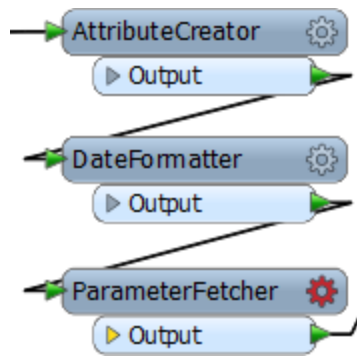
Place a ParameterFetcher transformer (after the DateFormatter is fine). Open the parameters dialog.

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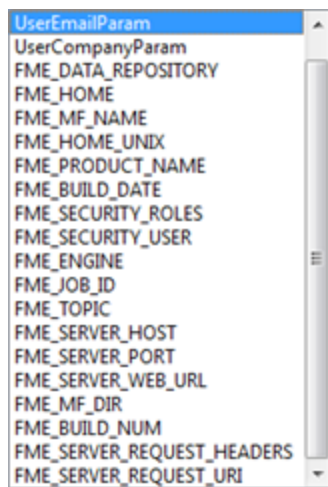
Ms. Analyst says...

"Did you notice that the list of parameters includes many FME-related system parameters?

These are particularly useful for use on FME Server."

7) 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.
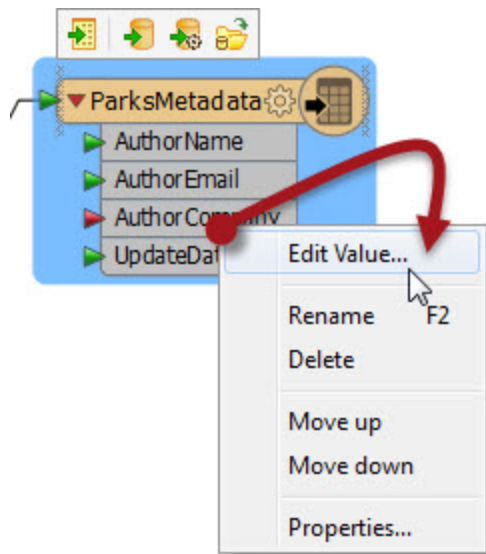
In the dialog that opens, you can enter a fixed (constant) value, but in our case we'll click on the drop-down arrow, select User Parameters, and then select UserCompanyParam:
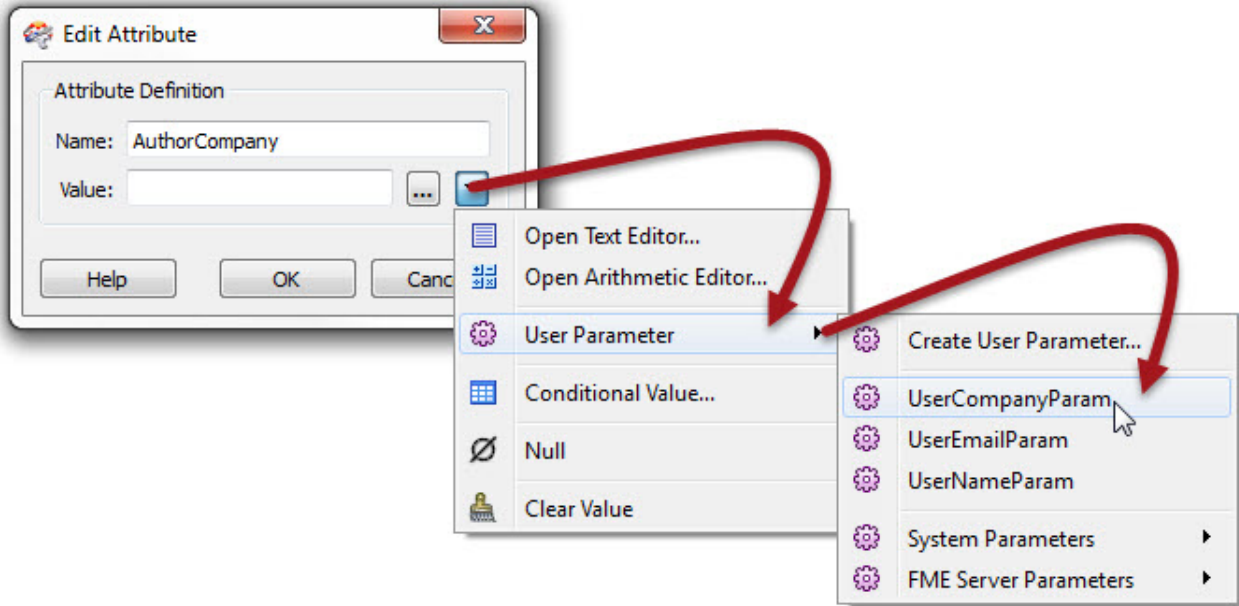
Click OK to close the dialog and the feature type should look like this. Notice how the attribute that has had its value set is now highlighted with a specific icon:
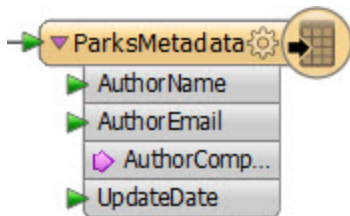
8) Save and Run Workspace

Save the workspace and then – as if you were the end-user – run it using Run > Prompt and Run.

When prompted enter your details into the fields that have been newly created:
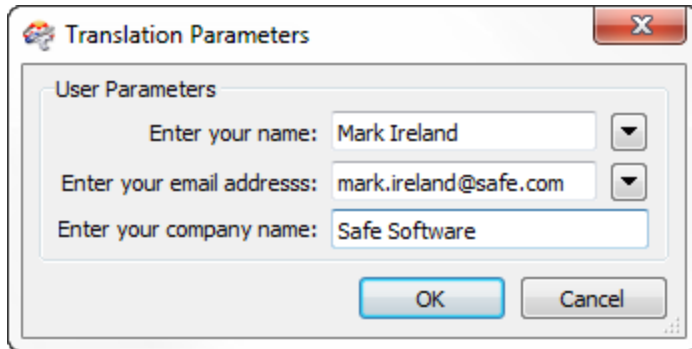
Locate and open the XML file to ensure the contents have been inserted as expected:

```
<fme:AuthorName>Mark Ireland</fme:AuthorName>
<fme:AuthorEmail>mark.ireland@safe.com</fme:AuthorEmail>
<fme:AuthorCompany>Safe Software</fme:AuthorCompany>
<fme:UpdateDate>2014-05-12T14:07:58</fme:UpdateDate>
```

More User Parameters

More User Parameters

Types of User Parameter
Linking User-FME Parameters
Creating Direct Links

FME Desktop Advanced Training 2015

There are many different types of user parameters and many different ways to make use of them.

Types of User Parameters

One of the key parameters when creating a user parameter is the parameter type. There are many different types, of which the more common ones are:

- Choice
- Choice with Alias
- Float
- Integer
- Text and Text (Multiline)

Choice and Choice with Alias

A choice parameter is when the user is presented with a fixed list of options and selects one of them. Here is a similar example to before. The user is being asked to enter 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:
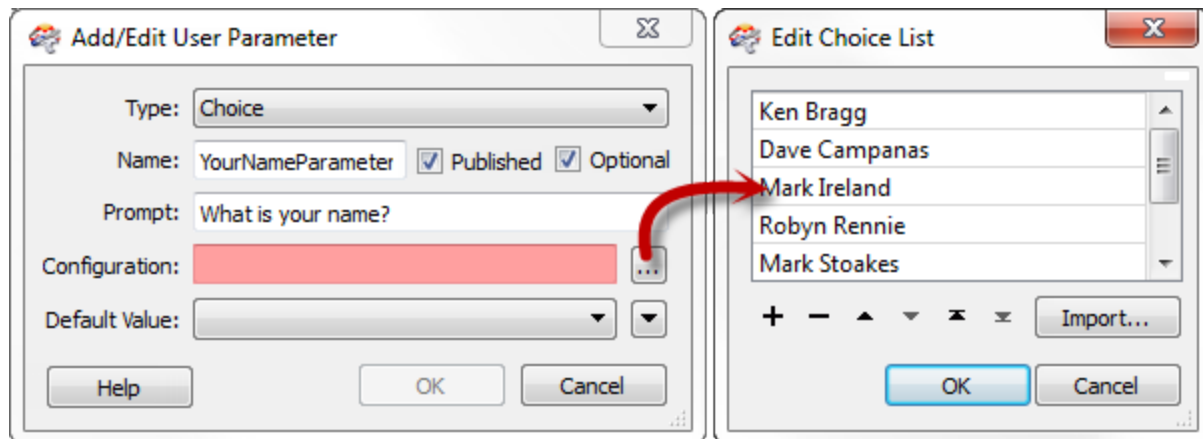
That way, the user is prompted to select their name from a list. They don't have to type it in manually.
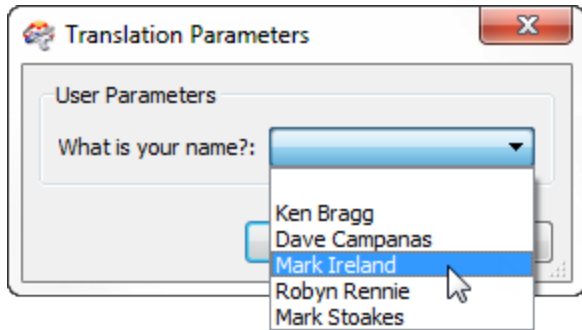
A Choice with Alias parameter is the same as a Choice parameter in that the end-user gets to pick a value from a list. However, a lookup table maps the chosen entry to a value that gets provided to FME.

For example, this workspace takes incoming features and matches them to a database using an EmployeeID.
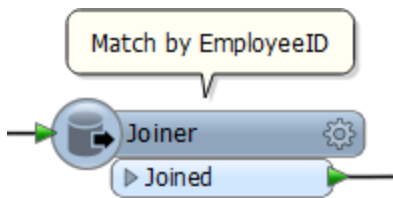
EmployeeID is provided by the end-user, but they can't always remember their own ID number. So the author creates a Choice with Alias user parameter.

The parameter is configured like so:
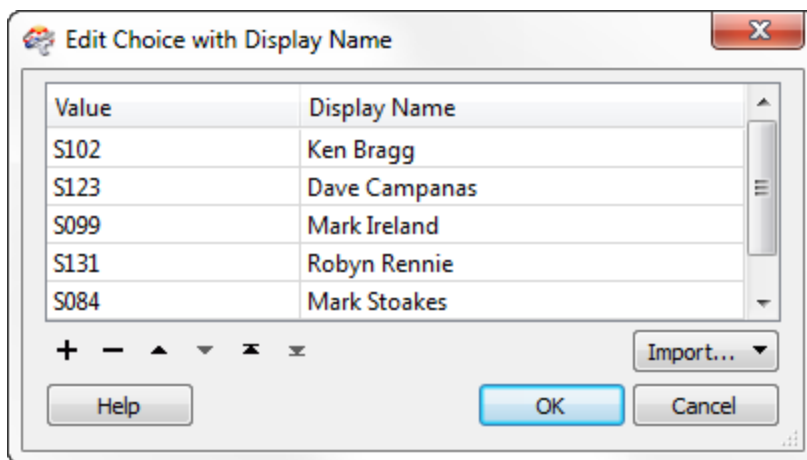
Notice that there are two fields in this configuration dialog; the display name and the actual value.
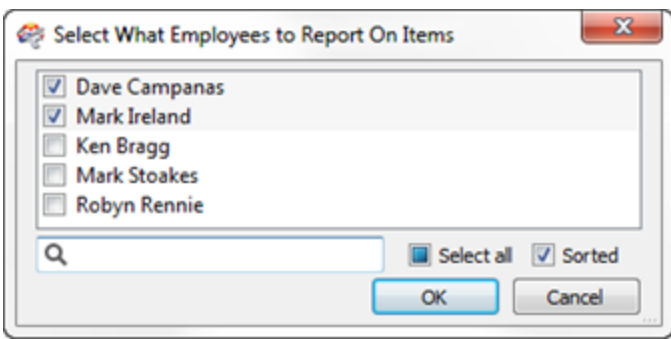
When a user selects their name from the list, then the value provided to the workspace is actually their employee ID. That way employee ID can be used as a match in the Joiner, without the end-user having to remember it!

Ms. Analyst says…

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

Select What Employees to Report On Items

☑ Dave Campanas
☑ Mark Ireland
☐ Ken Bragg
☐ Mark Stoakes
☐ Robyn Rennie

🔍   ☐ Select all   ☑ Sorted
OK   Cancel

Multiple values are returned space-delimited."

Float and Integer Parameters

Float and Integer parameters – as their names suggest – are simply ways for a user to enter a floating point number or an integer number.

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

User Parameters
Enter a Number: 123.4a
Enter an Integer: 1.4
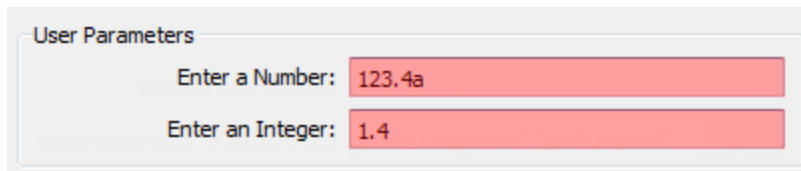
This image shows how non-numeric characters in either type, or a decimal point in the integer type, will be detected and rejected with a red-colored field.

Text Parameters

Text parameters are a simple way to accept plain text values into a workspace.

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:
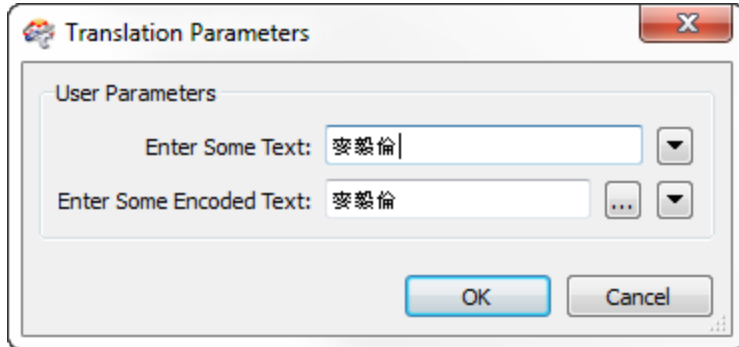
That way FME will retain and use the actual value, here shown in the Data Inspector:

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."

Linking User-FME Parameters

It's not just the case that a workspace author will want to apply user input to an attribute value. In some cases the author will actually want to give the end-user control over an FME parameter.

In this scenario the author will create a user parameter, and then link it to the FME parameter.
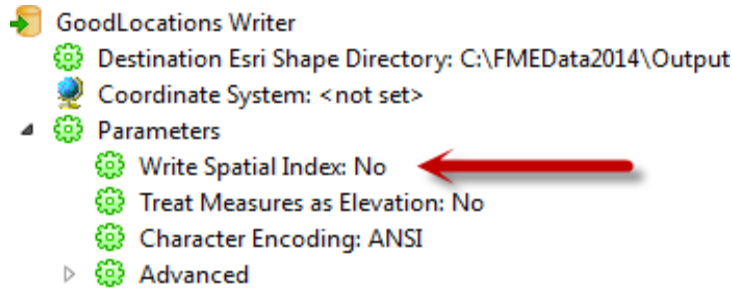
For example, here the author has a workspace that writes Shape data:

They wish to give the end-user control over whether a spatial index is created or not

They already have a user parameter defined called WriteSpatialIndex, with choices Yes and No, however the user parameter does not do anything yet. It must be linked to the FME parameter.

The author can do this by either right-clicking the FME parameter and choosing Link to User Parameter, or they can right-click the User parameter and choose Apply To [FME Parameter].
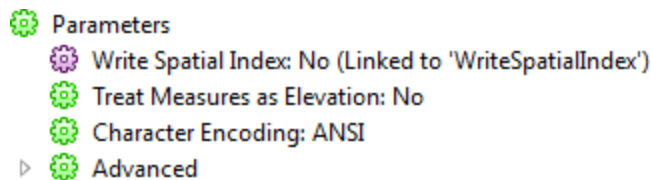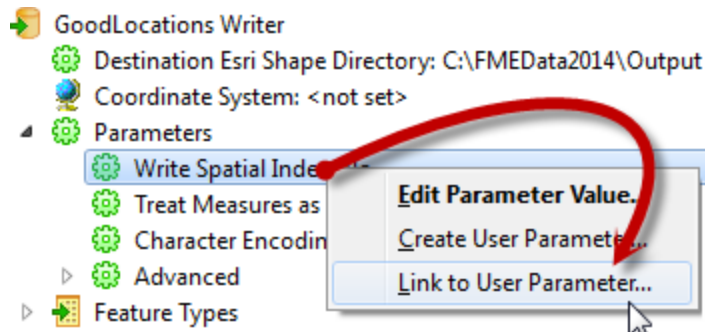
Now the FME parameter is linked to the User Parameter, so whatever the user chooses will be applied directly to Write Spatial Index

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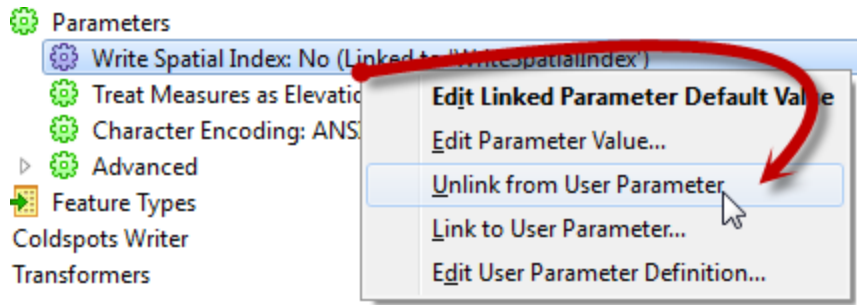
Creating Direct Links

In the previous example, a user parameter was created separately and then linked to the FME parameter. However, it is possible to both create and link a parameter simultaneously.
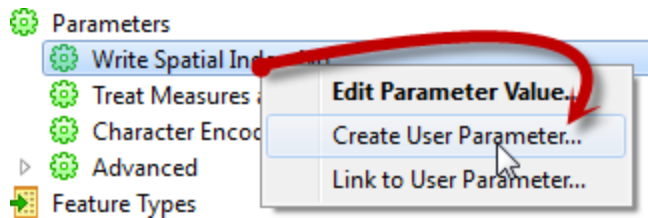
In the Navigator window, simply right-click an existing FME parameter and choose the option to "Create User Parameter".
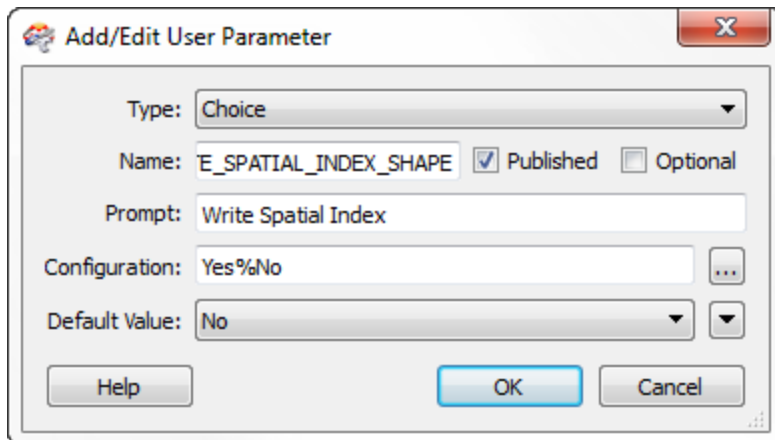
This opens a dialog and automatically fills in a definition to create a new user parameter.

Click OK and the user parameter is created and automatically linked to the FME parameter.

The same can be done from within a transformer dialog, like so:
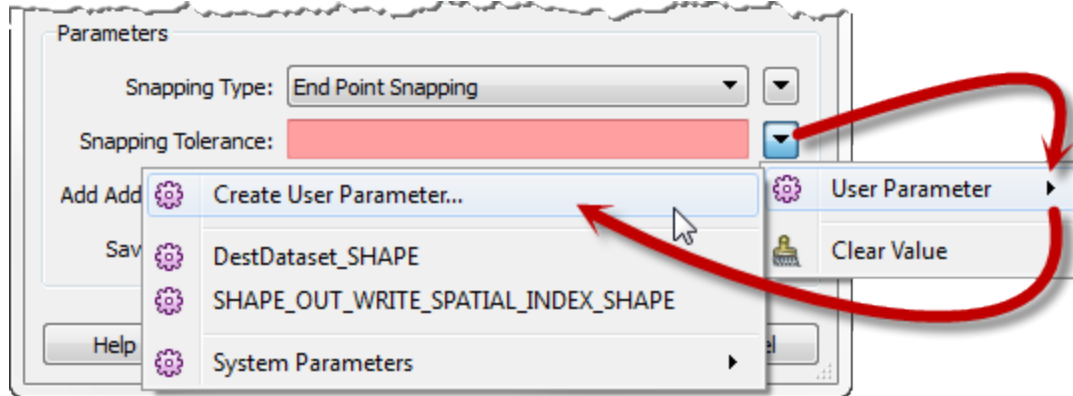
Here the workspace author is creating a user parameter linked to the Snapping Tolerance FME parameter in a Snapper transformer.

Advantages and Disadvantages of Direct Links

Creating a linked FME parameter directly like this has some obvious advantages, and perhaps some not-so-obvious disadvantages.

The most obvious advantage is that this is a single step process. The creation and linking of the user parameter is done in a single action.

Additionally, the parameter typing is taken care of automatically. If the FME parameter requires an integer value then a user parameter created directly from it will be automatically defined with an integer parameter. There's no possibility of getting the wrong parameter type.

For example, the Snapper parameter in the previous screenshot, allows a floating point number, therefore it will create a user parameter of type float. No choice is provided; it will be a float automatically:
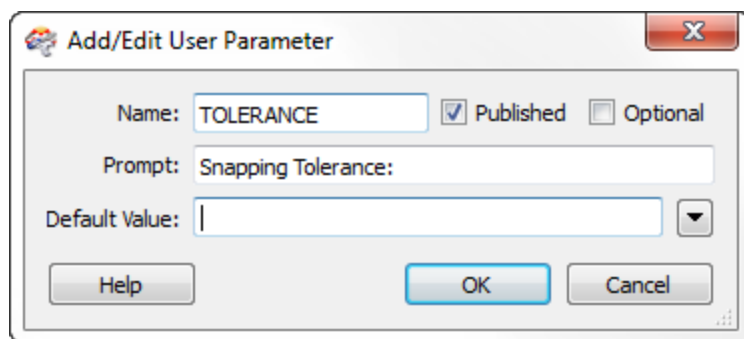
However, this also becomes a limitation too. Say, for example, the author wanted to provide a list of permitted tolerances; 0.5, 1.0, 5.0, etc. In that scenario they would have to create the user parameter separately – as type Choice – and then link it to the FME parameter manually.

Of course, the author would need to take care that the values provided by the user parameter were of a type that matched those expected by the FME parameter.

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

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, of course, could be seen as a disadvantage, depending on whether you would like this behavior or not.

Here a user parameter was created automatically from a Snapper transformer parameter. If the Snapper is deleted, then the user parameter is deleted too:
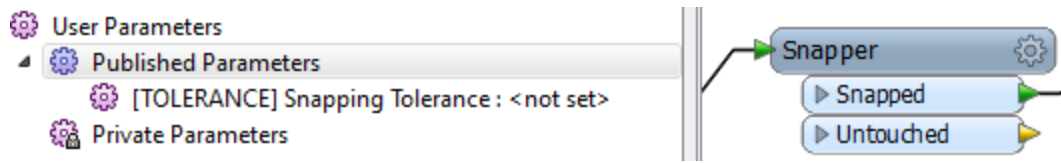
Pre-linked Parameters

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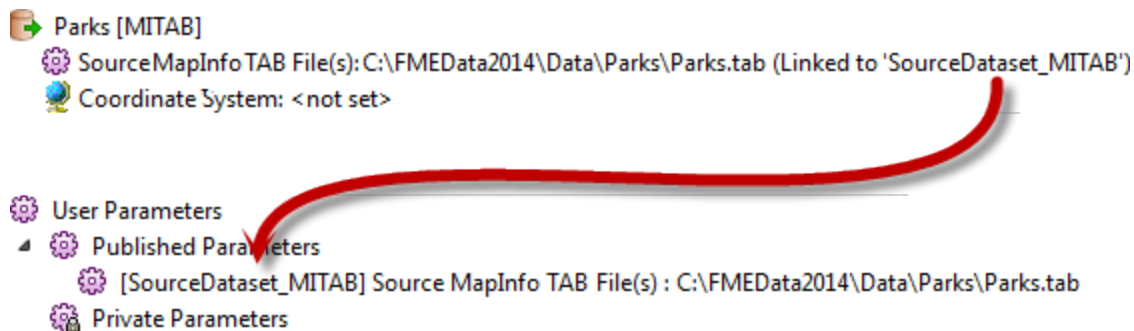
This automatically occurs for parameters that are important to the end-user and that appear in nearly all workspaces.

The same thing happens to the Feature Types to Read parameter in any Reader that is added in dynamic mode:

⚙ Advanced
    ⚙ Network Authentication: <not set>
    ⚙ Network Proxy: <not set>
    ⚙ Start Feature: <not set>
    ⚙ Max Features to Read: <not set>
    ⚙ Min Features to Read: <not set>
    ⚙ Feature Types to Read: (Linked to 'FEATURE_TYPES')

Ms. Analyst says…

"If you – as the workspace author – don't want or require the end-user to have access to these FME parameters, then you can either delete the user parameter, or just unlink it from the FME parameter."

Exercise 1b Simplifying Workspaces - Part 1	
Scenario	FME author; City of Interopolis
Data	Community Map (File Geodatabase)
Overall Goal	Simplify workspaces for inexperienced users
Demonstrates	Use of complex User Parameters.
Starting Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise1b-Begin.fmw*
Finished Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise1b-Complete.fmw*

Colleagues have purchased FME to do some data translations. However, not having taken the training course they are not confident users and would like to have their workspace made as simple as possible for them to run.

So, your task is to simplify their workspace using User Parameters, so the workspace can be run with minimal user intervention.

1) Start Workbench

Open the workspace C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise1b-Begin.fmw.

This is the workspace created by your colleagues:
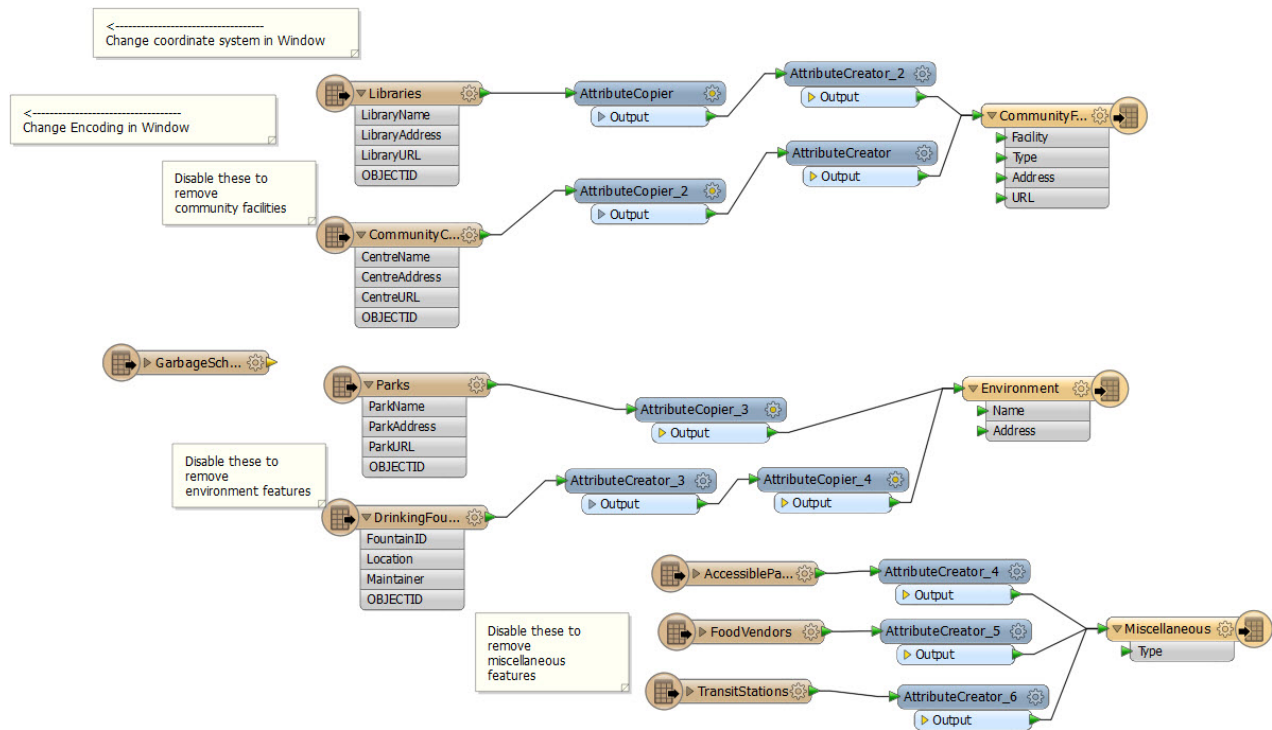
Notice that source tables are being grouped together and written to a Shape feature type. Their current method is to choose which tables to process by disabling them in the workspace. Similarly, they are setting the destination coordinate system and data encoding using Navigator parameters.

Also notice that the only annotations in the workspace are there to help the end user make edits to the workspace. 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.

[SourceDataset_FILEGDB] Source Geodatabase : C:\FMEData2015\Data\CommunityMapping\CommunityMap.gdb
[DestDataset_SHAPE] Destination Esri Shape Folder : C:\FMEData2015\Output\Training

However, you know that the source data will never change, so that parameter is of no use. So, delete the user parameter labelled "SourceDataset_FILEGDB."

Keep the parameter for DestDataset_SHAPE, as that can still be used to set the output location.

3) Create Encoding Parameter

One facet of the output that the users would like to control is the encoding of the dataset. This is achieved through a Writer parameter. However, it would be better for the users to set this as a user parameter.

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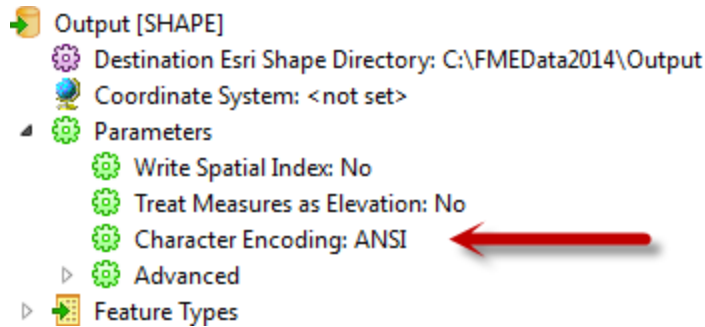
Identify the Character Encoding parameter, right-click on it and choose Create User Parameter:

Simply click OK on the dialog that opens and a user parameter is created and linked to the FME one.

4) Create Coordinate System Parameter

Another requirement is an ability to set the output coordinate system.

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.

It would be preferable if the parameter only allowed the end-user to select a coordinate system from a smaller list. For example, since the data is located in Vancouver, BC, it makes little sense for the user to be able to reproject it to NZMG (a New Zealand coordinate system).

So, locate the User Parameters in the Navigator Window, right-click and choose Add Parameter.

Set the Type to be Choice with Alias; set the Name to be CoordSysParam, and set the prompt to be Select Output Coordinate System:
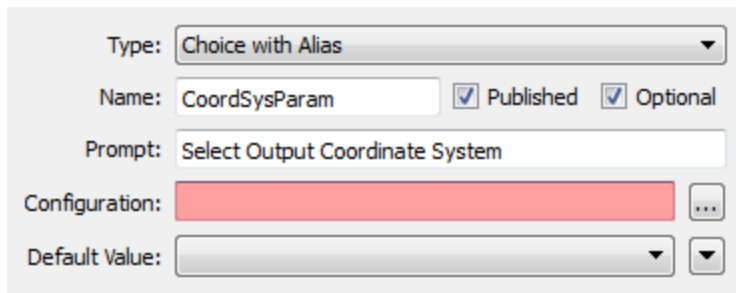
Now click the [...] button to the right of the Configuration setting. This opens a dialog in which to configure the parameter.

Normally we would enter values manually. However, 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):
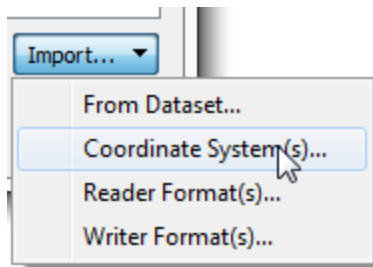
This opens a list of coordinate systems that we wish to import as values in our user parameter.

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.

Value	Display Name
BCALB-83	Albers Equal Area, Province of British Columbi...
CANBC-LCC	British Columbia; Lambert Conic projection, B...
LL83	NAD83 datum, Latitude-Longitude; Degrees [E...
UTM83-10	UTM with NAD83 datum, Zone 10, Meter; Cent...

+ − ▲ ▼ ⊼ ⊻   Import... ▼

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:

Output [SHAPE]
  Destination Esri Shape Directory: C:\FMED...2014...tput (Link...
  Coordinate System: <not set...
  Parameters
    Write Spatial Index: N   **Edit Parameter Valu...**
    Treat Measures as Ele| Create User Param...er...
    Character Encoding: A...   Link to User Parameter...   DI...
    Advanced
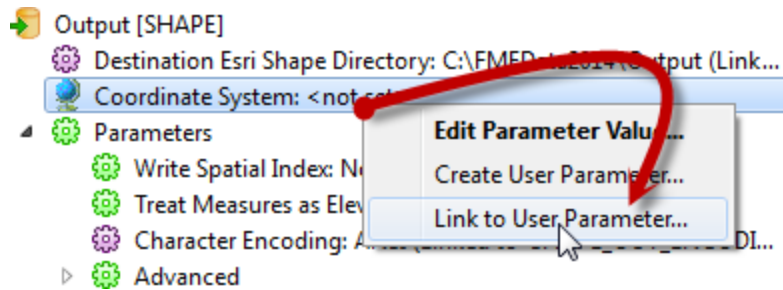
When prompted, select the newly created CoordSysParam and click OK to accept the selection.

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.) based on 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.

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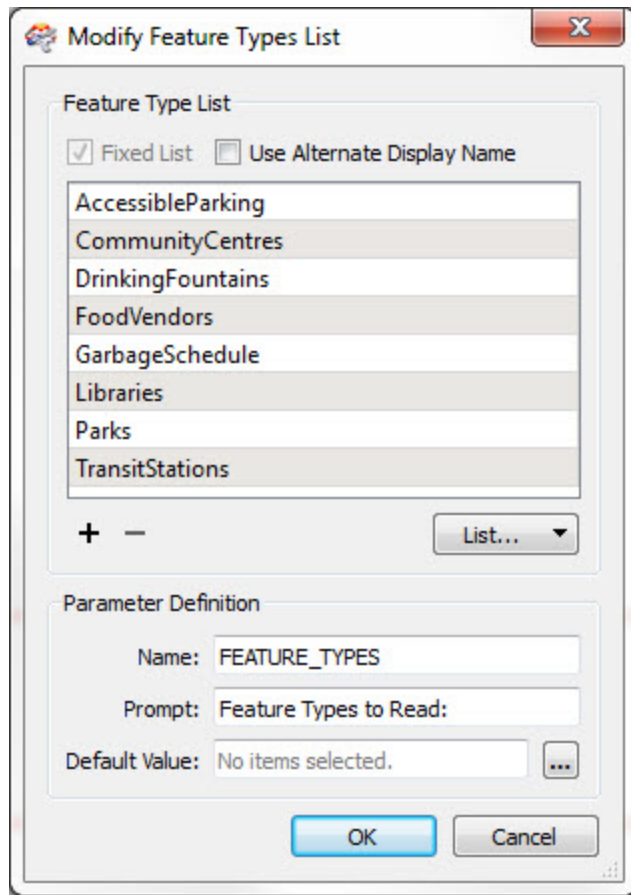
Check the box that is labelled Use Alternate Display Name. This provides the ability to give alternate names for each feature type. What we need to do is use this dialog to group together common Reader feature types under a single display name.

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:

Miscellaneous	AccessibleParking
Community Facilities	CommunityCentres
Environment	DrinkingFountains
Miscellaneous	FoodVendors
Community Facilities	Libraries
Environment	Parks
Miscellaneous	TransitStations
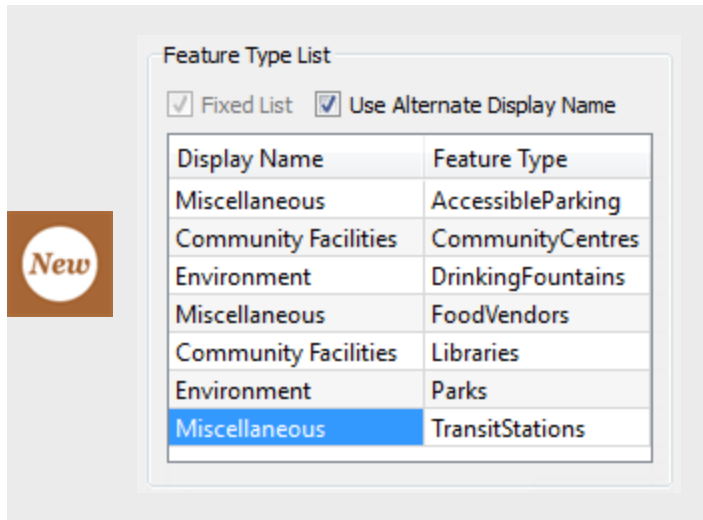
Underneath that change the prompt to read "Tables to Write" and then click OK to close the dialog.
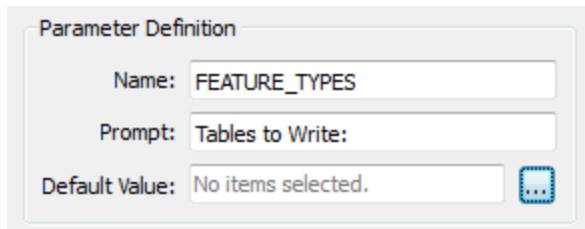
7) Save and Run Workspace

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:
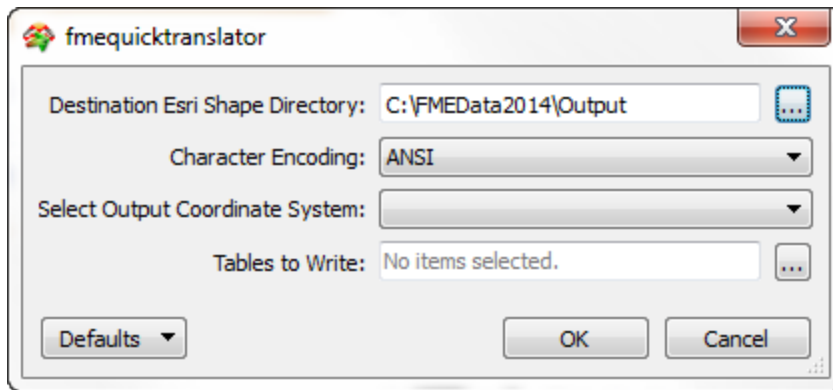
Pick Unicode 8-bit (utf-8) as the encoding. Select a coordinate system, noting how the user is restricted to those chosen by us. Select one or two of the tables to write and click OK to run the workspace.

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

NB: *If you run the workspace multiple times you will get multiple sets of results in the same folder! So Best Practice suggests you empty the output folder each time you run the workspace, but you can also find the latest results by checking the file dates/times.*

Advanced Task: *Speaking of Best Practice, don't forget to tidy up the workspace*

and give it a better style and structure.

Even More User Parameters

Even More User Parameters

Shared Parameters
Published and Private User Parameters
Embedded Parameters
Scripted Parameters
Attribute Name Parameters

Experience...

CONNECT.
TRANSFORM.
AUTOMATE.

2015.0

FME Desktop Advanced Training 2015

User parameters are very flexible and can be used in a variety of ways and scenarios

Shared Parameters

When a user parameter is used – either by an attribute or by being linked to an FME parameter – there's no reason why it should only be used once. The value obtained from a user parameter can be used as many times as is required.

In this scenario it can be described as a shared parameter.

For example, here the tolerance in a Snapper transformer is being set by a user parameter called TOLERANCE. The value of that same parameter is being fetched into an attribute so it can be recorded in the output dataset:
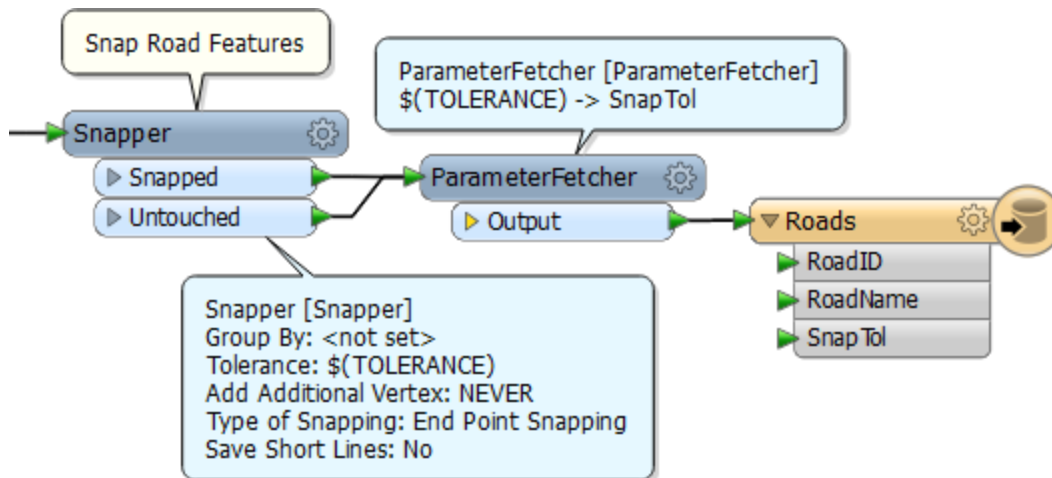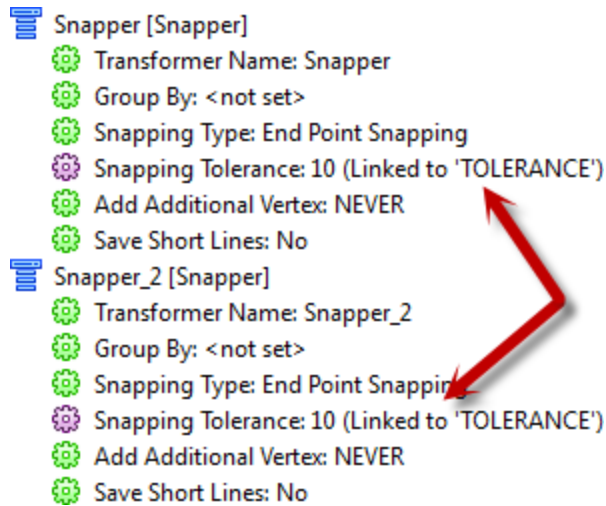
Similarly, if there are two (or more) Snapper transformers in the workspace, they all might use the same user parameter for tolerance.

The advantage in this is that the same value can be used without the user having to enter it multiple times.

Published and Private User Parameters

When a user parameter is created, one of the options is a checkbox labelled "Published":
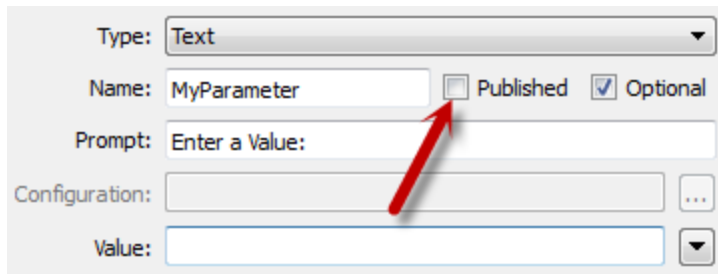
The purpose of this option is to expose or hide the parameter from the end user. If the Published box is checked, then the user will be prompted to enter a value. If the box is unchecked then they will not.
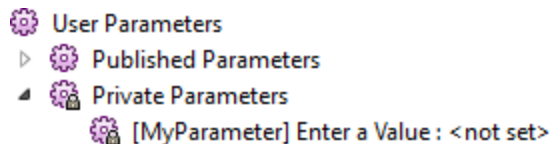
Such a parameter is then listed as Private:

Private Parameters have two uses. Firstly, they are a way for the workspace author to share a value without having it exposed to the user.

For example, if the Snapper transformers (in the previous screenshots) needed to get a common value – but that value is set by the author, not the user – then a private parameter can be used. It allows the author to set all Snapper tolerances to the same value, but without exposing that option to the end-user.

The other use of a private parameter is when a user parameter only provides partial input, and the true value has to be constructed. This is done as an Embedded Parameter.

> *Ms. Analyst says…*
>
> *"The Optional checkbox tells FME whether this user parameter is compulsory or optional. Turn off the checkbox – like the author should in the above example – when the input is a required field."*

Embedded Parameters

An embedded parameter is when the value of one user parameter is used inside another.

For example, here the user is allowed to enter a filename, but not the path the data is written to.

A published user parameter is created to accept a filename from the user. A private user parameter constructs the full name from that input [as C:\FMEData2015\Output\$(UserFileName)].

Roads [REALDWG]
    Destination Autodesk AutoCAD RealDWG File: C:\FMEData2014\Output\ (Linked to 'RealFileName')
    Coordinate System: <not set>
   ▷ Parameters
    Feature Types
Transformers
Bookmarks
User Parameters
  ◢ Published Parameters
    [UserFileName] Enter a filename : <not set>
  ◢ Private Parameters
    [RealFileName] Real File Name : C:\FMEData2014\Output\$(UserFileName)
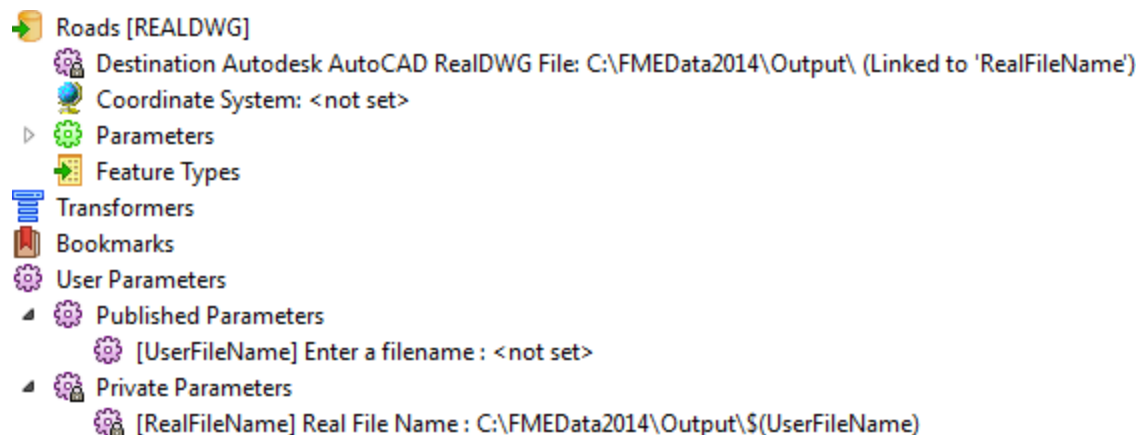
Finally, the private parameter is linked to the FME parameter for the destination dataset:

The RealFileName parameter is obviously kept private because the user doesn't need to know about it, much less set a value for it.

In this example the private parameter was created using a text editor. All user parameter creation dialogs have the ability to open a text editor. This is where simple concatenation of other parameters with fixed values can take place:
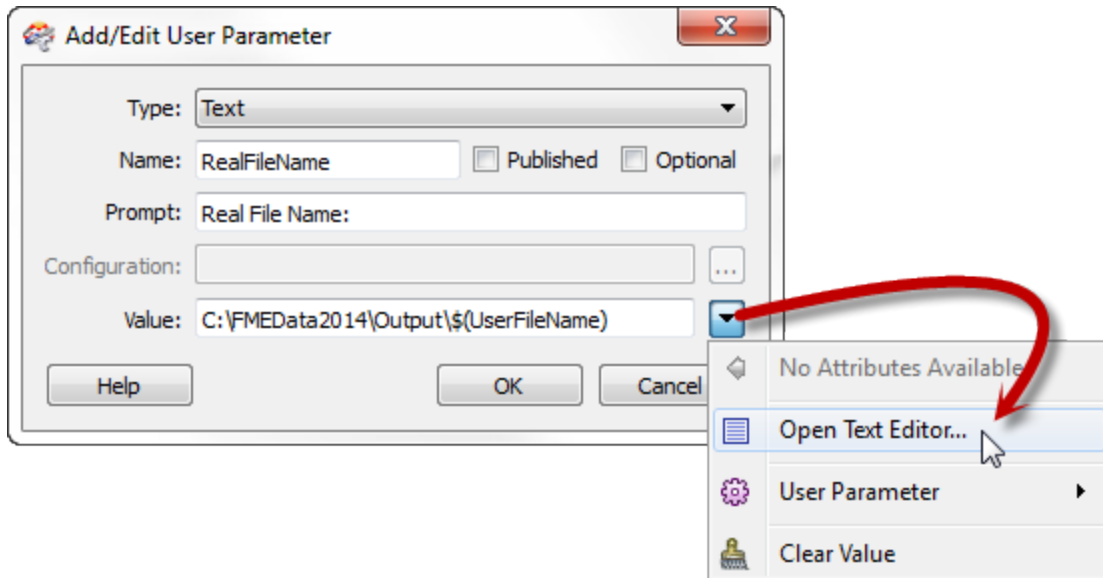
Scripted Parameters

A scripted parameter is a special type of user parameter.
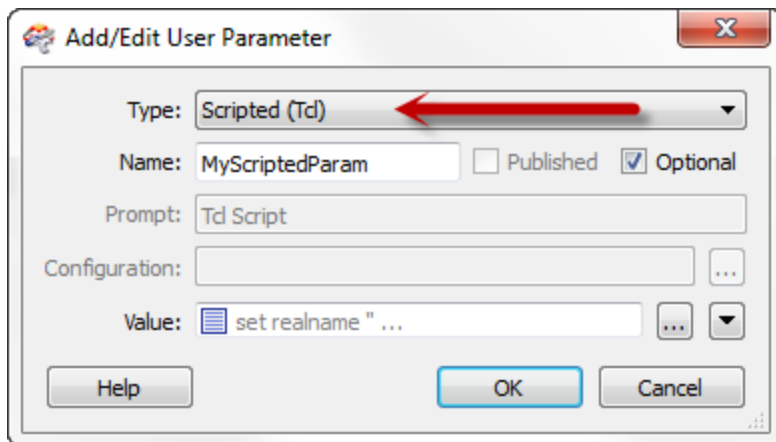
Scripted parameters go one step further than an embedded parameter.

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:

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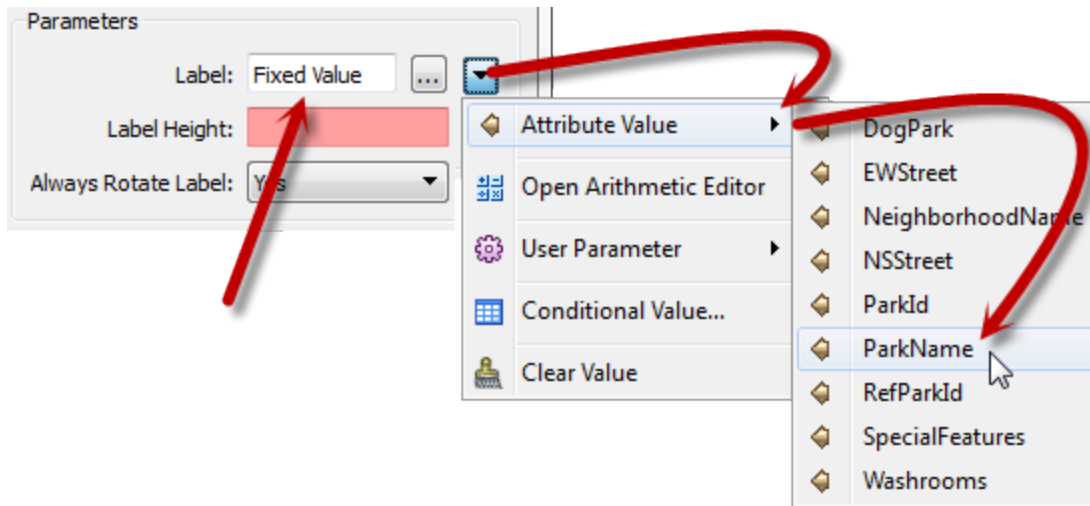
When the end-user is required to set this FME parameter, then it's simply a case of turning it into a linked user parameter. When the workspace is run, FME will scan the workspace to find what attributes are available to that transformer, and allow the user to select one or enter a fixed value.

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:
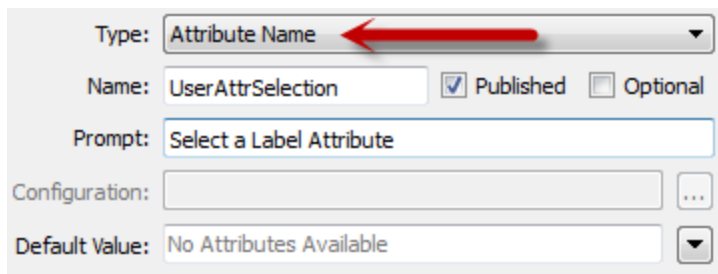
Now when the workspace is run, the user is permitted to select an attribute, and ONLY an attribute:
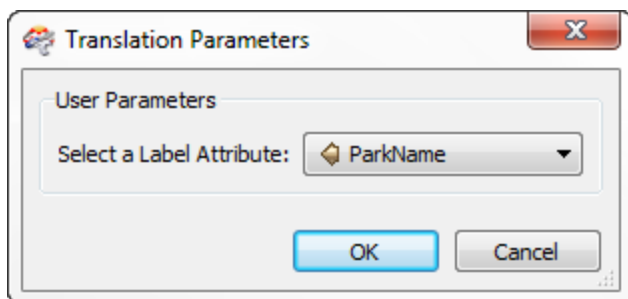
However, there is a catch to this operation. The user parameter – as the type suggests – is simply returning an attribute name.

ParkName

ParkName ParkName ParkName
ParkName
ParkName
ParkName ParkName
ParkName

ParkName ParkName
ParkName ParkName
ParkName
ParkName ParkName ParkName ParkName
ParkName ParkName

If the workspace is run then the LabelPointReplacer is supplied with the attribute name, and uses it as the label, like so:

What the author must do is open the dialog and change the label (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:

Parameters

Label: @Value($(UserAttrSelection)) [...] [▼]

Label Height: 100 [▼]

Always Rotate Label: Yes [▼] [▼]
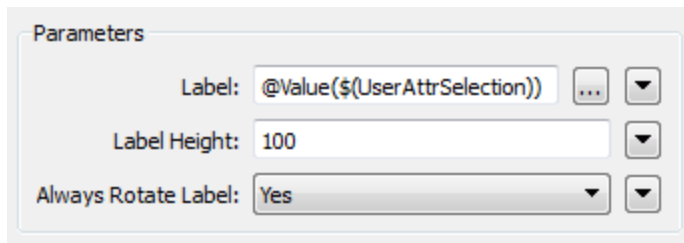
Now when the workspace is run the output will be correct:

CRAB Park at Portside
Wendy Poole Park

Oppenheimer Park
Pioneer Place (Pigeon Park)

Sun Yat-Sen Gardens
Andy Livingstone Park   MacLean Park
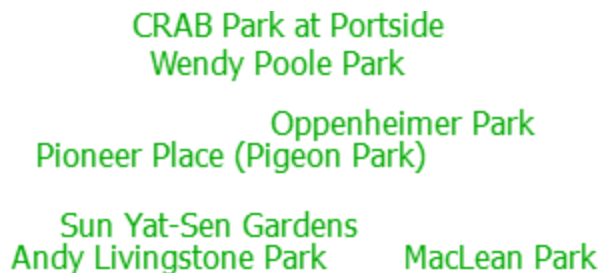
Ms. Analyst says…

"A set of system parameters exist for use specifically by FME Server. These are accessed by right-clicking on User Parameters in the Navigator window and choosing Expose FME Server Parameters."
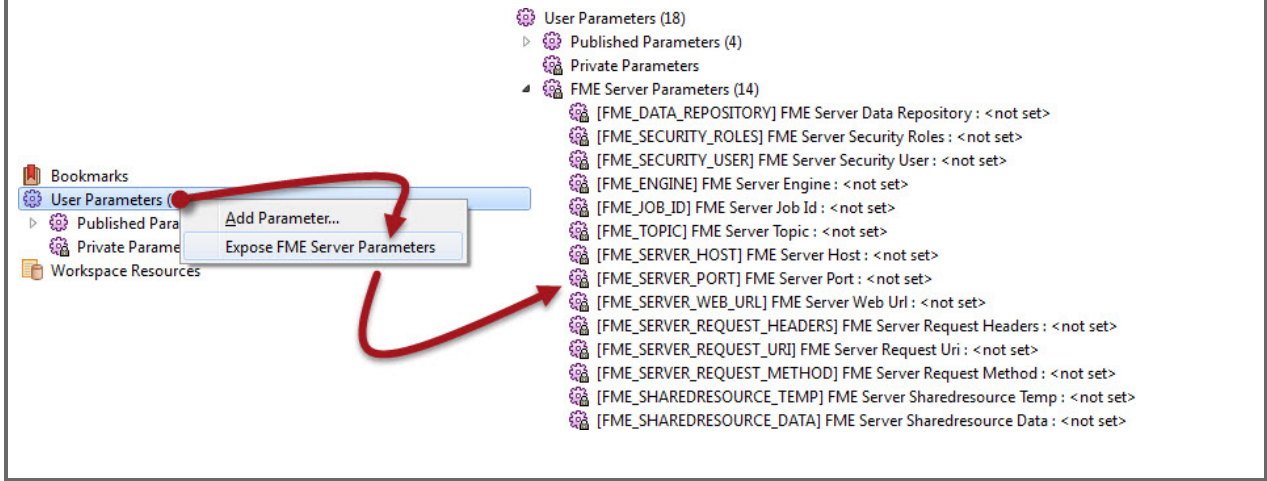
Exercise 1c Simplifying Workspace – 2	
Scenario	FME author; City of Interopolis
Data	Community Map (File Geodatabase)
Overall Goal	Simplify workspaces for inexperienced users
Demonstrates	Use of complex User Parameters.
Starting Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise1c-Begin.fmw*
Finished Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise1c-Complete.fmw*

The same colleagues from Exercise 1b are back with another request for help.

This time they have tried to experiment with a shared user parameter, but it is not working properly. Again, your task is to simplify their workspace and fix the User Parameters.

1) Start Workbench

Open the workspace *C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise1c-Begin.fmw*.

This is the workspace created by your colleagues:
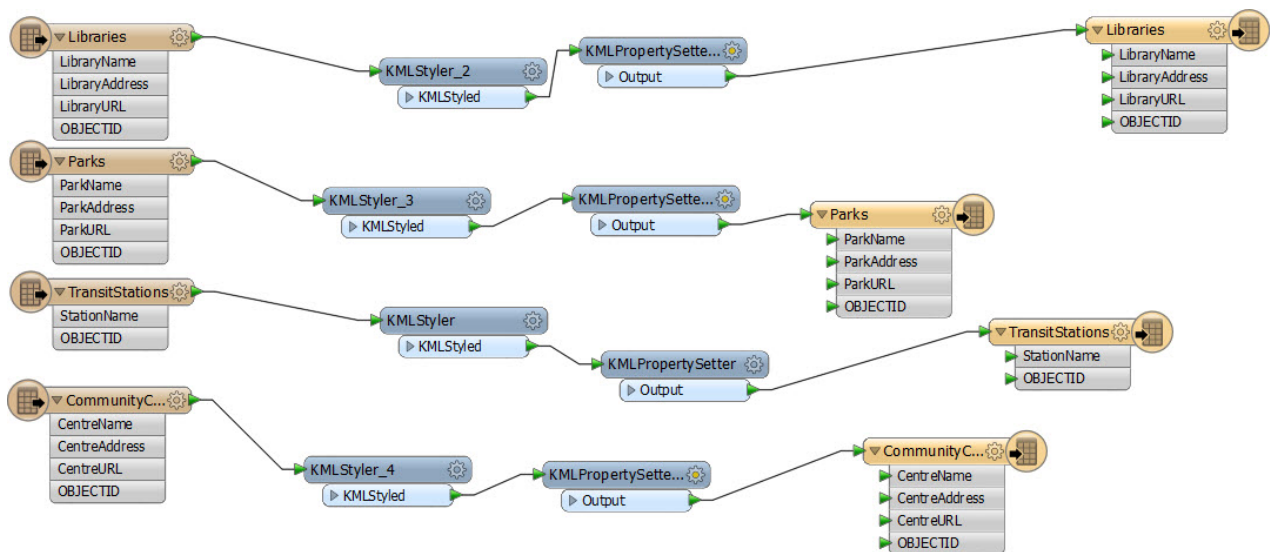
There are four tables being read from Geodatabase, and four layers being written to KML.

Open the KMLPropertySetter transformer parameter dialogs in turn. Note that each is using the same user parameter to set the KML name of the features:
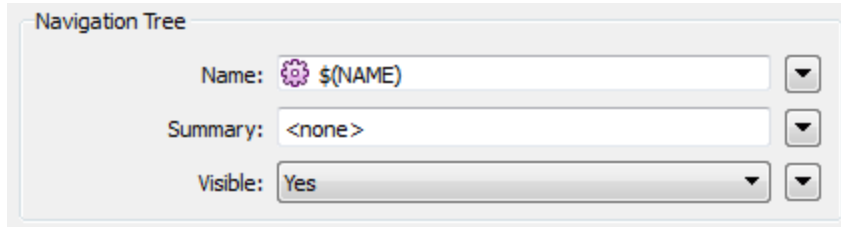
The KML name parameter defines a value that will be used as a label when the data is viewed in Google Earth.

2) Rename Attributes

The problem reported by your colleagues is that when they run the workspace they cannot select the right attributes for the KML name. They can only enter a hard-coded value (which they don't want to do) or select the attribute OBJECTID. No other attributes are available.

Run the workspace (using Prompt and Run) to confirm that this is the case.
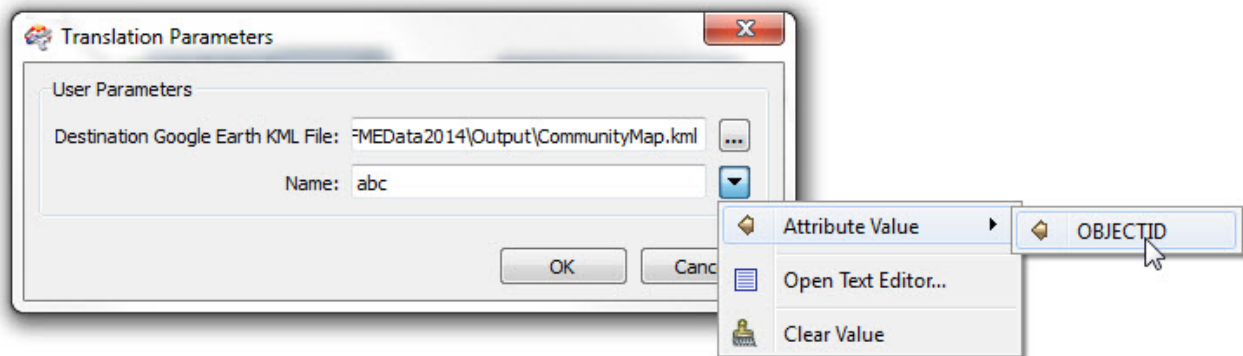
Can you see why this problem exists?

It is because - for a shared parameter - FME only lists attributes that are available on ALL instances of the user parameter. For example, LibraryName does not appear because it only exists on the library features. OBJECTID is the only attribute that exists at all four KMLPropertySetter transformers.

In other words, the end-user is only provided with the union of all available attributes. To get what the users want, their attributes will have to be renamed to a common name.

So, add AttributeRenamer transformers in each stream of data. For the library features use it to rename:

- LibraryName to Name

- LibraryAddress to Address

- LibraryURL to URL

Then use it to delete OBJECTID. You do this by simply setting it as an old attribute, and leaving out a new attribute name:
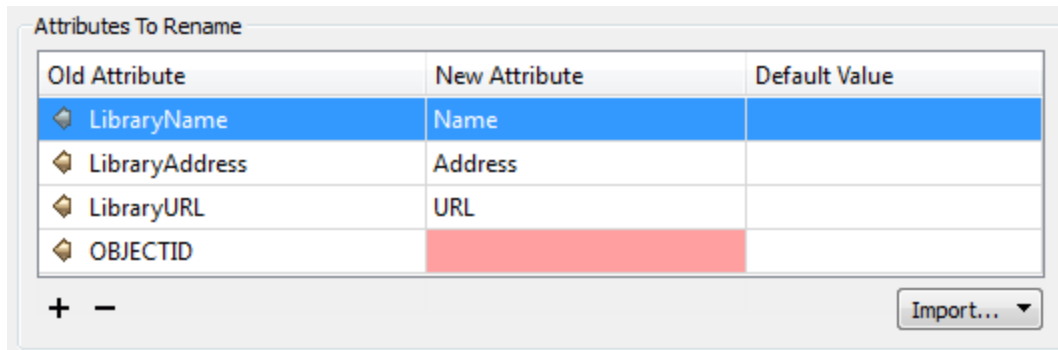
Now do the same for all other streams of data.

For parks:

- Rename ParkName to Name

- Rename ParkAddress to Address

- Rename ParkURL to URL

- Delete OBJECTID

For TransitStations:

- Rename StationName to Name

- Delete OBJECTID

- Add Address

- Add URL

For CommunityCentres:

- Rename CentreName to Name
- Rename CentreAddress to Address
- Rename CentreURL to URL
- Delete OBJECTID

Ms. Analyst says…

"While you're doing that, please do something about all those crooked lines! They're driving me crazy!"

The workspace will now look like this:
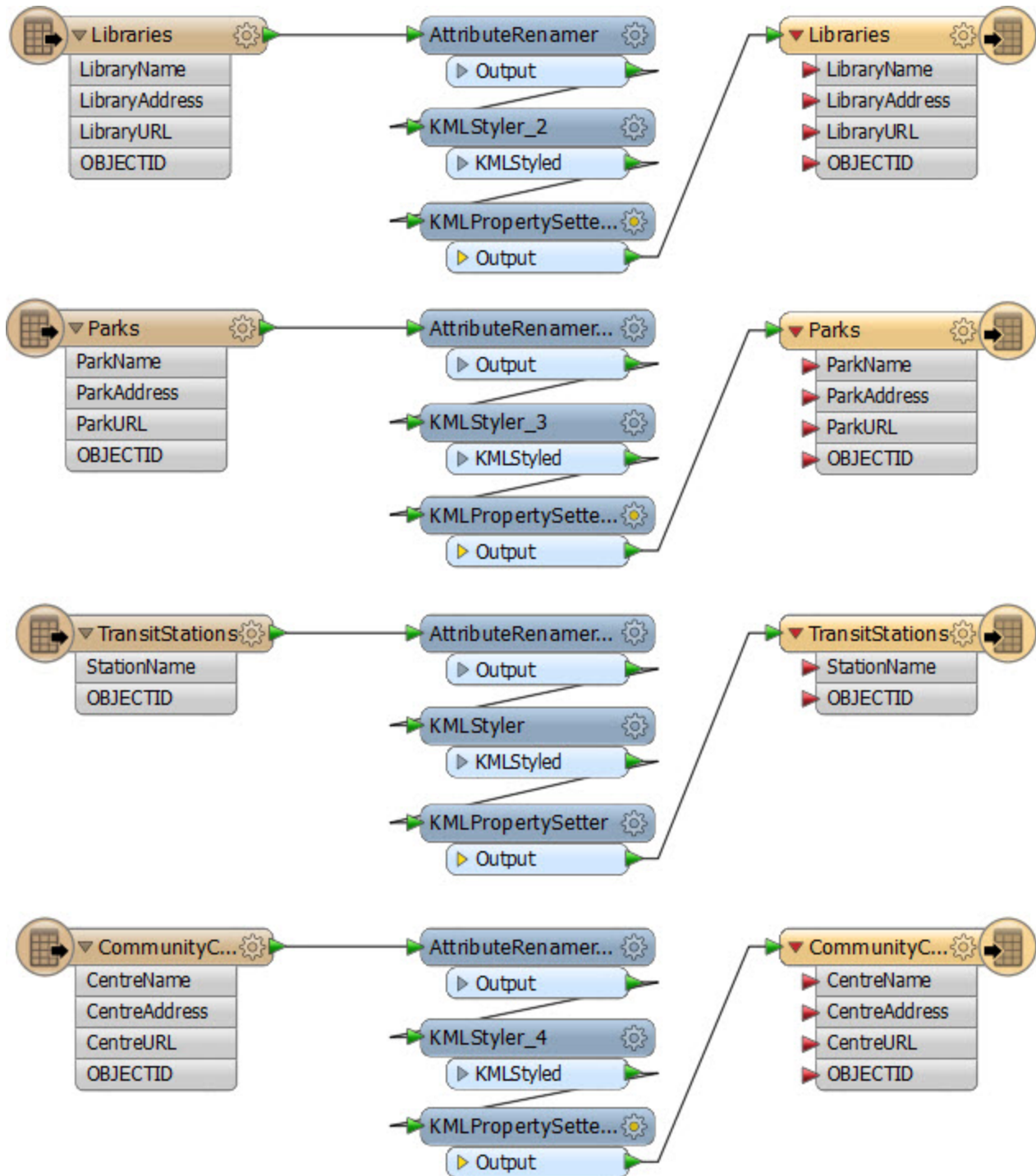
A consequence of this renaming is that you'll now need to re-map the attributes to the Writer schema. While doing so, delete OBJECTID from all Writer feature types, as they are not needed in this output.

Re-run the workspace and you'll find that it's possible to now select either Name, Address or URL as the name for the feature in the KML output. That's because all of these attributes exist at each location the "Name" user parameter is being used.

3) Add User Parameter

Two problems remain of which the first is that the KML name is an _OR_ATTR parameter. It can also be set as a plain text string, and this is not what your colleagues want. So let's add a new user parameter to solve that.

Create a new user parameter of type Attribute Name. Set the name and prompt according to what you think would be suitable; something like this:

Preferably turn off the optional checkbox, so the user is forced to select a value for this field.

4) Use User Parameter

Now we've created a new user parameter, let's make use of it.

Open up each KMLPropertySetter transformer's parameter dialog in turn. For each set the Name field to be: @Value($(LabelAttr)) – where LabelAttr is the name you gave to the new parameter.

NB: *It's easiest to set it in one transformer, and then copy/paste it to all the others.*

Notice that when you complete the last transformer, the previously used parameter – NAME – is automatically deleted from the Navigator window (you can use undo-redo to prove this is so). So we can tell that it must have been created by FME and linked automatically, not created by the author and then manually linked. If that was the case it wouldn't be deleted.

Now when the workspace is run the user can select an attribute, but not enter a value:
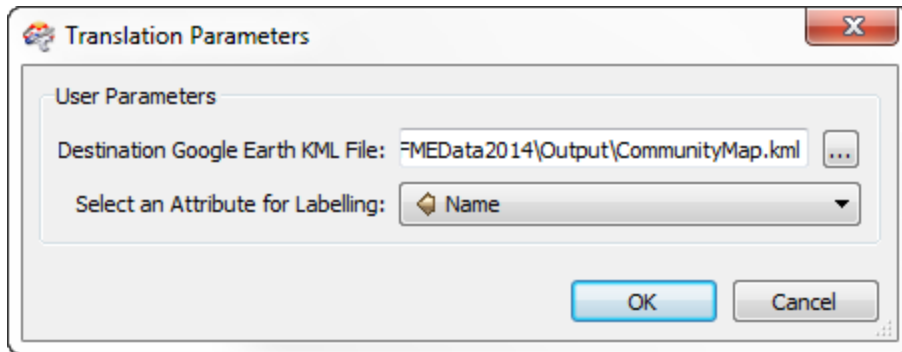
5) Create User Parameter

The other thing we could perhaps do is fix it so the user can enter an output filename, but not specify where the output folder will be.

So, create a user parameter of type text. Set the name and prompt according to what you think would be suitable; something like this:

You really need to turn off the optional checkbox here, as if there is no value, the workspace will fail.
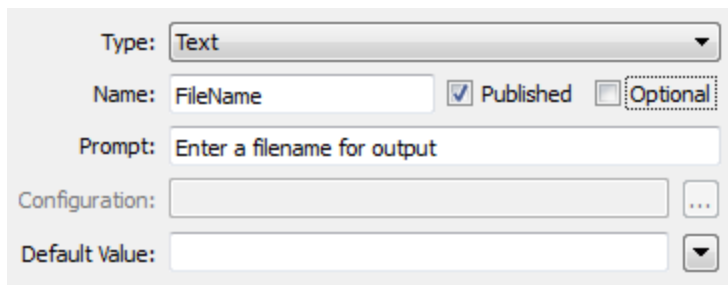
6) Create Embedded Parameter

Now create a second Text type user parameter. This time turn off both Published and Optional checkboxes. It is a parameter we need to use, but the end-user doesn't need to see.

The value for the parameter will be: C:\FMEData2015\Output\$(FileName).kml
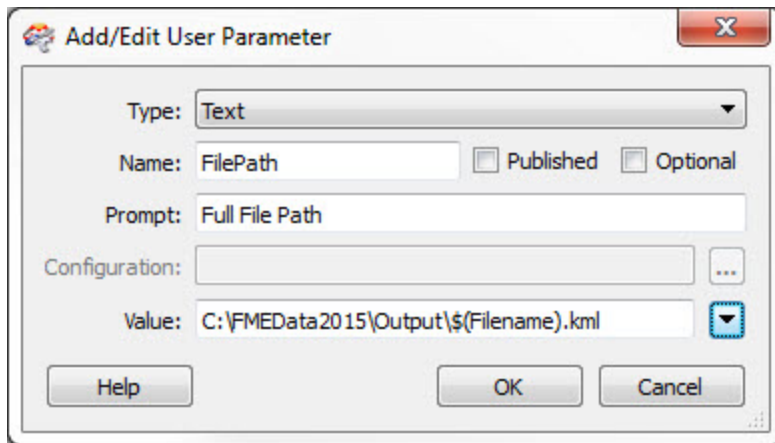
7) Link Embedded Parameter

As a final step, link this newly created parameter to the FME parameter for the output KML:
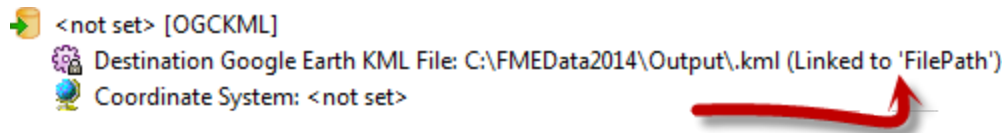
As you do so the existing user parameter – the one automatically created by FME – will be deleted automatically; again, this is because it is no longer used anywhere.

8) Save and Run Workspace

Save and then run the workspace to test it. Enter some parameters when prompted:

```
INFORM|KML: Initializing dataset `C:\FMEData2015\Output\MarksData.kml'...
```

The data will be written to the file specified and FME will label features with the required attribute:
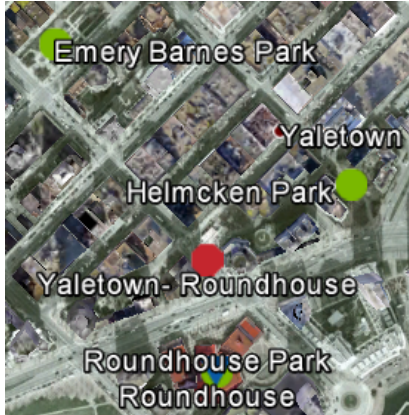
NB: *If you get the error message: "Undefined macro 'xxxx' dereferenced in file" then it's probable that the private parameter is wrongly defined. For example it is C:\FMEData2015\Output\$(Filename).kml when it should be C:\FMEData2015\Output\$(FileName).kml (notice the different case of the N in "FileName").*

Module Review

Module Review

What you should have learned from this module

FME Desktop Advanced Training 2015

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

Chapter 2 - 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.

In particular, some such factors are:

- 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 minimal licensing limits that 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…

"I recently read an article where the author suggested 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}, meaning it can use a maximum of 4 GB of memory. However, a 64-bit operating system can use memory addresses up to 2^{64}, which technically allows up to 16 billion GB of memory.

In practice, a 64-bit system uses nowhere near that much memory; still it allows a vast amount of additional memory which can be taken advantage of by 64-bit software.

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.

What are the Disadvantages of 64-bit?

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

Firstly it 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.

Secondly – and more importantly – 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.

Additionally, you need to take care to use the correct 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 (safe.com/formats) with info on which are supported on 64-bit."

Filter by DataType:	Windows 32-bit	Windows 64-bit	Linux 64-bit
Esri ArcSDE	R/W	R/W	R/W
Esri Geodatabase (ArcSDE Geodb)	R/W	R/W	—
Esri Geodatabase (File Geodb API)	R/W	R/W	R/W
Esri Geodatabase (File Geodb ArcObjects)	R/W	R/W	—
Esri Geodatabase (Personal Geodb)	R/W	—	—

FME and 64-Bit

Here are some helpful hints and tips on using 64-bit FME:

- If you are using 32-bit Windows then you'll need to use 32-bit FME. If you are using 64-bit Windows then you can use either 32-bit or 64-bit FME. Both will work, but only 64-bit FME will take advantage of any extra memory available to it.

- It is possible to install both 32-bit and 64-bit FME on the same 64-bit computer. That way you can use the 32-bit version to access all of the formats you might require, and use the 64-bit version for more intensive processing.

- It is also possible to install 32-bit and 64-bit FME engines on the same FME Server core. Job Routing techniques will allow you to designate which jobs should be processed by which engine.

- If you are running 32-bit FME on a 64-bit system then you would need the 32-bit client to connect to, for example, an Oracle database. There are similar requirements for reading Microsoft Office files. FME, with either client, can read and write to a 32-bit or 64-bit Oracle database, provided your license level is sufficient.

- There are many helpful articles on 64-bit on the FME knowledgebase, FMEpedia.

Log File Interpretation

If you can't understand what FME is doing then there isn't much chance you'll be able to improve upon it.

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.

Deconstructing a Log File

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

We'll look at the time fields later. The Message Type field 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.

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\FME2015\fme.exe C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise1c-Begin.fmw
     --DestDataset_OGCKML C:\FMEData2014\Output\CommunityMap.kml
     --SourceDataset_FILEGDB C:\FMEData2014\Data\CommunityMapping\CommunityMap.gdb
```

In terms of performance, this section doesn't tell us much. However, it is useful to confirm what FME version is running. Also, if you wanted to run the translation through the command-line, this is the statement that you'd use.

Configuration and Setup Information

This part of the log file tells 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.

```
FME 2015 Beta (20141202 - Build 15223 - WIN32)
FME_HOME is 'C:\apps\FME2015\'
FME Desktop Smallworld Edition (floating)
Permanent License.
Machine host name is: vash
```

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: 107.95 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)
```

In FME2015 measurements of disk and memory are now given in the most appropriate units, quite often GB and TB, rather than MB.

Then comes important information about the system resources and FME's memory management:

```
Process limit is 4.00 GB (out of 24.00 GB physical memory and 4.00 GB address space)
Start freeing memory when process usage exceeds 2.83 GB of memory or 3.41 GB of address space
Stop freeing memory when process usage is below 2.12 GB of memory and 2.56 GB of address space
Autodetermining optimal maximum number of objects in memory
```

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 limit is 24.00 GB (out of 24.00 GB physical memory and 8.00 TB address space)
Start freeing memory when process usage exceeds 8.00 TB of memory or 8.00 TB of address space
Stop freeing memory when process usage is below 6.00 TB of memory and 6.00 TB of address space
Autodetermining optimal maximum number of objects in memory
```

Translation and Transformation Statements

The main body of the log file 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, so it isn't always apparent which transformers will be processed in which order.

Secondly, FME works by pushing features through the workspace on an individual basis, not as a group. For example, if there are three transformers in a workspace, 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.

Translation Summary

The final part of a log file includes a report of the number of features read and written but, more important 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) (160 feature(s) output)
FME Session Duration: 0.7 seconds. (CPU: 0.4s user, 0.2s system)
END - ProcessID: 16584, peak process memory usage: 107100 kB,
                        current process memory usage: 107100 kB
```

Peak memory usage is an important statistic, as it is this number we'll want to reduce in order to improve FME performance.
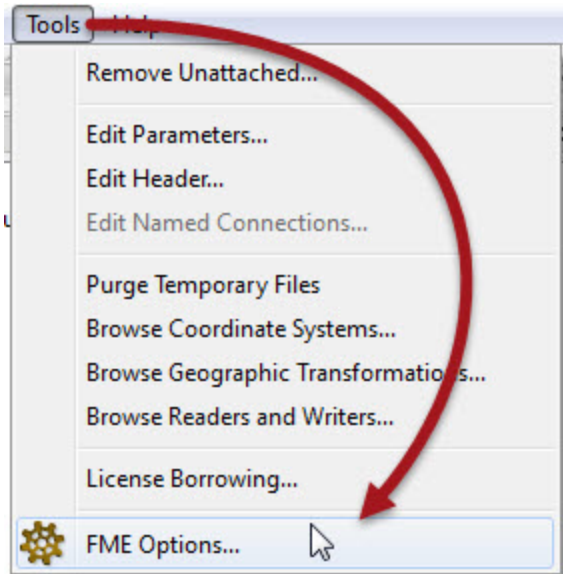
Configuring the Log Window

There are a number of options you can use to adjust the log file and what is displayed.

To do this, first Choose Tools > FME Options from the menubar.
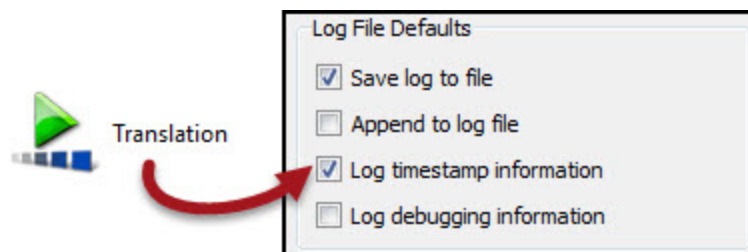
Then choose the Translation set of options.

To assess times logged in the log window, you first need to turn them on! In the options menu, ensure that 'Log timestamp information' is turned on.

NB: The log *file* always contains timestamps, regardless of this setting.

Now when a workspace is run, each message in the log gets stamped with the time and date it occurred.

```
2014-04-30 14:34:33| 3.7| 0.0|INFORM|Opened Shape File C:\ALotOfData.shp for output
2014-04-30 14:34:33| 3.7| 0.0|INFORM|Opened DBF file 'C:\ALotOfData.dbf' for output
```
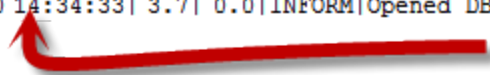
The Log debugging information option turns on a series of extra log messages that are usually hidden from the user. Not only will a lot of the underlying mapping file be exposed, there will also be a number of ERROR messages labelled BADNEWS.

These messages can help during debugging, but it's very unlikely you'll want to keep them turned on in general FME use. For instance, many of the BADNEWS messages are simple "errors" like an end-of-file message that FME has trapped and kept to itself. Unless you are looking for more information on an existing problem, these messages are likely to cause more confusion than clarity. Turn them off and de-clutter your log file.
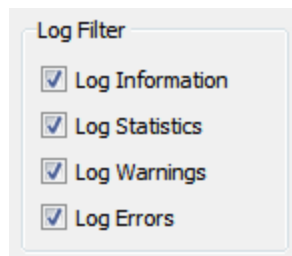
A final set of options allow the user to turn on or off each type of message in the log window. It can be particularly useful to turn off INFORM and STAT messages in order to make it easier to spot ERRORs and WARNs; however it does appear strange at first to run a translation and not see the usual stream of information!

Logs and Performance Indicators

Let's look at a few of the specific performance indicators we can identify in a log file and how we can improve upon them.

Log Timings

One interesting thing is that the cumulative time reported by a translation is the amount of time the FME process was actually translating or transforming data. It doesn't include the amount of time spent waiting for other processes to do their work.

For example, take this section of log timings:

```
2014-07-10 14:43:06| 8.5| 0.0|
2014-07-10 14:43:13| 8.8| 0.3|
2014-07-10 14:46:29| 18.0| 9.1|
2014-07-10 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, when a query is issued to a database, the time taken to respond with the results is not included in the FME processing time. The more a query is not well-formed or a database is not well-structured, the longer this time difference will be.

Similarly, the amount of time taken to read/write data from/to a disk is not included in this number.

Look at the section on Database Efficiency for more information on performance tuning FME for databases.

Default Paths

One of the most important paths to examine is the one for temporary folder. When memory resources become low and FME starts to cache to disk, the temporary folder is where it will write data to:

```
FME Configuration: Temporary folder is `C:\Users\imark\AppData\Local\Temp'
System Status: 107.95 GB of disk space available in the FME temporary folder
```

Firstly it's important to ensure this folder does have enough temporary disk space and 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.

Resource Manager

One key message to look out for is this:

```
  ResourceManager: Optimizing Memory Usage.  Please wait...
```

This means 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.

If you see this message frequently, then it's time to either redesign your translation or switch to a 64-bit setup.

Written Features

Maybe not quite performance-related, but 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, here 80 features are reported as sent to an (Esri Shape) Writer:

```
|=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
|                    Features Written Summary
|=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
|Parks_polygon                                             80
|=============================================================
|Total Features Written                                    80
|=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

However, a warning earlier in the log tells us:

`|WARN  |Rejected 80 output features`

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) as many as necessary 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.

Exercise 2a What Does the Log Say?	
Scenario	FME user; Interopocom, the Interopolis Telecoms company
Data	City Neighborhoods (KML) Cell Signals (CSV)
Overall Goal	Improve Workspace Performance
Demonstrates	Interpreting an FME log file
Starting Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise2a-Begin.fmw*
Finished Workspace	None

Here we have a prototype workspace that processes a dataset of cell phone signals. The dataset contains a series of recordings that show how strong the cell signal is at different locations.

The idea 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.

However, the workspace runs perhaps a little slower than it could, which is bad news when this is just the prototype and we wish to eventually run it on the entire country's cell data. 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:\FMEData2015\Workspaces\DesktopAdvanced\Exercise2a-Begin.fmw.
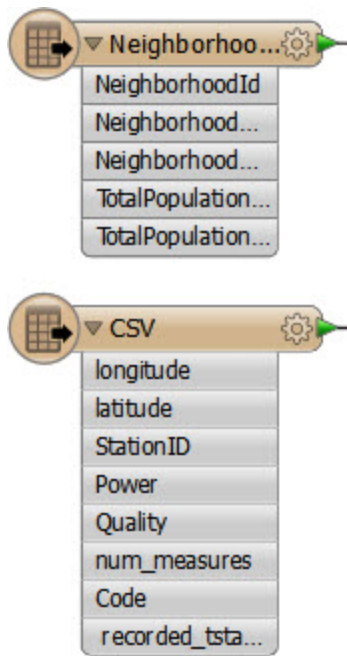
Notice there are two key feature types: one for the cell signals in CSV format, the other for the Neighborhood boundaries in KML format. They each have their own set of attributes:

There are also three transformers – two to attach the neighborhood and then filter the data – then two Writers, to write the data out to a final dataset. One has a few attributes, the other none.

The third transformer is a Logger that is recording features to the log.
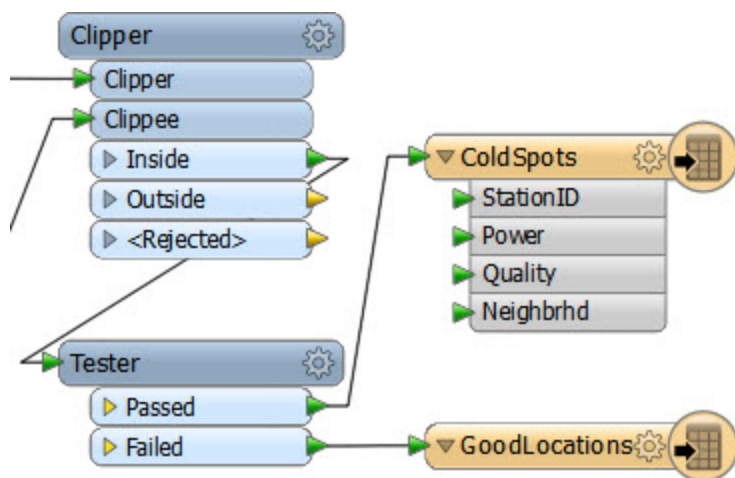
Don't run the workspace yet – we don't know how long it might take!!!

2) Open Log File

Now start up a text editor and open up the log file for inspection.

You can find it at: C:\FMEData\Workspaces\DesktopAdvanced\Exercise2a-OriginalLog.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 FME2015 (beta) with a fixed license, Smallworld Edition.

```
INFORM|FME 2015 Beta (20141202 - Build 15223 - WIN32)

INFORM|FME_HOME is 'C:\apps\FME2015\'

INFORM|FME Desktop Smallworld Edition (floating)

INFORM|Permanent License.
```

As a beta build there may be an advantage to upgrading to a newer FME version (although not as much as there might be if the user was using FME2011, perhaps).

Now look for the more important performance indicators:

```
INFORM|System Status: 105.46 GB of disk space available in the FME temporary
folder (C:\Users\imark\AppData\Local\Temp)

INFORM|System Status: 4.00 GB of virtual memory available

INFORM|Operating System: Microsoft Windows 7 64-bit Service Pack 1 (Build 7601)

INFORM|FME Platform: WIN32
```

And the resource manager parameters:

```
INFORM|FME Configuration: Process limit is 4.00 GB (out of 24.00 GB physical
memory and 4.00 GB address space)

INFORM|FME Configuration: Start freeing memory when process usage exceeds 2.83
GB of memory or 3.41 GB of address space

INFORM|FME Configuration: Stop freeing memory when process usage is below 2.12
GB of memory and 2.56 GB of address space

INFORM|FME Configuration: Autodetermining optimal maximum number of objects in
memory
```

There is plenty of disk space and adequate memory. We're on a 64-bit platform but only using 32-bit FME, which is a bit disappointing as there is 25 GB of memory, but we can perhaps let that go.

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:

```
INFORM|FME Session Duration: 4 minutes 1.0 seconds. (CPU: 185.7s user, 50.4s
system)

INFORM|END - ProcessID: 96656, peak process memory usage: 3065096 kB, current
process memory usage: 652096 kB
```

Two things are a worry there. The system time seems high (50.4 seconds) considering we're reading/writing local files and not a database.

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:

```
INFORM|Finished clipping 558250 / 872545 clippees against all clippers

INFORM|ResourceManager: Optimizing Memory Usage. Please wait...

INFORM|Finished clipping group 1 / 2

INFORM|Finished clipping 569200 / 872545 clippees against all clippers
```

So we can see that FME had to start optimizing memory usage. It probably resulted in some disk caching, and that might be the cause of the high system time.

3) Run Workspace

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. When you run this workspace, if you think your computer may be slower than average, you can reduce the amount of data being processing by changing the source dataset to "CellSignals2015-Lite.csv". Memory Optimization, in particular, may work differently depending on what other applications you have running at the same time.

Anyway, run the workspace. Obviously you can expect that it will take approximately 4 minutes to complete on a 32-bit machine. Do your results match what occurred in the above log file?

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.

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

```
INFORM|FME Session Duration: 2 minutes 59.3 seconds. (CPU: 161.8s user, 12.1s
system)

INFORM|END - ProcessID: 95448, peak process memory usage: 6202968 kB, current
process memory usage: 5287992 kB
```

Notice how the time used is about 1 minute less, and the amount of time used by the system (e.g. for reading and writing files) is reduced by about 90%

The amount of memory usage has increased, indicating that the 32-bit FME was having problems with an excess of data and had to resort to disk caching.

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!"

Reader and Writer Optimization

Reader and Writer Optimization

Assessing Reader Performance
Improving Reader Performance
Assessing Writer Performance
Improving Writer Performance

FME Desktop Advanced Training 2015

There are perhaps fewer methods to improve Reader and Writer performance, but they are no less important.

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| 0.0|INFORM|Emptying factory pipeline
```

But sometimes that number can be misleading. Take this workspace that reads a set of address points and finds their nearest neighbor in a second dataset:
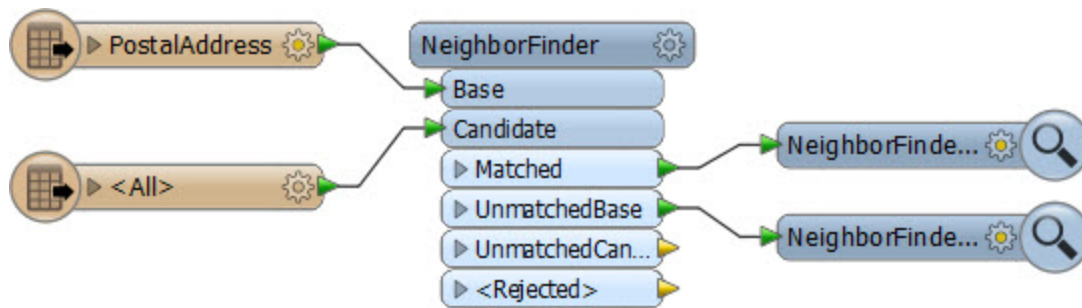
According to the log file the data took 27.5 seconds to read:

```
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.

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:
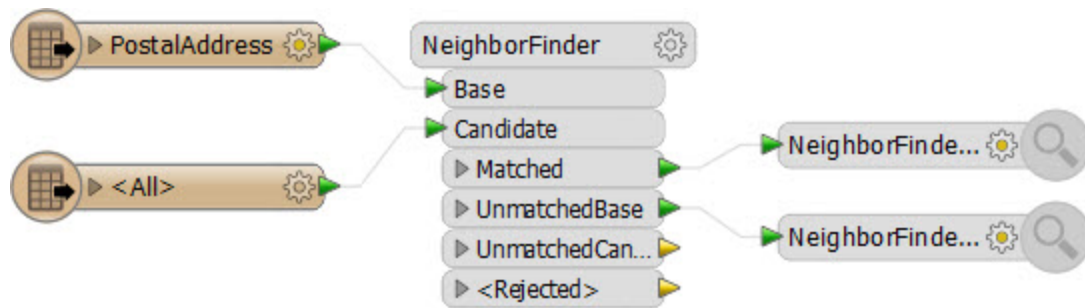
Now when the workspace is run it is reading the data only, with no transformation, and the factory pipeline message appears after a mere 5.4 seconds:

```
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 important to know to help interpret a log file and assess your Reader's performance, particularly since the log that appears while processing would appear to be still reading data:

For example, if your processing was very complex and taking hours to complete then you might mistakenly believe the data was still being read because "Reading source feature" messages were still appearing.

```
2014-12-08 13:24:54| 3.7| 2.1| INFORM| Reading source feature #5000

2014-12-08 13:24:58| 8.2| 4.4| INFORM| Reading source feature #5000

2014-12-08 13:25:03| 13.1| 4.9| INFORM| Reading source feature #10000
```

...when, in fact, data was simultaneously being processed.

Jake Speedie says…

"Not to confuse matters, but the exact structure of the log will depend greatly on whether the transformers being used are feature-based or group-based.

Remember, 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

One 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, here approximately 12% of the translation (0.4 seconds) was spent waiting for the system to return the requested data:

FME Session Duration: 3.1 seconds. (CPU: 2.6s user, 0.4s system)

Here the delay is not particularly problematic; but with larger amounts of data, and perhaps reading from a remote database server, the time taken could be more important.

The second obvious way to improve reading performance is to minimize the amount of data that is being read. For example, this workspace reads nearly 14,000 features, but immediately discards all except 132 of them:
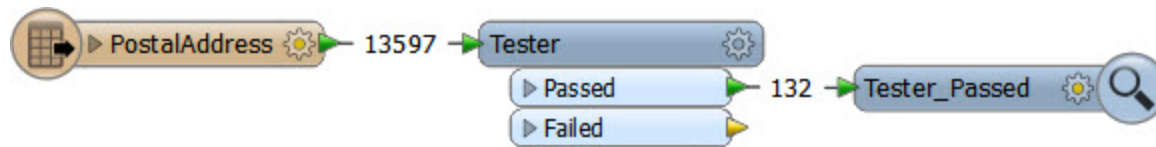
In this scenario, and where possible, it would be much more efficient to simply just read those three features.

> *Jake Speedie says…*
>
> *"How sensible is it, to go into a restaurant and order the entire menu, when you only intend to eat some of the dishes? Ordering is quicker, but way longer to be delivered, and it will certainly be more expensive!*
>
> *The same applies reading data with FME. 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)"*

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 – define the data to read as a geographic area. Then only that area of data needs to be read. Envelope parameters are found under the Advanced Parameters section in the Navigator:
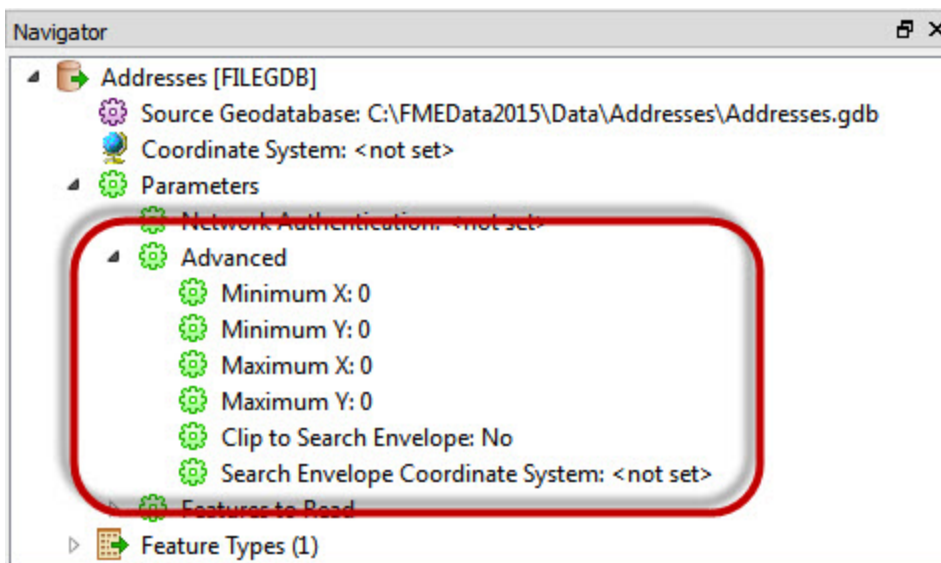
These parameters are available on every spatial data Reader, but have the most effect when the source data is spatially indexed. Then the query is being carried out at its most efficient.

Similarly, there are a number of parameters designed to let the user define how many features to read. These appear in a section of parameters caled Features to Read:
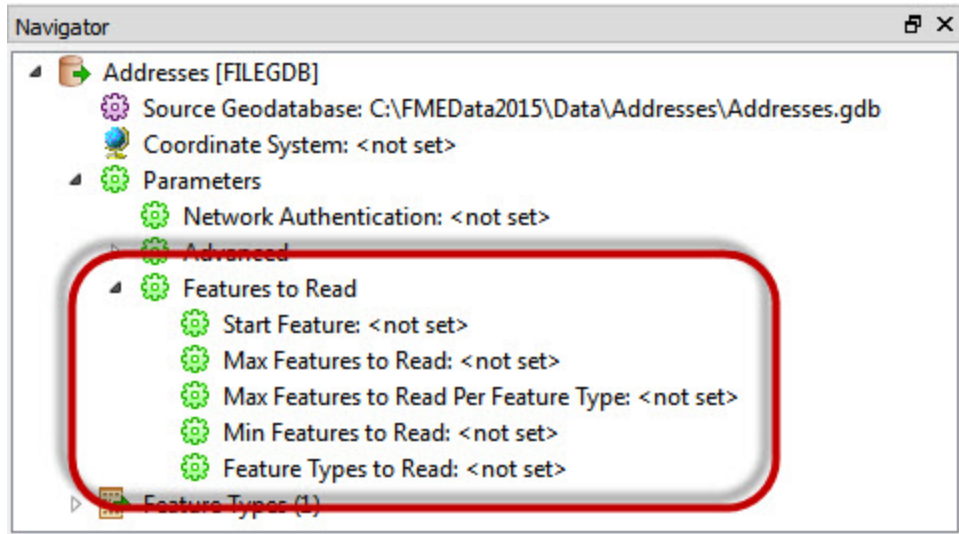
These parameters include the ability to define a maximum number of features to read, and what feature to start at. There is also a parameter that defines which feature types (layers or tables) should be read.

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.

Here, for example, this SQL Server Reader has a 'WHERE Clause' parameter that can be applied:
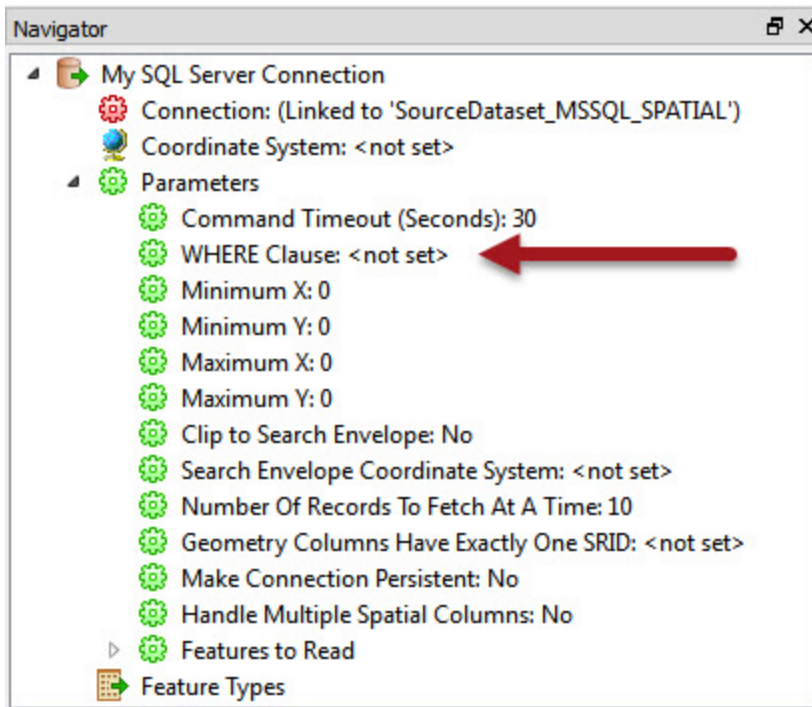
Using this parameter is way more efficient than reading the entire contents of a large table and using a Tester transformer in the workspace.

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!"*

There are two other Reader scenarios to avoid, as they can cause performance degradation.

Firstly – specifically for formats with a table list – there 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 (5)
▷ public.LowerFloor
▷ public.LowerWalls
▷ public.UpperCeiling
▷ public.UpperFloor
▷ public.UpperWalls

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:
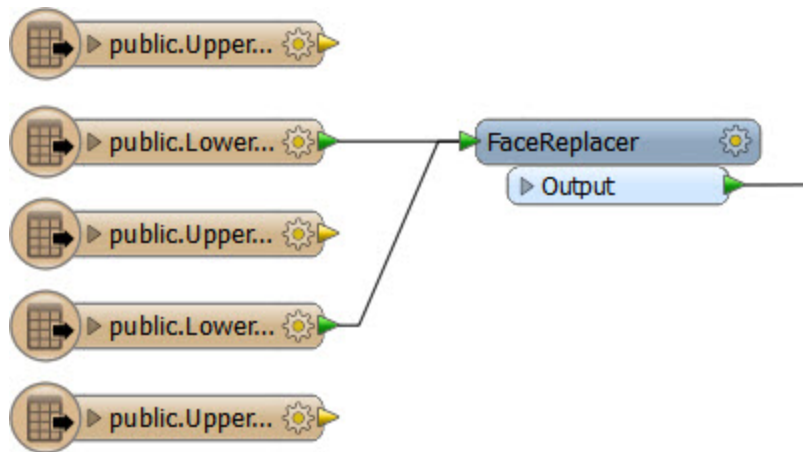
Basically, if you don't need the data, and the feature type is not connected to anything in the workspace, then delete that feature type.

Then the table will not be read and performance will improve.

The second scenario – specifically for file datasets – is a "dangling" Reader. This is where you have deleted all of the feature types, but not the underlying Reader:
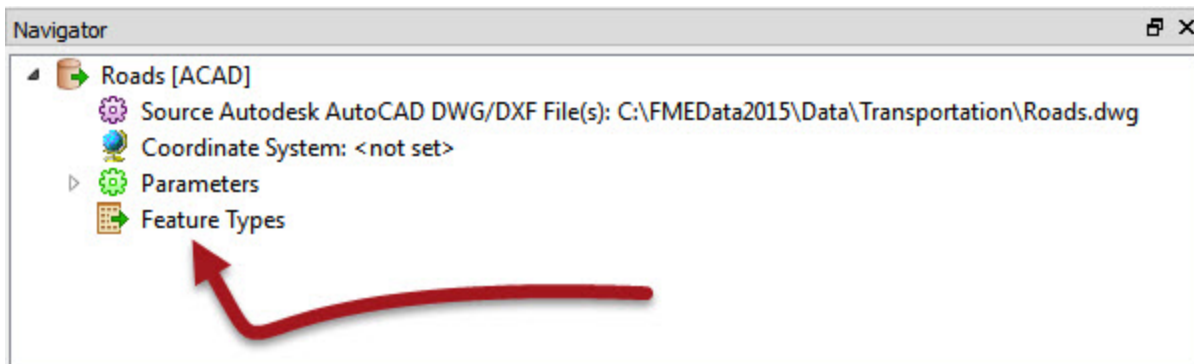
Here the user set up a Reader to read an AutoCAD dataset. The feature type (layer) definitions were deleted from the workspace, but the Reader remains.

When the workspace is run, all the data is still being read from this file, but then discarded as "unexpected input."

Remember, any extra data that is read – of whatever amount – takes time and resources to read, and if it subsequently goes into the workspace, will slow it unnecessarily too.

Assessing Writer Performance

Assessing the speed of writing can be similarly difficult to assessing the speed of reading; again the reason is the same: FME starts writing data as soon as it becomes available, and doesn't necessarily wait until processing is done.

Take the previous workspace, which read a set of address points and found their nearest neighbor, but now has a Writer too:
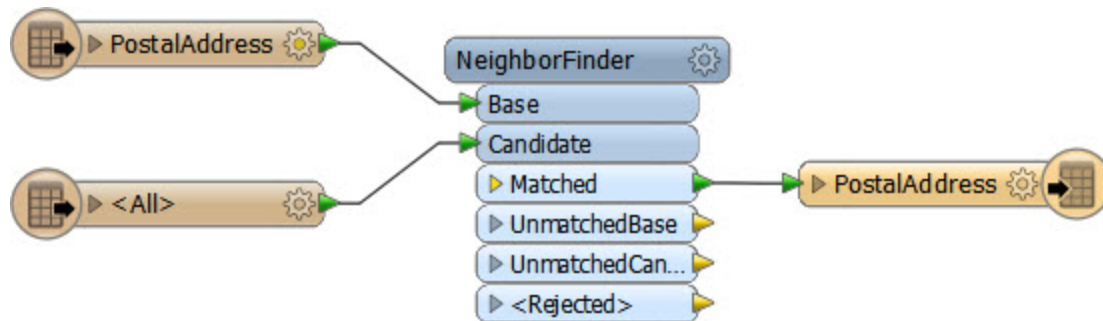
According to the log file, we find that the output dataset is open for writing before the source dataset has even finished being read!

Opened Shape File C:\FMEData2015\Output\PostalAddress.shp for output

Opened DBF File 'C:\FMEData2015\Output\PostalAddress.dbf' for output

DBF Writer: Writing DBF File using character encoding 'ANSI'

Reading source feature # 5000

Reading source feature # 7500

Reading source feature # 10000

Reading source feature # 12500

Plainly it's harder to isolate the Writers to check how fast they are operating; you can't just disable everything else in the workspace without preventing anything from happening.

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:
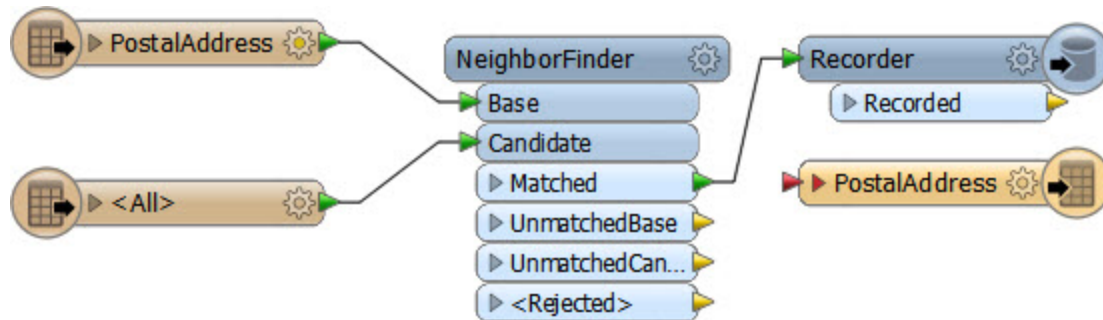
Now replace the Recorder with a Player transformer – to re-read the preserved FFS data – and disable everything else up to that point.
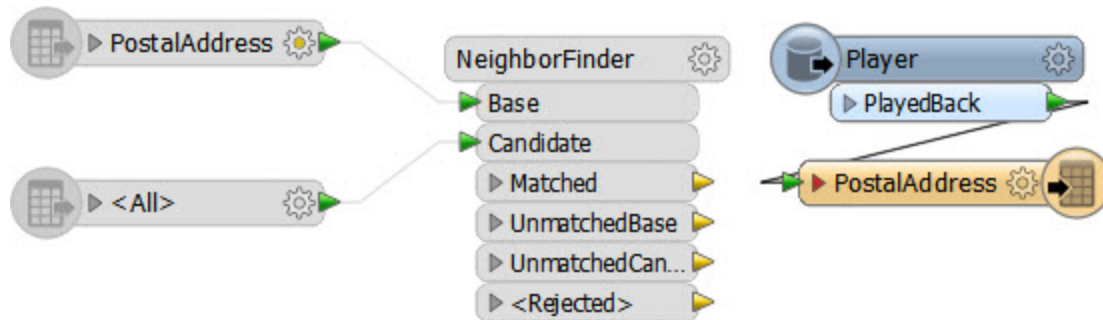
Now the data will be played back into the workspace, and is followed up by being written to the output.

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

Again, the most obvious way to improve writing performance is to upgrade the underlying system to minimize the amount of time FME spends waiting for a response.

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.

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.

However, the main tip for improving Writer performance is for the scenario where a workspace has multiple Writers. In this case it's important to get the writers into the optimum order.

Each Writer is listed in the Navigator window in Workbench. The key is to ensure that the Writer that is to receive the largest amount of data is at the top of the list.

Writers can be re-ordered by moving them up and down in the list in the Navigator window, using the context (right-click) menu, like so:
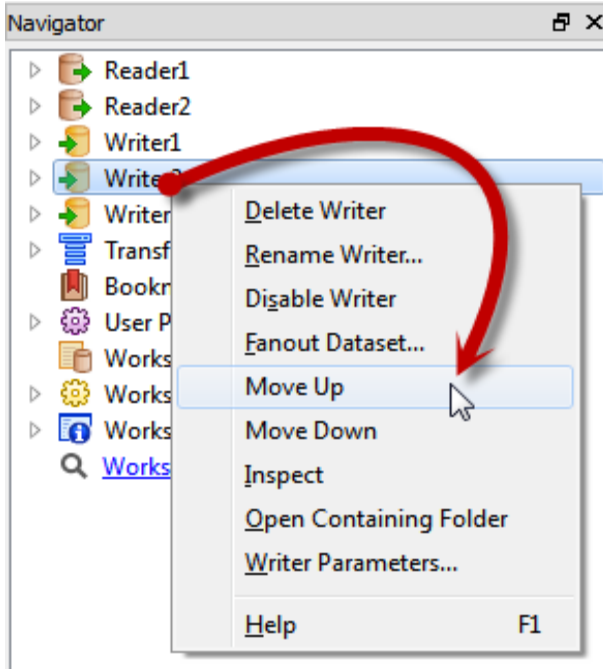
Alternatively, they can be simply dragged up and down with the mouse cursor.

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.

For more information see the FME Evangelist article at: http://fme.ly/FirstWriter.

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."

Exercise 2b Performance Tuning Reading and Writing	
Scenario	FME user; Interopocom, the Interopolis Telecoms company
Data	City Neighborhoods (KML) Cell Signals (CSV)
Overall Goal	Improve Workspace Performance
Demonstrates	Performance tuning Reading and Writing
Starting Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise2b-Begin.fmw*
Finished Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise2b-Complete.fmw*

Let's continue to work on the workspace that processes a dataset of cell phone signals. Remember, because of different specifications, your experience may be greatly different to what is described here.

Now we've deconstructed the log, let's start to clean up any performance issues with the Readers and Writers.

1) Start Workbench

Open the workspace C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise2b-Begin.fmw (or stick with Exercise2a-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.
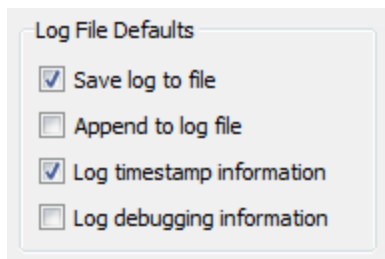
Click on the Runtime icon. Ensure that the Log File Defaults has "Log timestamp information" turned on.

Since the workspace doesn't fail – it's just a little slow – we aren't debugging anything so you can turn off "Log debugging information" (assuming it was already turned on).

2) Assess Readers

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.

```
2014-12-08 10:48:28| 60.2| 0.0|INFORM|MULTI_READER(MULTI_READER): Done reading
872554 features from 3 readers

2014-12-08 10:48:28| 60.2| 0.0|INFORM|Emptying factory pipeline
```

It occurs after 60.2 seconds, about 25% of the total translation time. 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)

The workspace now looks like this:
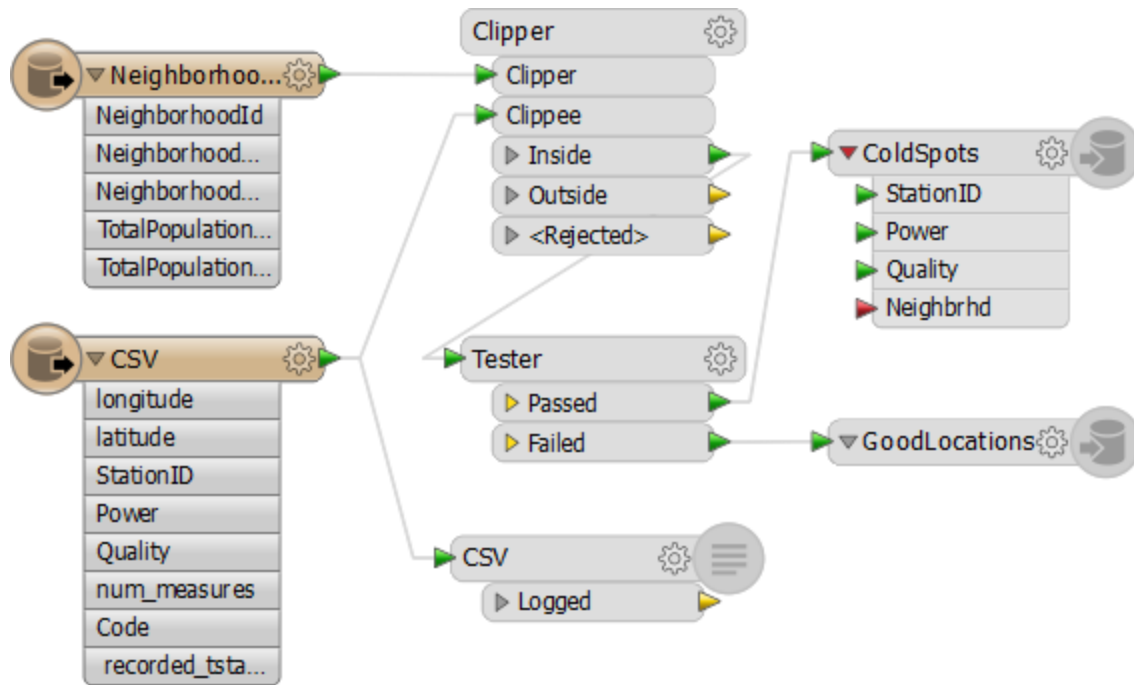
Run the workspace. The data will be read, but not processed or written.

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

```
INFORM|FME Session Duration: 45.3 seconds. (CPU: 37.9s user, 0.4s system)
```

```
INFORM|END - ProcessID: 97468, peak process memory usage: 85344 kB, current
process memory usage: 83164 kB
```

So it's actually a little quicker than the full translation suggests – not surprisingly FME is starting to process that data as it is being read. Still, we might improve on that somehow.

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, but they are regular expressions for text fields, and it would be difficult to filter on a number:
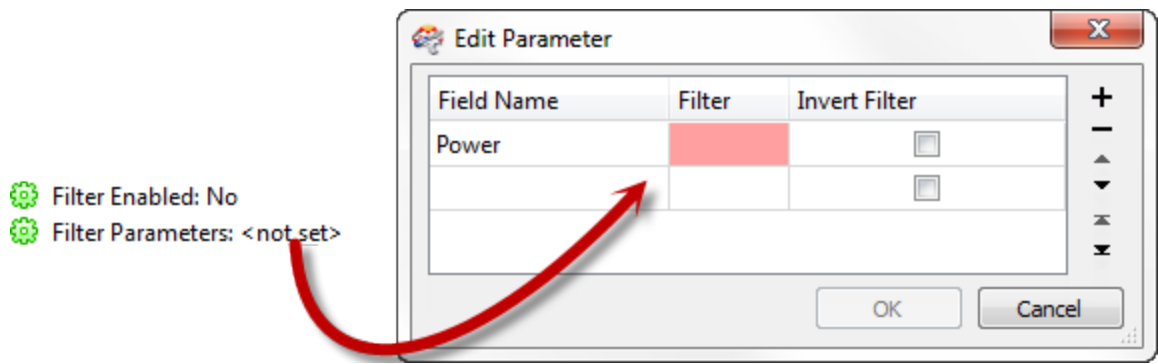
4) Check Other Reader Issues - 1

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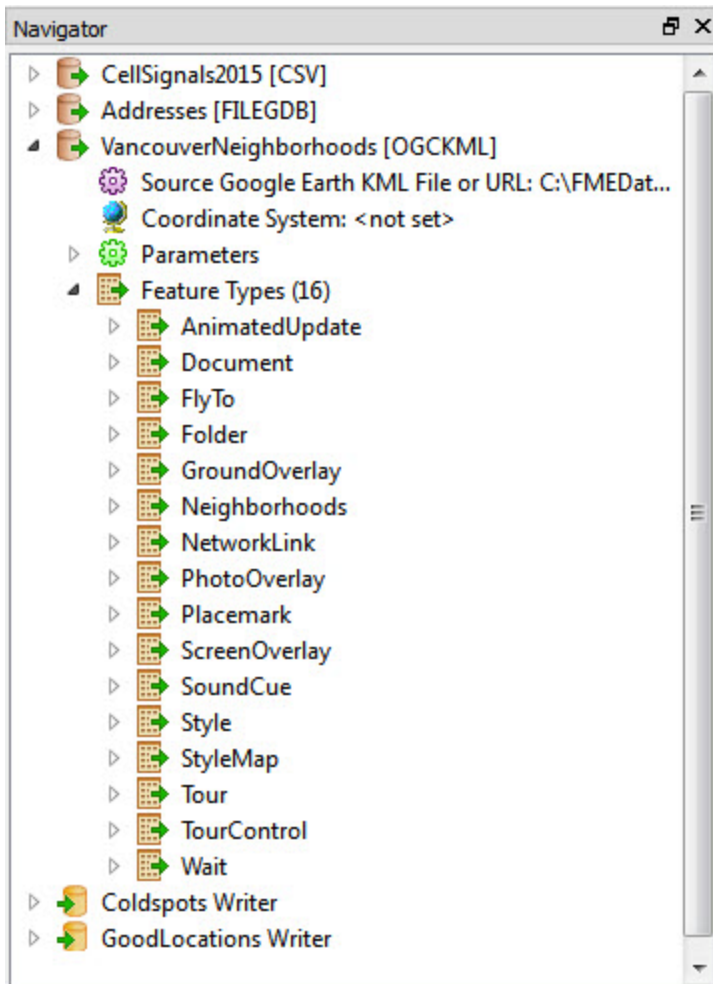
The only feature type we need is Neighborhoods, and that's already connected into the Clipper transformer.

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.
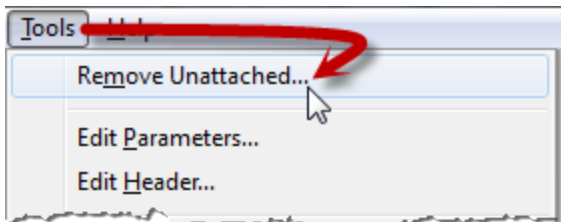
5) Check Other Reader Issues - 2

Another oddity with our Readers is that there are three listed in the Navigator window, but only two used on the canvas.

▷ 🔁 CellSignal750 [CSV]
▷ 🔁 Addresses [FILEGDB]
▷ 🔁 VancouverNeighborhoods [OGCKML]
▷ 🔁 Coldspots Writer
▷ 🔁 GoodLocations Writer

The Addresses [FILEGDB] Reader seems to be reading data that we don't want and don't use. It's what I call a "dangling" Reader, one without feature types. Expand the Reader in the Navigator window to prove that there are no feature types.

🔁 Addresses [FILEGDB]
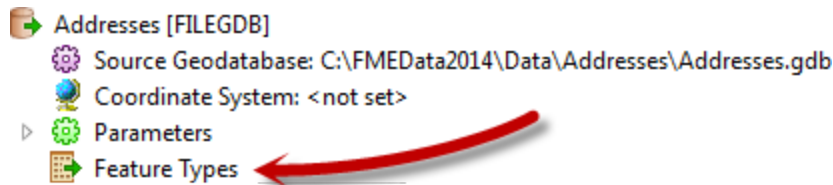   ⚙ Source Geodatabase: C:\FMEData2014\Data\Addresses\Addresses.gdb
    🌐 Coordinate System: <not set>
  ▷ ⚙ Parameters
    🔳 Feature Types ⬅

Having satisfied yourself of that fact, click on the Reader and press the delete key to remove it. Now run the workspace again to see if reading is any quicker.

```
INFORM|FME Session Duration: 38.2 seconds. (CPU: 37.0s user, 0.4s system)

INFORM|END - ProcessID: 88220, peak process memory usage: 80740 kB, current
process memory usage: 78560 kB
```

On my computer it shaves about 7 seconds (15%) of the time off reading, which is a good start.

6) Check Writer Performance

We won't check the time being taken to write the data, just look for the only obvious improvement: the order of the Writers.

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:
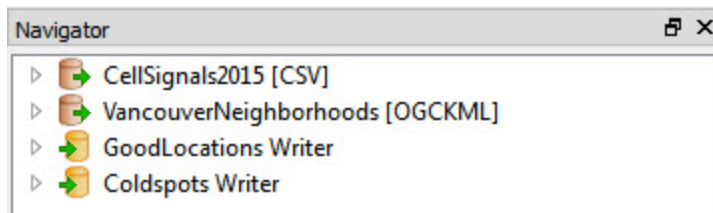
Writer	Features	Position
ColdSpots	69	1st
GoodLocations	697348	2nd

OK, the Writer with the most features is not currently top of the Writers list. Let's fix that.

Right-click on the GoodLocations Writer in the Navigator window and choose the option Move Up.

Now that Writer should appear above the other in the 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:

```
INFORM|FME Session Duration: 4 minutes 1.0 seconds. (CPU: 185.7s user, 50.4s system)

INFORM|END – ProcessID: 96656, peak process memory usage: 3065096 kB
```

Now, on my computer, I get the following:

```
INFORM|FME Session Duration: 2 minutes 44.1 seconds. (CPU: 151.9s user, 8.9s system)

INFORM|END – ProcessID: 98972, peak process memory usage: 1734748 kB
```

That's way better. I've reduced the time taken by 40% from the original. Additionally, peak memory use has dropped by 50%!

> *In FME2015 there is a new workspace parameter called "Order Writers By" that will let you adjust the order in which Writers are initialized; either by the order of Writers in the Navigator (as described here) or by the order features arrive at the Writers.*

Transformation Optimization

Transformation Optimization

Assessing Transformation Performance
Improving Transformation Performance

CONNECT.
TRANSFORM.
AUTOMATE.

2015.0

FME Desktop Advanced Training 2015

Now that you can interpret a log file, you can start to work on improving the performance of your data transformation.

Assessing Transformation Performance

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.

If you want to be able to assess the time taken in transformation then the first thing to do is calculate the read time so that it can be ignored. So firstly disable everything in the workspace except the Readers and run the workspace. Take a note of the elapsed time.

Now re-enable the transformers, but leave the Writers disabled.
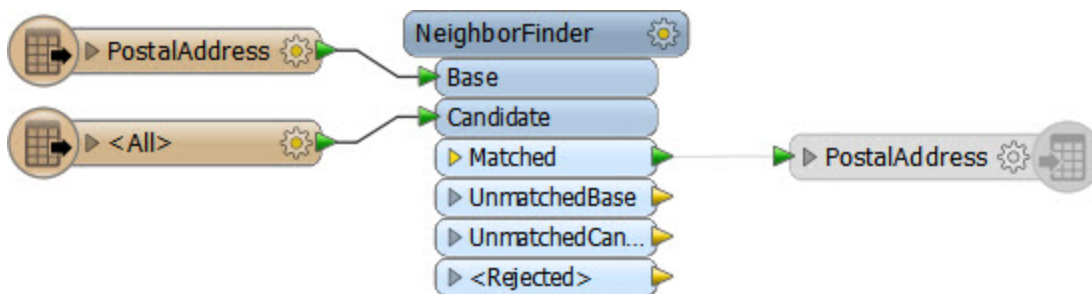
Don't be tempted to add an Inspector or Logger transformer to the end to see what is happening to the output. This will only slow the translation down and give you a false measure. And be sure to disable the actual Writer, and not just the feature types or connections to them.

The one 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.
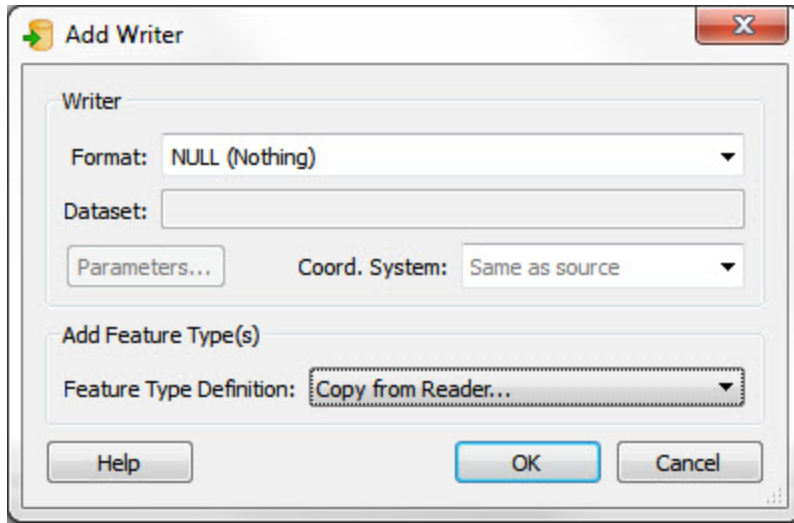
Now I know it took 5.4 seconds to read the data, and the whole process took 28.2 seconds, so I can infer that the transformation part takes 22.8 seconds.

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."

However, 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. But in a group-based transformation a transformer performs an operation on a group or collection of features.

It is this grouping of data that causes performance degradation. Group-based transformers must store the group of features together (cached either to memory or disk) to be processed, whereas feature-based transformers do not need to do so.
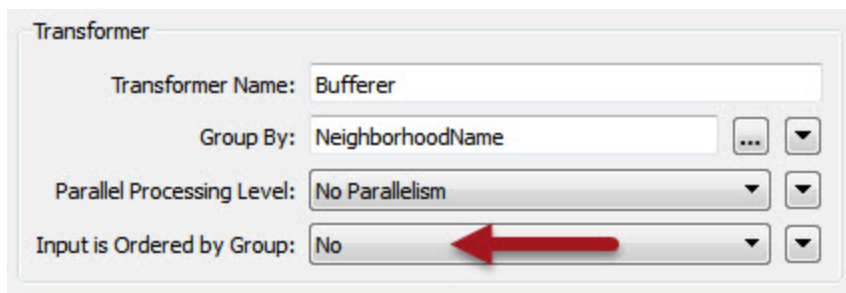
Jake Speedie says…

"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)"

Turning Group-based Transformers into Feature-based Transformers

Obviously, when a group-based transformer is needed, then it must be used. However, most group-based transformers have a parameter that, in effect, turns them into feature-based.

The usual parameter is called "Input is Ordered by Group" and appears near the Group By parameter in most transformer dialogs.

The condition for applying this 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.

Jake Speedie says...

Let's think back to the airport departure gate boarding passengers. Most airlines first call passengers with a physical disability, 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.
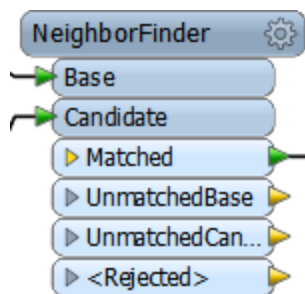
Look at this log file for a workspace that uses a NeighborFinder. By default it looks like this:

```
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 do we 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.
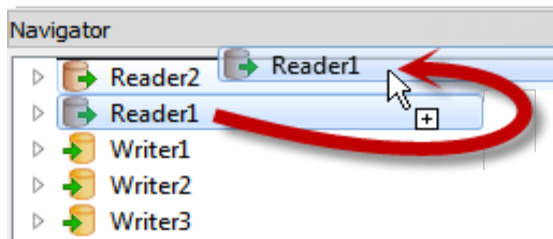
Transformer Selection

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, you could check for matching features in two different datasets using either the Matcher transformer, or the ChangeDetector, or even the FeatureMerger!

Another example would be isolating all road features that pass through a park.

One could use either the Clipper, like so:
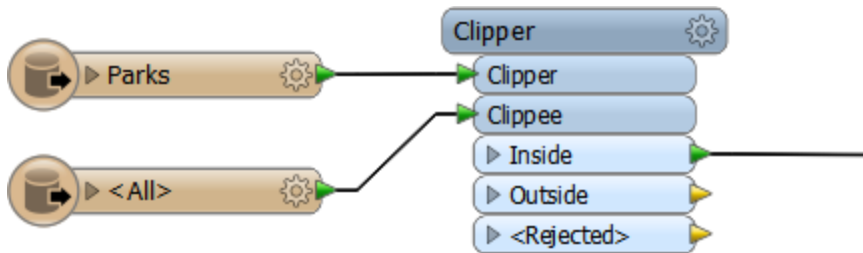
Or the LineOnAreaOverlayer, with a test for _overlaps => 1, like so:

The performance for the LineOnAreaOverlayer would be this:

```
Translation was SUCCESSFUL with 0 warning(s) (0 feature(s) output)
FME Session Duration: 3 minutes 57.6 seconds. (CPU: 196.2s user, 37.6s system)
END - ProcessID: 23840, peak process memory usage: 3129576 kB
```
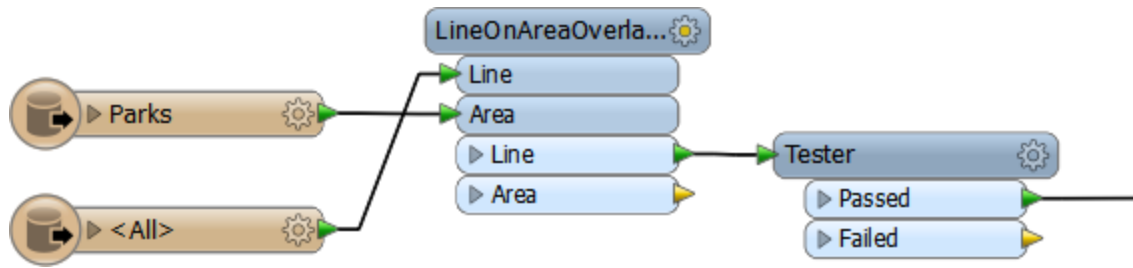
While the Clipper (in Multiple Clippers mode) would be this:

```
Translation was SUCCESSFUL with 0 warning(s) (0 feature(s) output)
FME Session Duration: 23.5 seconds. (CPU: 22.2s user, 0.8s system)
END - ProcessID: 13528, peak process memory usage: 1471896 kB
```

And in Clippers First mode would be this:

```
Translation was SUCCESSFUL with 0 warning(s) (0 feature(s) output)
FME Session Duration: 21.4 seconds. (CPU: 20.4s user, 0.6s system)
END - ProcessID: 24488, 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.
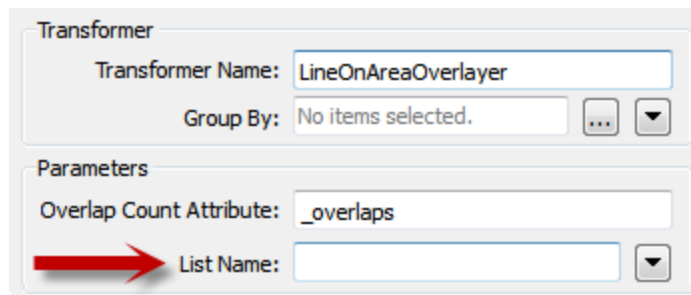
Basically, each transformer works in a different way, has subtle variances in functionality, and will have different performance for any given task. Therefore a translation will benefit in performance if the author is careful in their choice of transformers, and maybe carries out some testing first.

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:
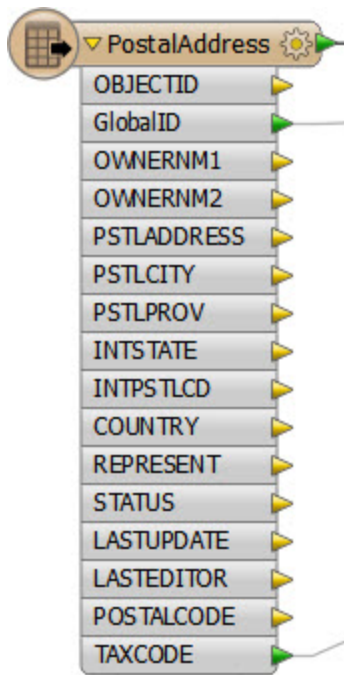
But the outgoing schema looks like this:

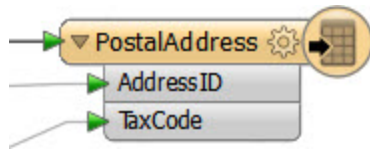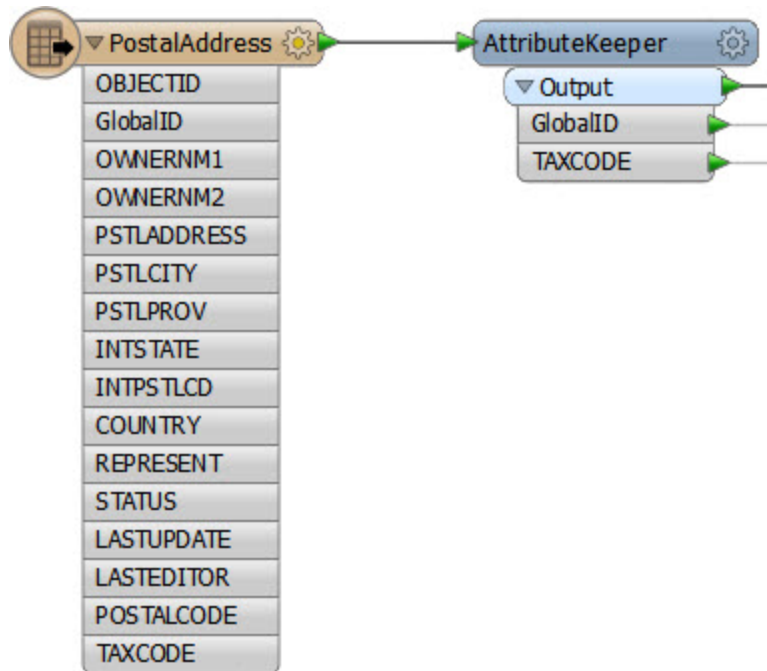Since so many of the source attributes are not required in the output, it makes sense to remove them from the translation, and as early as possible by using an AttributeRemover (or Keeper) directly after the source feature type:

One specific type of attribute to beware of is a List. A List 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!

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-style 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.

Geometry and Transformation

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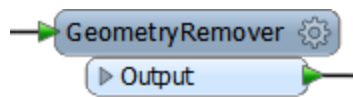
Many FME users create translations that handle tabular – non-spatial – data. If you are reading a spatial dataset, then writing it to a tabular format, be sure to remove the geometry early in the workspace, just as you would an attribute.

Additionally, you can reduce the size of individual features by removing vertices. The Generalizer transformer will help you to do so and has very many parameters to control the results.
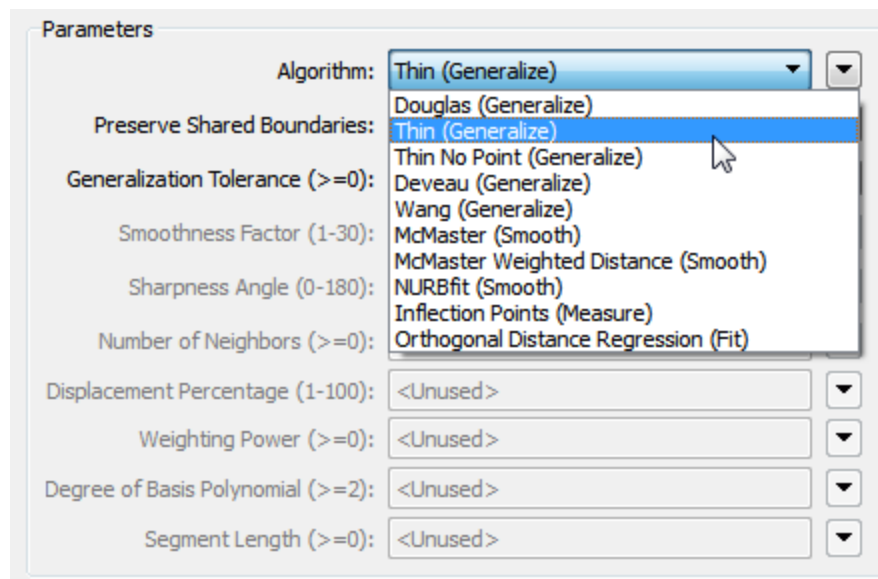
General Optimizations

Here are a few general suggestions that can be used to improve the performance of a workspace. Some of these come straight from our developers.

> *Jake Speedie says…*
>
> *"It's said that the 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 (new for FME2015). 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 marginally slower than fixed licenses.

Exercise 2c Performance Tuning Data Transformation	
Scenario	FME user; Interopocom, the Interopolis Telecoms company
Data	City Neighborhoods (KML) Cell Signals (CSV)
Overall Goal	Improve Workspace Performance
Demonstrates	Performance tuning data transformation
Starting Workspace	C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise2c-Begin.fmw
Finished Workspace	C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise2c-Complete.fmw

Let's continue to work on the workspace that processes a dataset of cell phone signals.

We've already deconstructed the log and cleaned up the Readers and Writers. Now let's use our new knowledge of transformation performance to try and speed up the workspace.

1) Start Workbench

Open the workspace C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise2c-Begin.fmw which follows on from exercise 2b.

2) Check for Extra Transformers

The first aspect of the workspace to check is any extra transformers that aren't needed and that will be slowing performance. The most obvious is the Logger transformer. It 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.
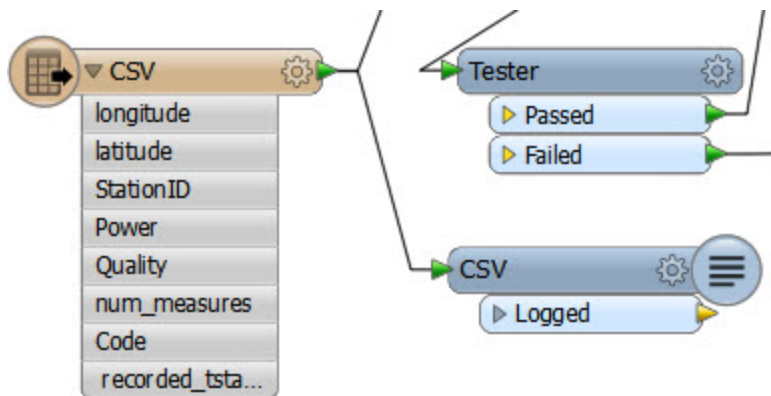
3) Remove Attributes

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:
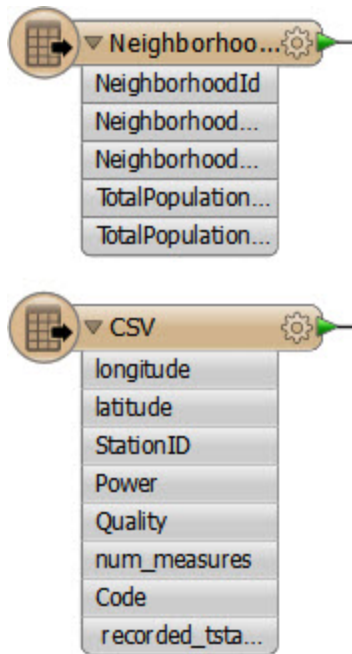
The Readers contain quite a lot of attributes, on both datasets. The Writers contain very few attributes, and the GoodLocations feature type has none at all. This suggests we can remove some attributes that are not going to be needed in the output.
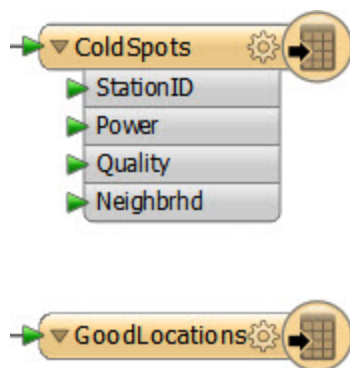
Put an AttributeKeeper transformer after the Neighborhood feature type, but before the Clipper.

Use it to keep only the NeighborhoodName attribute.

Focus on

Now do the same to the CSV (signal) data, keeping only the attributes StationID, Power, and Quality.
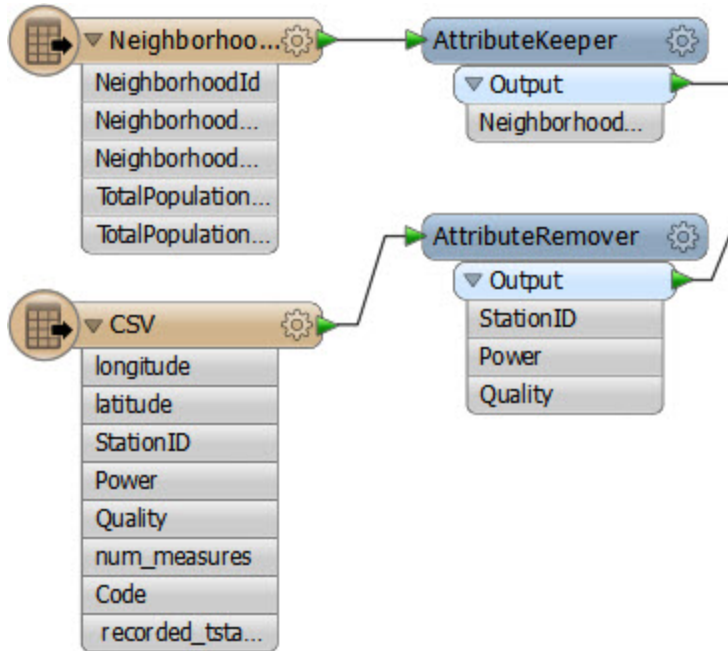
Jake Speedie says…

"Do we really need to remove attributes from only six neighborhood features?

Yes! Because we're copying them onto (depending on the source dataset used) 1.7 million CSV features."

---

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.

There's no real indication this is the wrong transformer to use (although there are others) but 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 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, right-click the VancouverNeighborhoods Reader in the Navigator window and choose Move Up to bring it to the top of the list:

5) Run Workspace

Let's run the workspace to see what we have so far.

Remember, after Reader improvements we had this result:

```
FME Session Duration: 2 minutes 26.5 seconds. (CPU: 133.8s user, 12.4s system)

END - ProcessID: 98972, peak process memory usage: 1734748 kB
```

The result now is:

```
FME Session Duration: 2 minutes 16.6 seconds. (CPU: 126.6s user, 6.9s system)

END - ProcessID: 102032, peak process memory usage: 109780 kB
```

It's improving (especially the memory use). But I think we can still do better!

6) Rearrange Transformers

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.

We could prevent that by moving the Tester transformer to before the Clipper.

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, but before it reaches the Clipper:
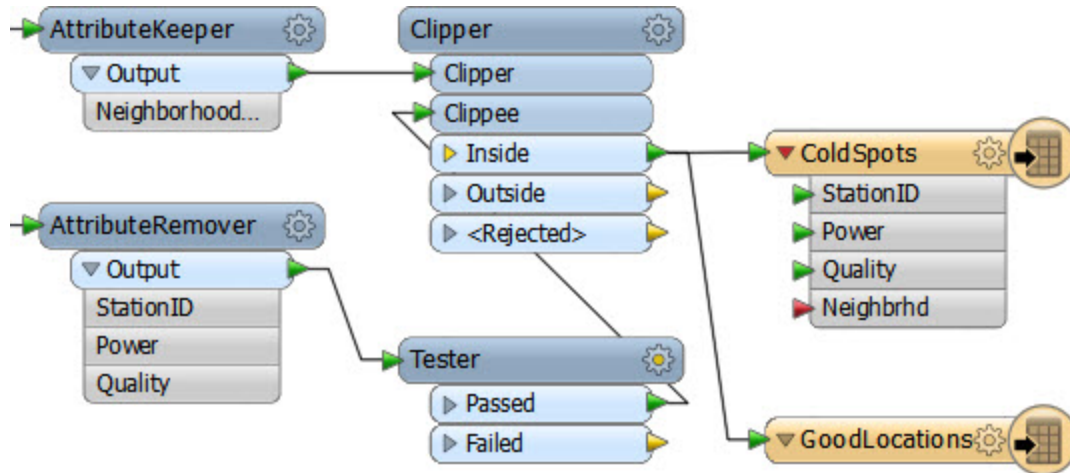
Finally, let's fix the feature mapping.

Click on the connection from Clipper:Inside > GoodLocations, making sure to click closer to the Clipper than the feature type.

Then drag that connection onto the Tester:Failed port.
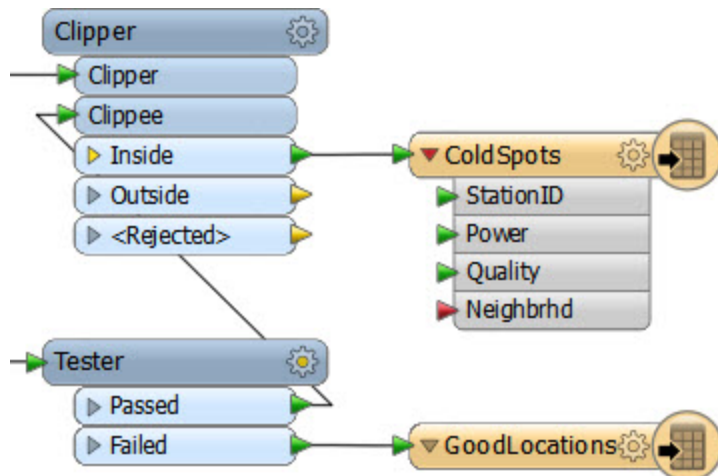
Re-run the workspace. The result will be something like this:

```
FME Session Duration: 1 minutes 41.6 seconds. (CPU: 88.7s user, 12.2s system)
```

```
END - ProcessID: 220504, peak process memory usage: 110028 kB
```

Hurrah! Compared to the original log we're over 50% faster with 95% less memory use!

> *There is, perhaps, one other way we can upgrade this workspace's performance. It's radical and unintuitive, but it might just work!*

Do you remember the previous tip about using the PointCloud XYZ reader for CSV data? Let's give that a try and see what we can do.

Open the workspace C:\FMEData\Workspaces\DesktopAdvanced\Exercise2c-Begin-Advanced.fmw

Firstly, delete the CSV Reader (and feature types), the AttributeRemover/Keeper for the CSV data, and the Tester transformer too.
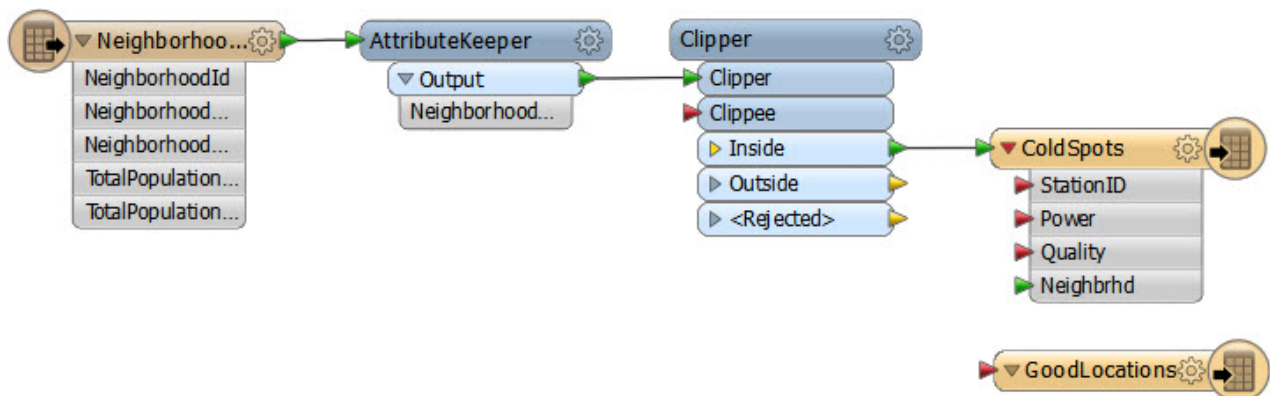
Now select Reader > Add New Reader and in the Add Reader dialog enter the following values:

Reader Format   Point Cloud XYZ
Reader Dataset   *C:\FMEData2015\Data\CellSignals\CellSignals2015.csv*

What are going to be key here are the Reader parameters, 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 following:

You can type a component name, which is what you will have to do for all of them except the X and Y fields.

This will cause each column in the CSV data to be stored as a point cloud component.

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.

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.
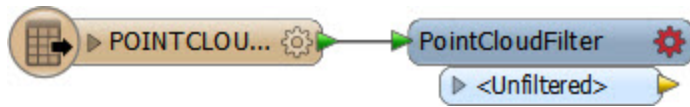
This is the point cloud equivalent to a Tester and may be quicker than testing each point individually.

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 <user component> 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. The dialog should look like so:
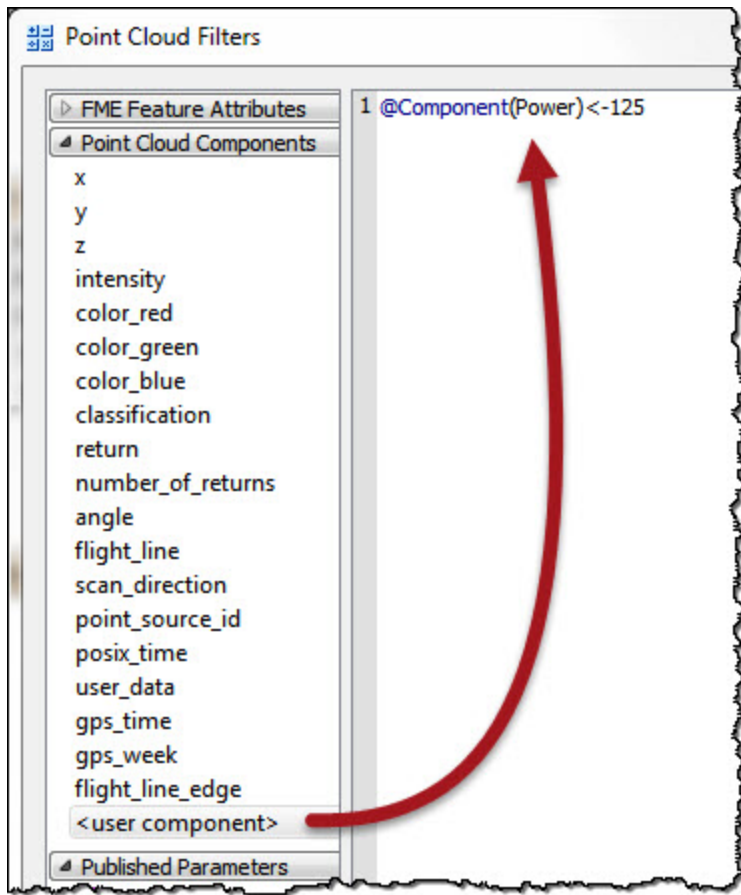
Click OK to close that dialog. Enter a new port name of Bad Signal. Click OK.

The workspace now looks like this, with a new output port:
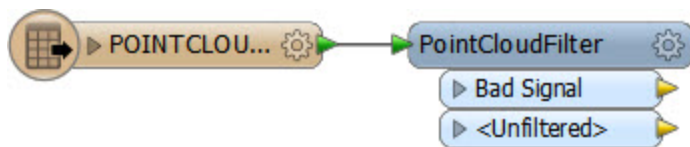
Add two PointCloudCoercer transformers to the workspace, one attached to each output from the PointCloudFilter. One output should be directed to the Clipper:Clippee port, the other to the GoodLocations output feature type:
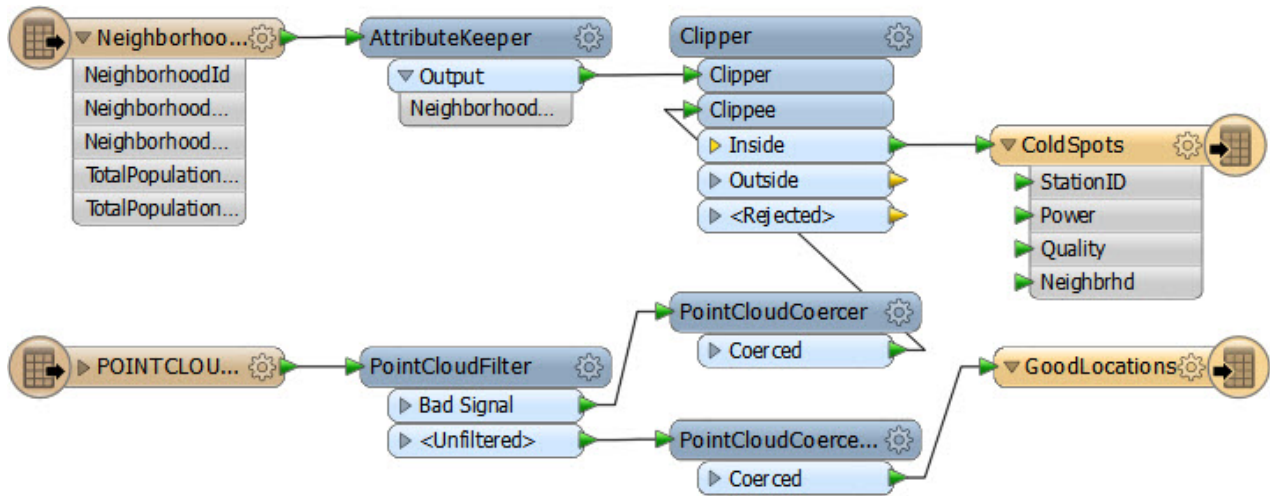
The final step is to set the coercer parameters and expose some attributes.

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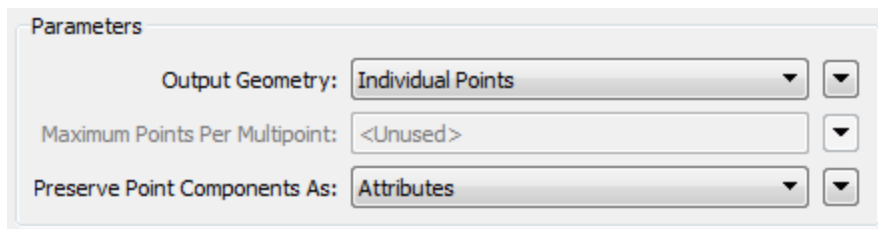
For the <Unfiltered> data, which doesn't need attributes, leave the Point Components to Preserve section empty. For the Bad Signal data, enter the following component names to extract them as attributes:

- StationID
- Power
- Quality

Now save and run the workspace. This time the log will report the performance as:

```
FME Session Duration: 54.7 seconds. (CPU: 43.1s user, 10.9s system)

END - ProcessID: 111126, peak process memory usage: 124744 kB
```

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

Database Optimization

Database Optimization

Database Reading
Database Writing

FME Desktop Advanced Training 2015

Using database functionality inefficiently will do more to degrade performance than almost anything else in FME.

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.

Database Reading

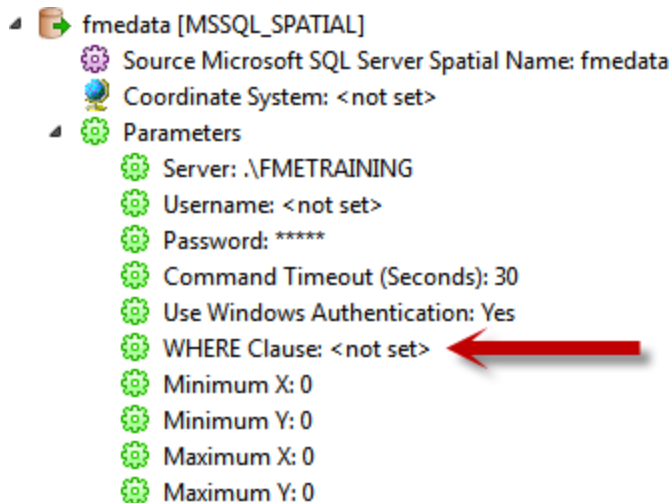Reading and filtering data (querying) from a database is nearly always faster when you can use the native functionality of the database.

▲ ↪ fmedata [MSSQL_SPATIAL]
    ⚙ Source Microsoft SQL Server Spatial Name: fmedata
    🌐 Coordinate System: <not set>
  ▲ ⚙ Parameters
      ⚙ Server: .\FMETRAINING
      ⚙ Username: <not set>
      ⚙ Password: *****
      ⚙ Command Timeout (Seconds): 30
      ⚙ Use Windows Authentication: Yes
      ⚙ WHERE Clause: <not set> ⬅
      ⚙ Minimum X: 0
      ⚙ Minimum Y: 0
      ⚙ Maximum X: 0
      ⚙ Maximum Y: 0

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 bounding box. When you set these then FME's query to the database includes these parameters.

This is way faster than reading the entirety of a database 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:
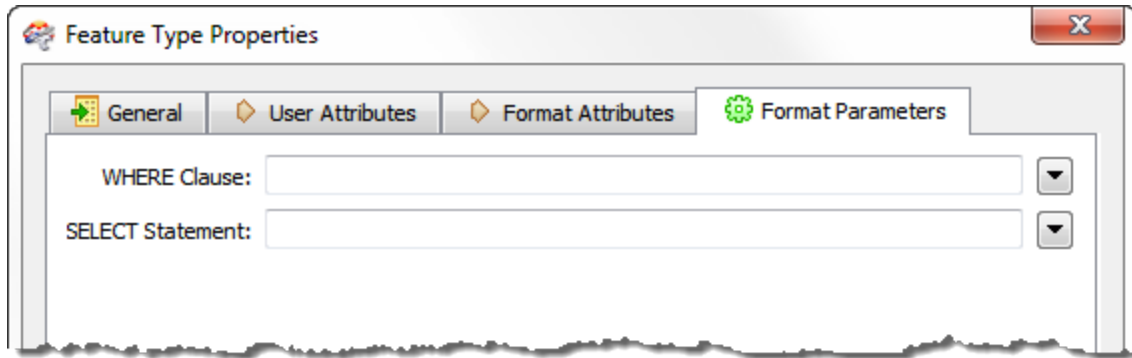
Besides Readers, transformers can also be used to query database data. The best to use is the SQLExecutor (or SQLCreator) as these pass their queries to the database using native SQL. If you don't want to write SQL then you can use the FeatureReader transformer; but be aware this transformer is more generic and won't give quite the same performance.
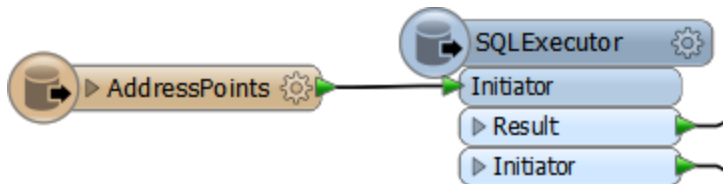
The SQLExecutor is particularly worth being aware of, as it issues a query for each incoming feature. This can be useful where you need to make multiple queries.

For example, here a query is issued to the database for every address 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:
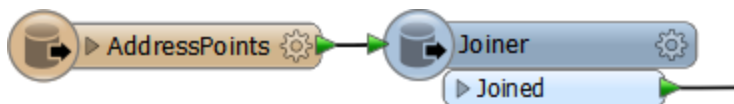
> *Jake Speedie says…*
>
> *"FME is fast, but if you can filter or process data in its native environment, it's likely to be faster still.*
>
> *For example, a materialized view (a database object containing the results of a query) is going to perform better than reading the data and filtering it in 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:

2014-07-10 14:43:06| 8.5| 0.0|

2014-07-10 14:43:13| 8.8| 0.3|

2014-07-10 14:46:29| 18.0| 9.1|

2014-07-10 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 inbuilt 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. A similar parameter exists for the Reader to be able to ignore Relationships.

WHERE Clause: <not set>
Spatial Data Only: No
Resolve Domains: No
Resolve Subtypes: Yes
Ignore Network Info: Yes
Ignore Relationship Info: Yes
Split Complex Edges: No
Minimum X: 0
Minimum Y: 0
Maximum X: 0
Maximum Y: 0
Clip to Search Envelope: No

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

Attributes				
Name	Type	Width	Precision	
B ▸ Forename	char	11		
C ▸ Middle Name	char	7		
D ▸ Surname	char	11		
E ▸ Born	char	5		

Q Filter

☐ Read formulas (.formula)

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.

The Oracle Spatial Writer parameters look like this:

⚙ Persistent Connection: Yes
⚙ Transaction To Start Writing At: 0
⚙ Features Per Bulk Write: 200   ⬅
⚙ Features To Write Per Transaction: 1000
⚙ SQL Statement To Execute Before Translation: <not set>
⚙ SQL Statement To Execute After Translation: <not set>
⚙ Enforce strict attribute conversion: No
⚙ Handle Multiple Spatial Columns: No
⚙ Fanout Dataset: No

The two common parameters for controlling database writes are "Features per Transaction" and "Features per Bulk Write" (chunk).

Features per Transaction

Features per Transaction controls how many features are entered into a database before a commit command is issued. Each commit command adds delay to the writing process, so setting this parameter is a way to 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.

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.

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.

Spatial Column SRID:	
Drop Table First:	Yes ▼
Truncate Table First:	No ▼
Create With OIDs:	No ▼

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

Exercise 2d Database Optimization	
Scenario	FME user; City of Interopolis, Planning Department
Data	Addresses (Geodatabase) Garbage Schedule (Geodatabase)
Overall Goal	Improve performance of workspace
Demonstrates	Database Reading Optimization
Starting Workspace	C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise2d-Begin.fmw
Finished Workspace	C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise2d-Complete.fmw

The city has an online service powered by FME Server, where a resident can input their postal code and receive information about their garbage collection.

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

1) Start Workbench

Open the workspace C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise2d-Begin.fmw

The workspace used for the service looks something like this:
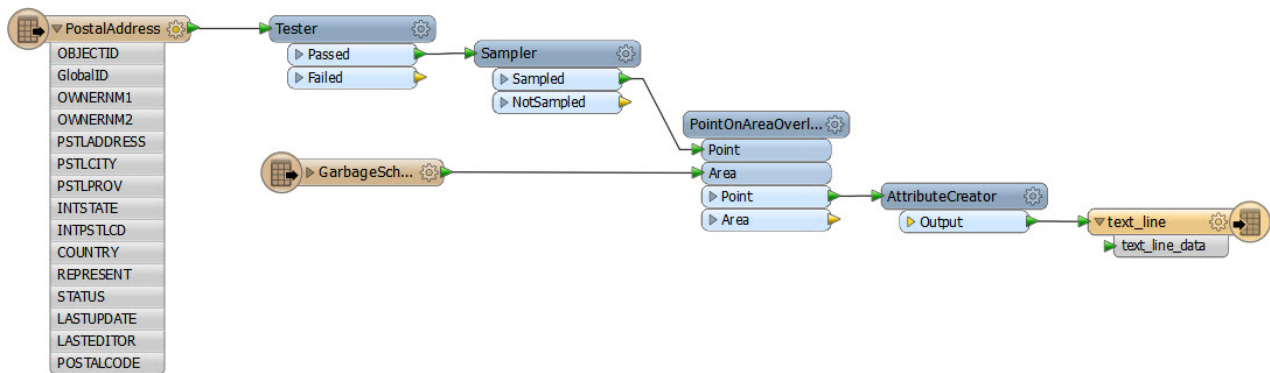
For our purposes, the important part is this:
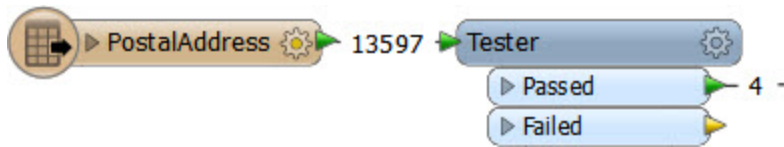
This shows the workspace is reading 13,500+ addresses and filtering out ones that don't match the required postal code. However, it makes sense to use a WHERE clause if possible, so we don't have to read so much data to start with.

2) Run Workspace

To get a comparison, run the workspace. The statistics for features read will look like this:

```
|=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=|
|                       Features Read Summary                    |
|=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
|GarbageSchedule                                               6
|PostalAddress                                            13597
|==============================================================
|Total Features Read                                      13603
|=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

The performance will read like this:

```
FME Session Duration: 4.3 seconds. (CPU: 4.0s user, 0.2s system)

END - ProcessID: 103496, peak process memory usage: 79428 kB
```

3) Open Feature Type Properties

The Geodatabase Reader doesn't have a WHERE clause, but the feature type does.

So, open the properties dialog for the PostalAddress feature type and click the Format Parameters tab.

In the WHERE Clause parameter enter: POSTALCODE = 'V6E1Y8.'
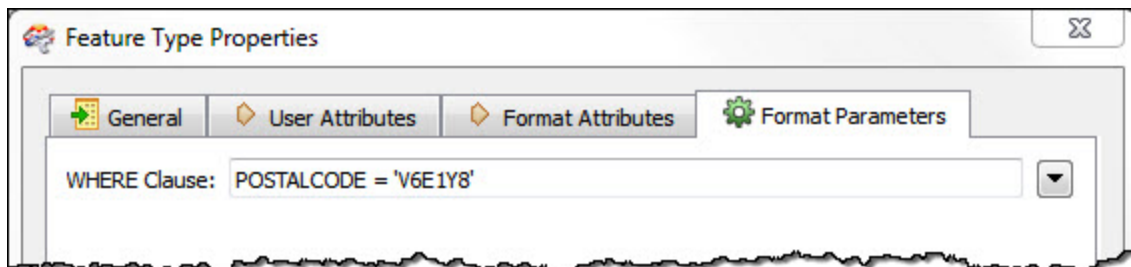
4) Delete Tester

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 4 features are read from the database (because 4 addresses match that postal code). The performance improves accordingly:

```
FME Session Duration: 0.6 seconds. (CPU: 0.4s user, 0.2s system)

END - ProcessID: 99108, peak process memory usage: 84404 kB
```

Memory usage hasn't improved, but the translation ran 80% faster.

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 decide to 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.

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

Parameter	Processes
No Parallelism	1 Process
Minimal	Cores / 2
Moderate	Cores
Aggressive	Cores x 1.5
Extreme	Cores x 2

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:

FME Edition	Process Cap
Base Edition	4
Professional Edition	8
All Other Editions	16

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 have a group-by parameter in order for the user to be able to define the parallel processing groups.
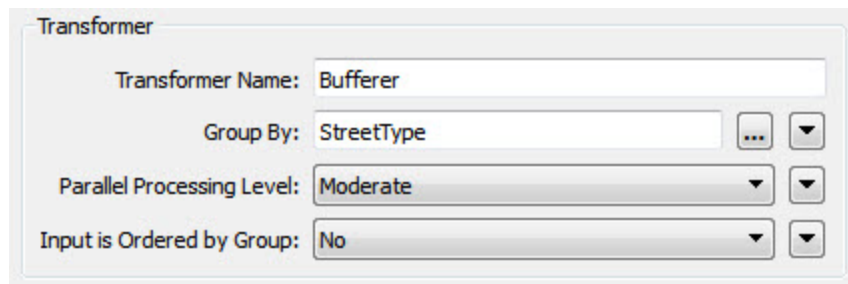
For example, this Bufferer transformer is set up to buffer a set of street features.

Each type of street (highways, roads, lanes, etc.) 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 you run a translation in parallel mode, then you'll see a number of "worker" processes appear in your process manager:

Image Name	User Name	CPU	Memory (...	Description
fmeworker.exe *32	imark	15	26,456 K	
fmeworker.exe *32	imark	15	26,460 K	FME EXE
fmeworker.exe *32	imark	14	26,460 K	
fmeworker.exe *32	imark	13	26,608 K	
fme.exe *32	imark	10	115,196 K	FME EXE
fmeworker.exe *32	imark	08	21,792 K	
fmeworker.exe *32	imark	04	12,648 K	
fmedatainspector.exe *32	imark	02	213,180 K	FME Data Inspector
fmeworkbench.exe *32	imark	02	202,304 K	FME Workbench

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:
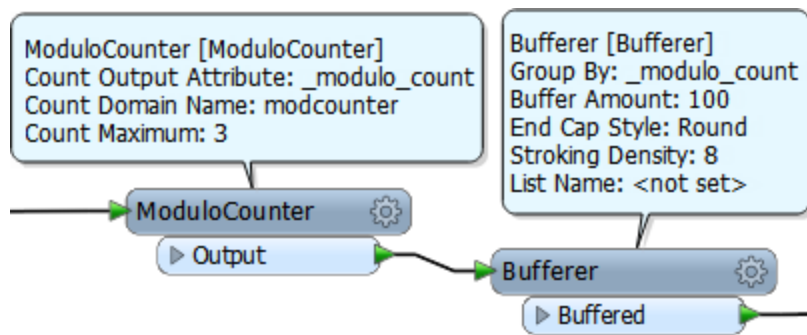
Exercise 2e Parallel Processing	
Scenario	FME user; City of Interopolis, Planning Department
Data	Various
Overall Goal	Assess Parallel Processing Capabilities
Demonstrates	Parallel Processing
Starting Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise2e-Begin-1.fmw* *C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise2e-Begin-2.fmw* *C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise2e-Begin-3.fmw*
Finished Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise2e-Complete-1.fmw* *C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise2e-Complete-2.fmw* *C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise2e-Complete-3.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) Start Workbench

Start Workbench and open the workspaces in turn.

Assess each workspace for its ability to employ parallel processing. You should answer the following questions for each workspace.

- 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 workspaces.

Performance, FME Server, and FME Cloud

FME Server adds an extra dimension to FME performance.

In terms of performance, the prime reason for using FME Server is one of scalability, and the main consideration 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.

For example, I have a very large set of vector data in a spatial database, and want to create a series of map tiles from it. Basically I need to clip the data to a map tile outline, rasterize it, and write it out to a raster format such as PNG:
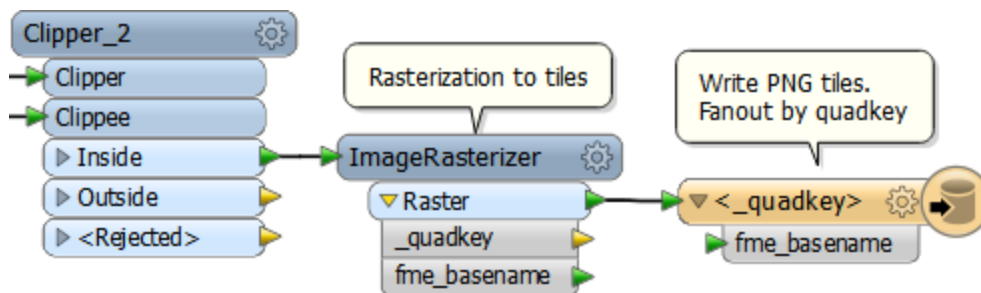
The problem is that the amount of data being processed is so great, the workspace takes days to run. I need to run the process on FME Server by dividing the work up into different jobs.

So I upload my workspace to FME Server and create a master workspace to control it. The master workspace calculates the bounds of each tile and runs the main workspace on FME Server using a ServerJobSubmitter transformer:
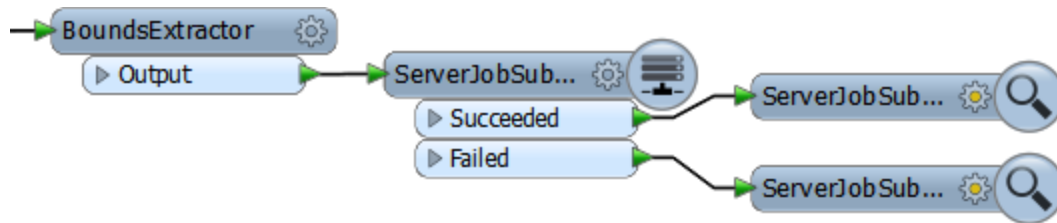
The bounds of each tile are sent to the main workspace by using published parameters:


Username: postgres
Password: *****
Minimum X: -10000000 (Linked to 'POSTGIS_IN_SEARCH_ENVELOPE_MINX_POSTGIS')
Minimum Y: 6000000 (Linked to 'POSTGIS_IN_SEARCH_ENVELOPE_MINY_POSTGIS')
Maximum X: -9000000 (Linked to 'POSTGIS_IN_SEARCH_ENVELOPE_MAXX_POSTGIS')
Maximum Y: 7000000 (Linked to 'POSTGIS_IN_SEARCH_ENVELOPE_MAXY_POSTGIS')
Clip to Search Envelope: Yes

So now I have a method by which I can pass the bounds of the tiles to be created to a workspace on FME Server, and share the load over multiple server engines by running the workspace once for each tile.

Pushing to 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:
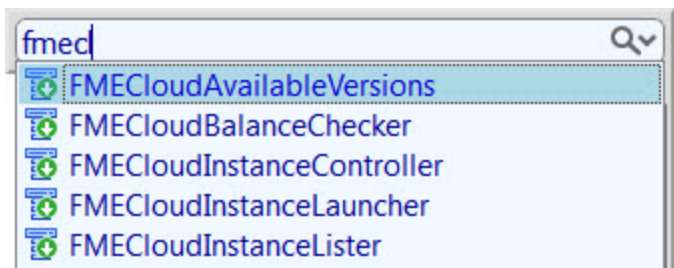
With the FMECloudInstanceLauncher transformer I can run my master workspace (as in the example above) and have it automatically start an FME Cloud instance and run the job on it.
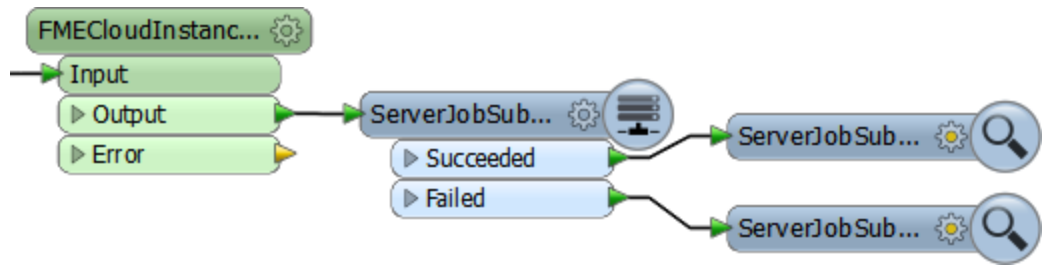
This way I can start a new instance for each job, or run several jobs on one instance, depending on the type of instance and how many engines it has running on it.

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

Chapter 3 - Custom Transformers

FME Training Presentation

Custom Transformers

Custom Transformers

Basic Custom Transformers

What is a Custom Transformer?
Custom Transformer Purposes
Creating a Custom Transformer
Naming a Custom Transformer
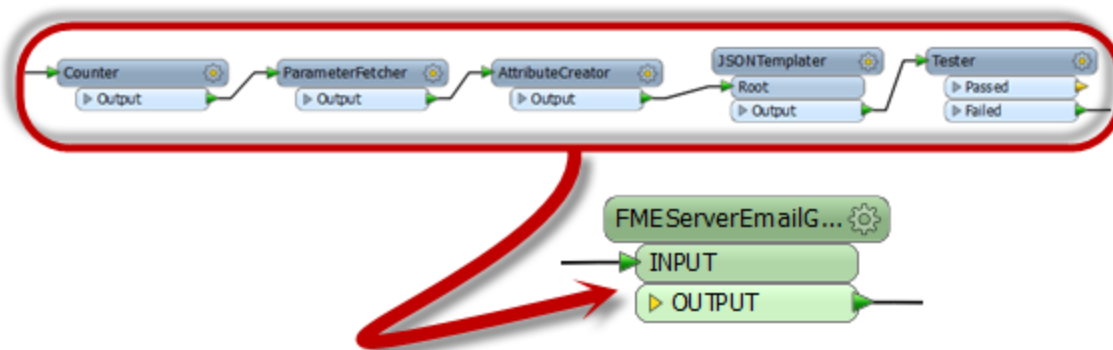The New Custom Transformer

FME Desktop Advanced Training 2015

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.

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

  By encapsulating a sequence of transformers in a single object, they 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

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.
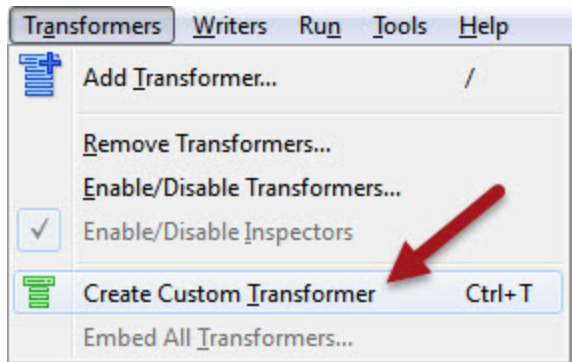
A Custom Transformer can be created from scratch – i.e. you start with an empty custom transformer and add content into it – or can be created from an existing sequence of transformers.

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 three raster-based transformers.
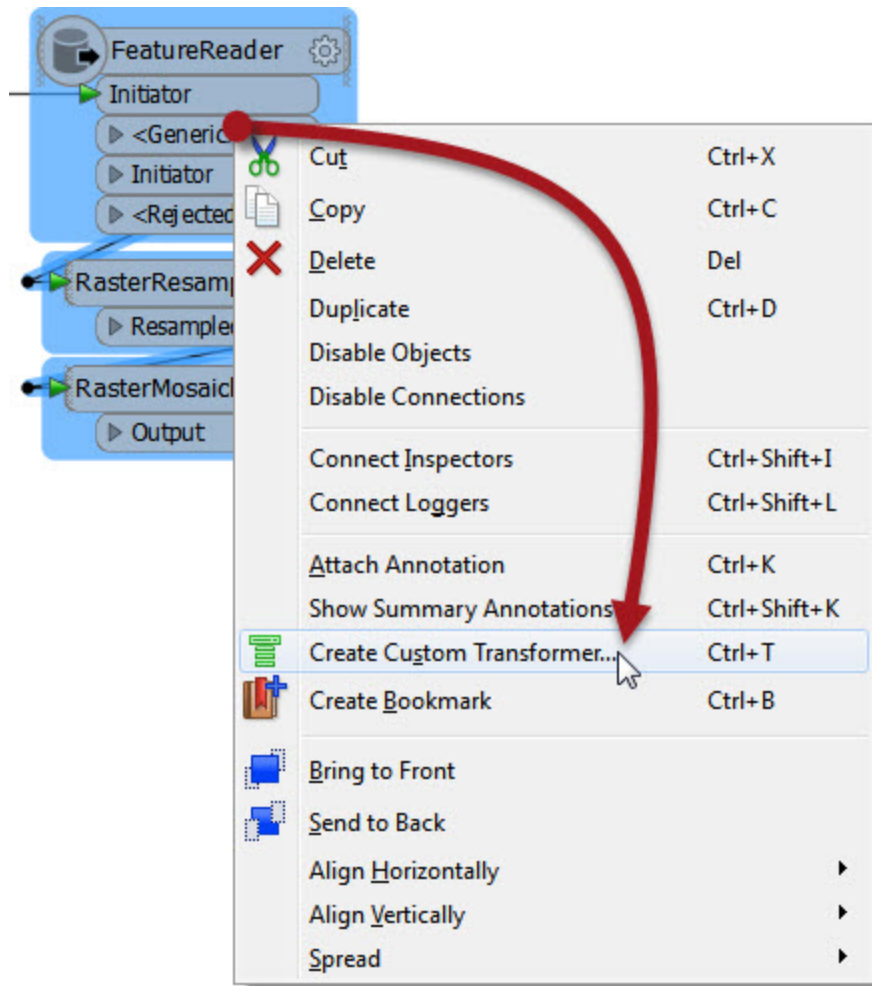
In FME2015 the FeatureReader transformer adds raster functionality and so replaces the RasterReader transformer.

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.
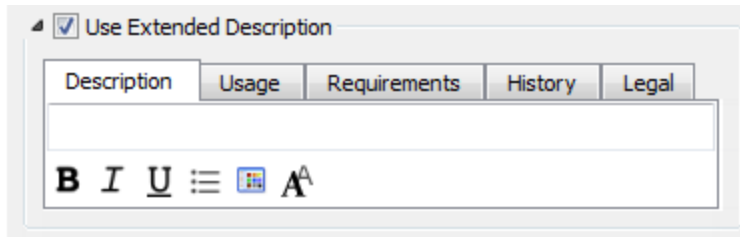
These fields are particularly important when you intend to share the custom transformer with work colleagues or clients.

The New Custom Transformer

A newly created custom transformer then looks like this:
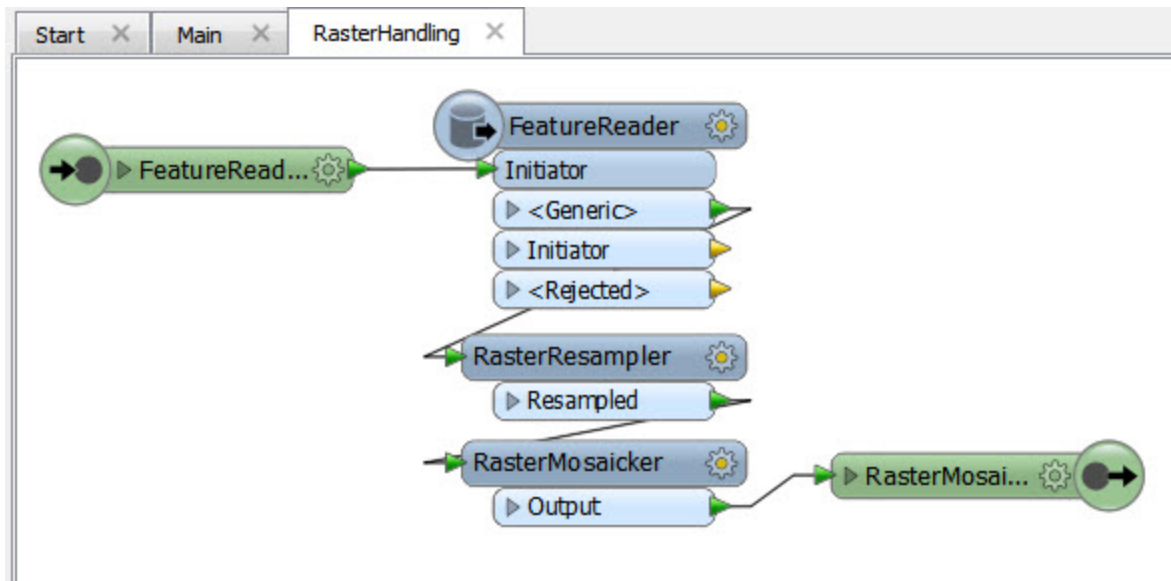
Notice that it appears under a new tab on the Workbench canvas and consists of the three original transformers with additional input and output objects.

First Officer Transformer says…

'*If the custom transformer had been created from scratch, without any original transformers, it would look like this:*
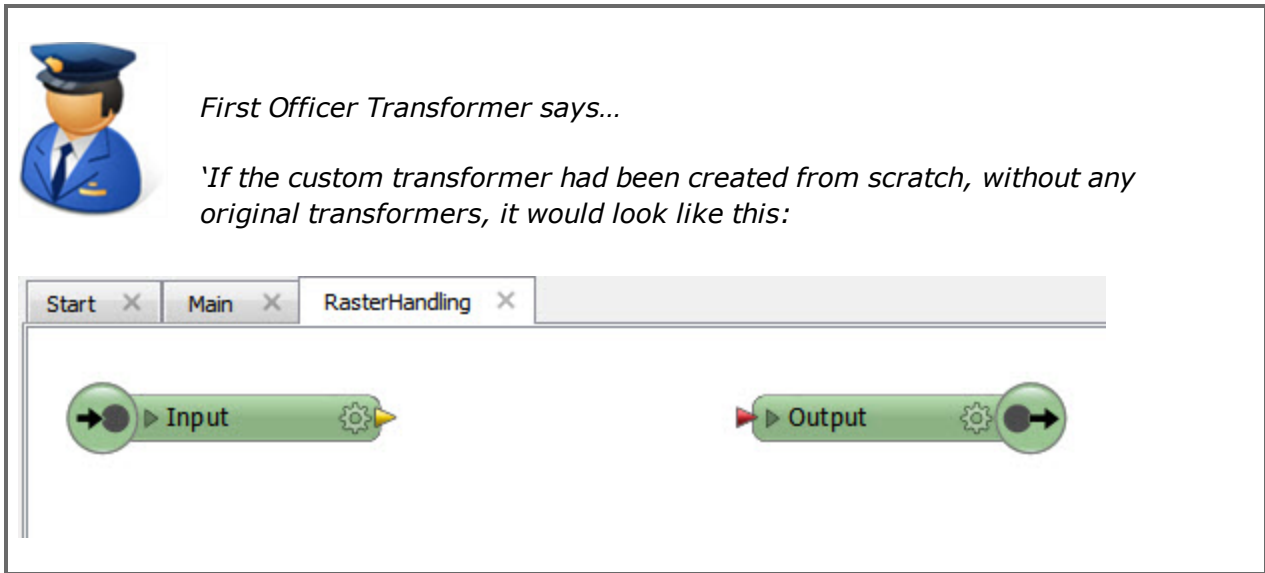
In the Navigator window, where a workspace would have a section labelled Workspace Properties, a custom transformer has Transformer Properties.

This is where the information – name, category, description, etc. – that was entered earlier can be edited.
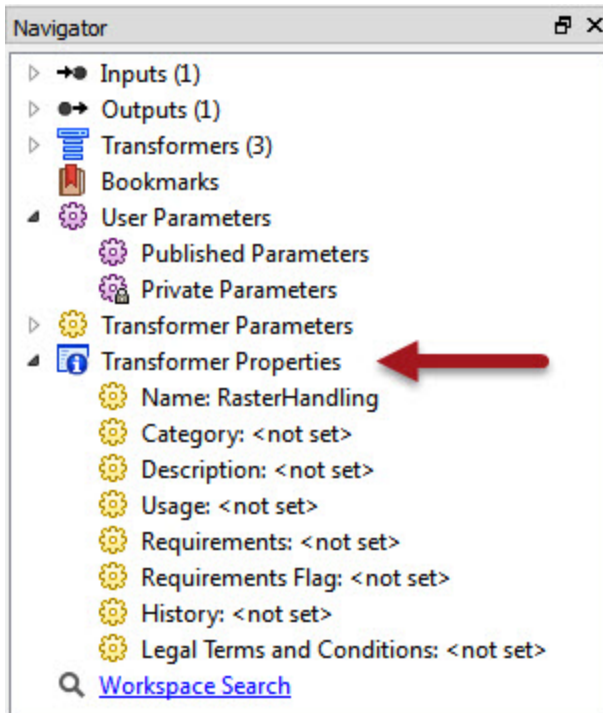
When you click on the Main tab, to return to the main canvas view, the three original transformers have now been replaced by a custom transformer object that is automatically connected into the existing workspace:
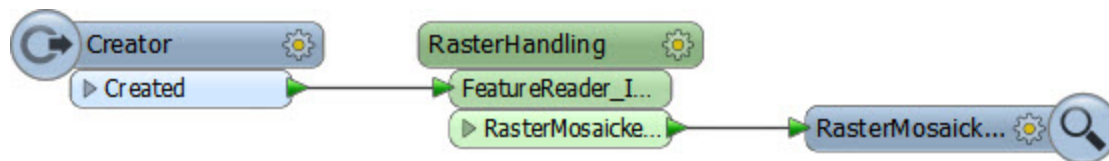
This custom transformer looks and behaves in the same way as any standard FME transformer; with input and output ports (that match the input/output objects in the custom transformer tab), plus a parameters dialog.

Exercise 3a Custom Transformer Creation	
Scenario	FME user; City of Interopolis, Planning Department
Data	City Neighborhoods (KML)
Overall Goal	To set up an easy way for FME authors to calculate the average density of an item, for any set of polygon features.
Demonstrates	Custom Transformer creation and use
Starting Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise3a-Begin.fmw*
Finished Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise3a-Complete.fmw*

The scenario for this exercise is that an FME user has a workspace that calculates the population density for neighborhoods in the city of Vancouver. Having just found out about custom transformers, the user thinks this would be a good workspace to turn into a general solution: a custom transformer that calculates the average density of items in a known space.

1) Start Workbench

Open the workspace *C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise3a-Begin.fmw*.

You may wish to run the workspace and examine the output to see what it does and how it works.

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.
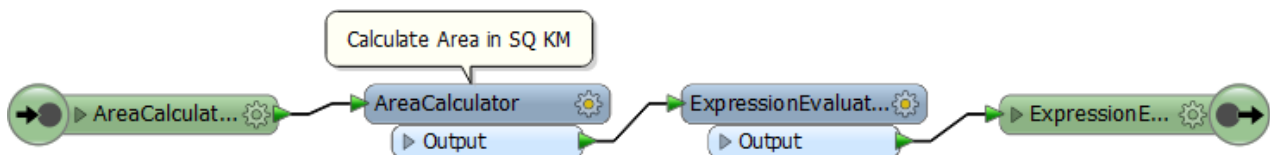
First Officer Transformer says…

'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.
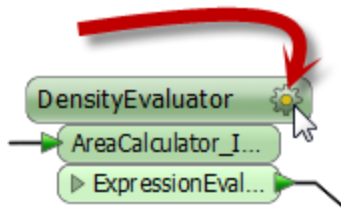
Back in the Main tab, click on the cog wheel icon on the custom transformer to open its parameters dialog.

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.

|	TotalPopulation2011	NeighborhoodArea	PopulationDensity2001	PopulationDensity2011
1	41375	5.47537933750436	7236.02832932823	7556.55406678333
2	31445	3.27441763660346	8674.82500780334	9603.23437318697
3	26400	3.6639492852468	6696.32631073586	7205.33990639604
4	54690	3.71337291984275	7537.62161899573	14727.8501730217
5	44540	1.98160703997034	21255.4755561579	22476.7065828887
6	12165	3.88490690100229	2979.47937877577	3131.34917000494

Using Custom Transformers

Custom Transformers aren't just for tidying a workspace, but also as a way of re-using content.

Re-Using a Custom Transformer

Once a custom transformer has been created, its definition is stored in the workspace and can be used any number of times within that workspace.

In fact, a custom transformer that has been created can be placed into the workspace just as any other FME transformer, either through the Transformer Gallery, or by using Quick Add.

Custom transformers can be found in the Transformer Gallery window under a section labelled "Embedded Transformers."
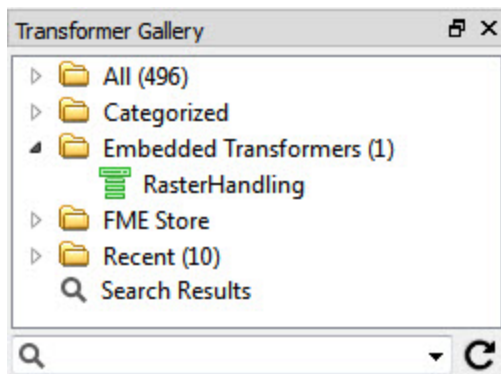
The transformer name, category, and description all appear in the Quick Add tool, so it is definitely worthwhile setting these parameters.
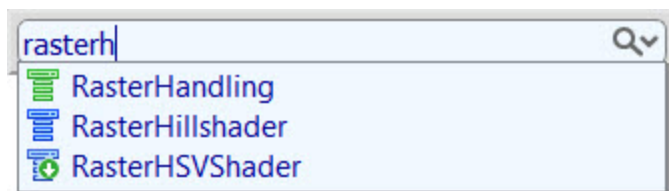
Newly placed instances of a custom transformer will be renamed (or numbered) as necessary, in order to avoid a clash of names.

Editing a Custom Transformer

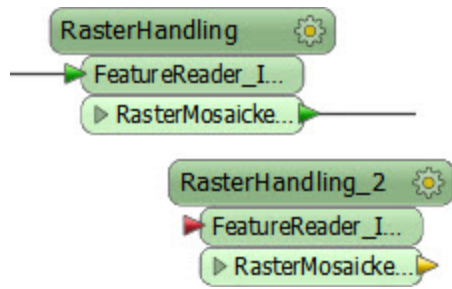When custom transformers are reused like this, a key benefit for maintenance purposes is that every instance can be updated or edited simply by editing the custom transformer definition.

Custom transformer definitions are edited by clicking the appropriate canvas tab and making whatever changes are required, just as changes are made on the main canvas.

For example, if the Interpolation parameter is changed for the RasterMosaicker 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.
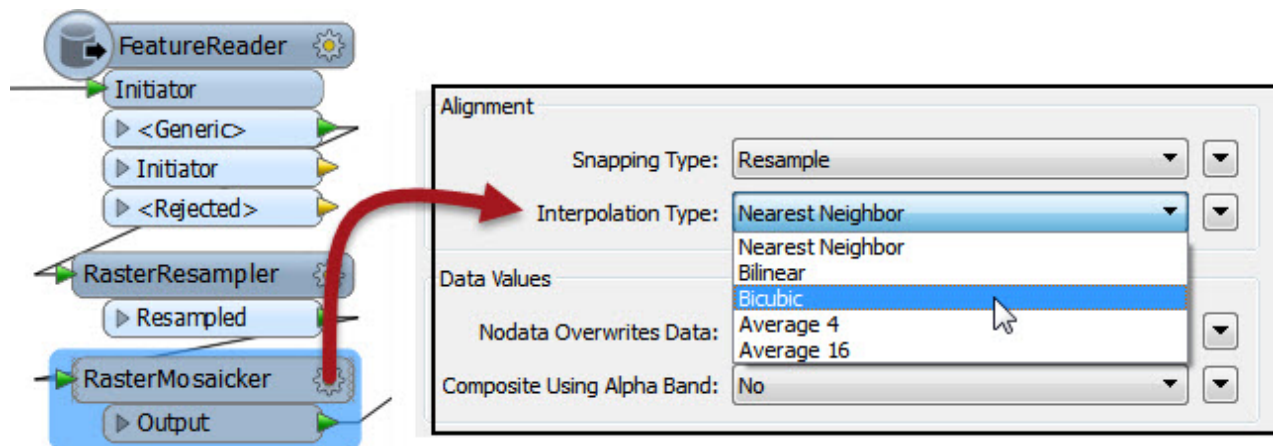
Custom Transformer Input/Output Ports

The input and output ports on a custom transformer are defined by input/output objects in the custom transformer definition itself.
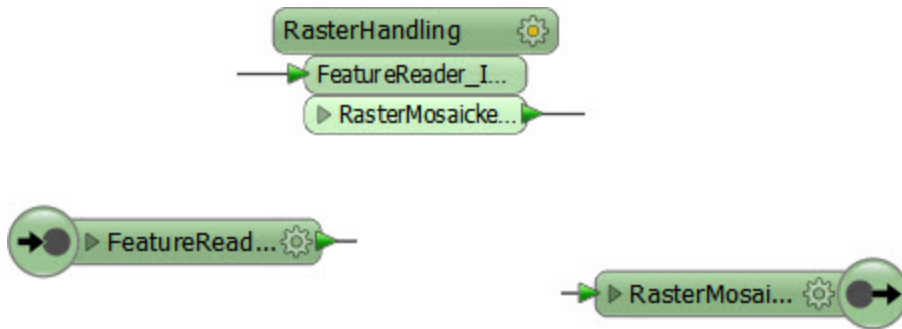
Firstly, these input/output objects can be renamed, in order that the transformer ports get named appropriately. You can either double-click the object, choose Rename from the context menu, or press F2, in order to rename the object.

For example, here the user is renaming an input port from FeatureReader_Initiator to simply Raster.
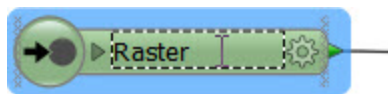
Renaming the input and output ports is useful for making the custom transformer object more legible, and for helping the user to understand what data is supposed to connect to the port.

For example, after editing the transformer might look like this:

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 menubar.
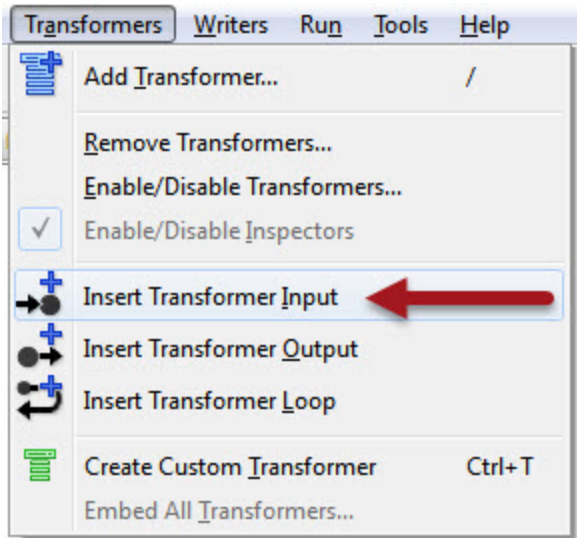
For example, here a user has added a new input port to handle vector data.
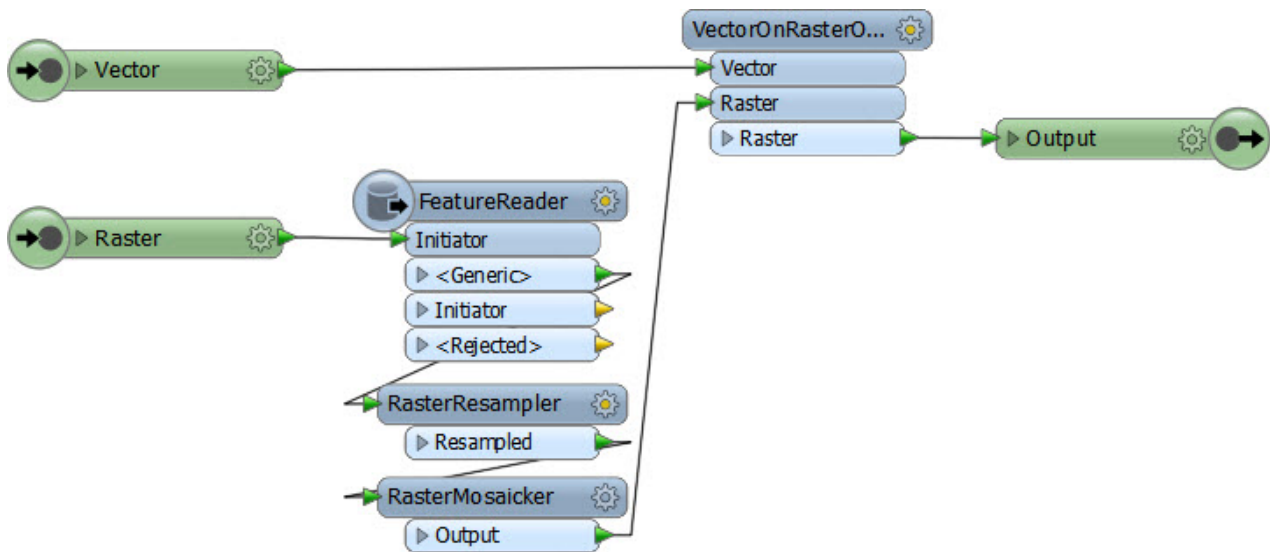
This means that each instance of the custom transformer in the main canvas will now have an extra input port, like so:
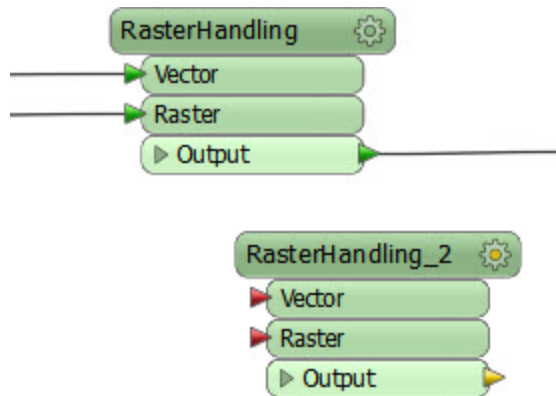
First Officer Transformer says…

'To arrange the ports in a set order, change their position in the custom transformer definition. The port that appears highest up in the canvas will appear first on the custom transformer itself!"

Exercise 3b Custom Transformer Re-use and Editing	
Scenario	FME user; City of Interopolis, Planning Department
Data	City Neighborhoods (KML)
Overall Goal	To set up an easy way for FME authors to calculate the average density of an item, for any set of polygon features.
Demonstrates	Custom Transformer re-use and editing
Starting Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise3b-Begin.fmw*
Finished Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise3b-Complete.fmw*

Having created a custom transformer in exercise 3a, we can now go on to make use of it multiple times and maybe even apply some edits.

1) Start Workbench

Open the workspace C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise3b-Begin.fmw

2) Duplicate Custom Transformer

Notice that we have one instance of the custom transformer, that is calculating population density, but it could be used a second time 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:
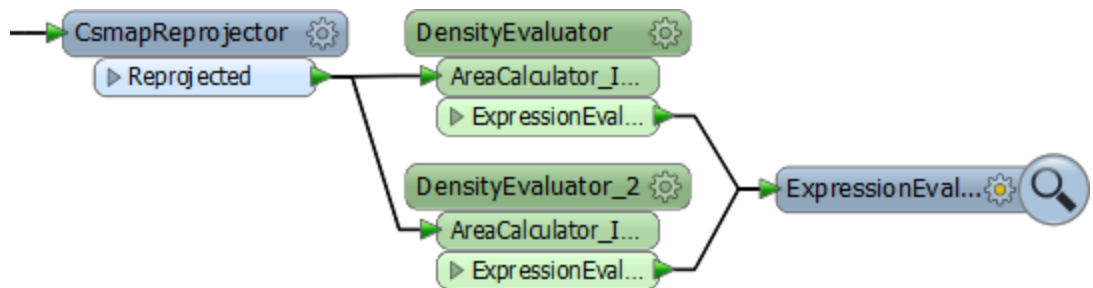
3) Set Custom Transformer Parameters

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.

5) Edit Custom Transformer

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.
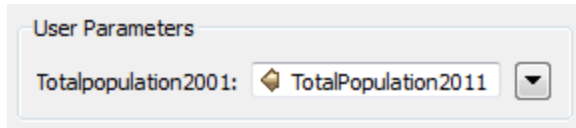
We can fix this by making the output name more generic. This is good practice for a custom transformer that might be used in multiple scenarios.

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 Destination Attribute parameter to DensityResult.
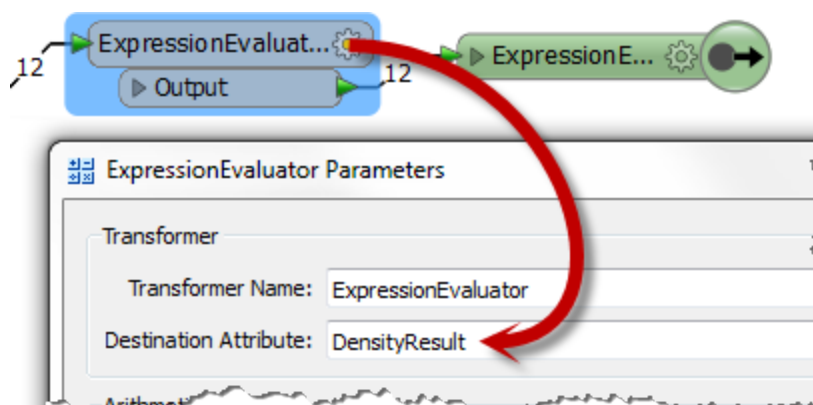
If you run the workspace again you'll notice that DensityResult is the attribute output by both instances of the custom transformer; i.e. one edit has fixed both of them!

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.

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:
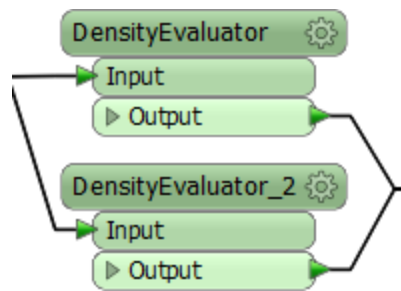
Custom Transformers and Schema

Custom Transformers and Schema

Attribute Schema
User Parameters
Automatic Schema Handling
Post-Creation Schema Handling
Manual Schema Handling
Handling Outgoing Attributes

FME Desktop Advanced Training 2015

Schema Handling is one of the most misunderstood components in Custom Transformers.

Attribute Schema

The main consequence of making a reusable custom transformer is that the author (and FME) cannot be sure where the transformer will be used and whether the schema will always match what is required.

For example, in this workspace a custom transformer carries out processing on incoming data using an attribute called AddressID as a key field.
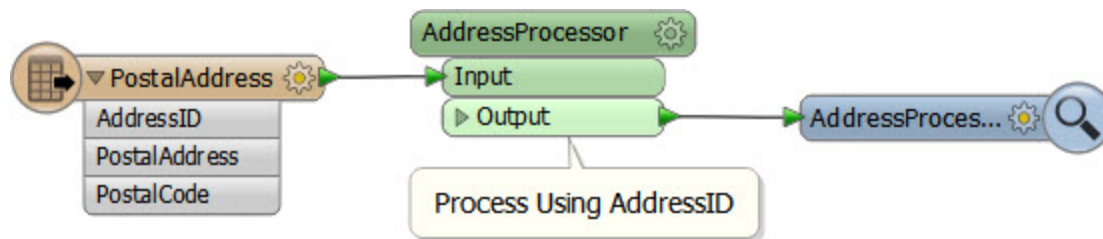
However, if – in the same workspace – that transformer is duplicated and connected to a different Reader feature type, there is no guarantee that AddressID will exist.

For example, here it is connected to a feature type where the field is in upper case, which will cause the workspace to fail when run:
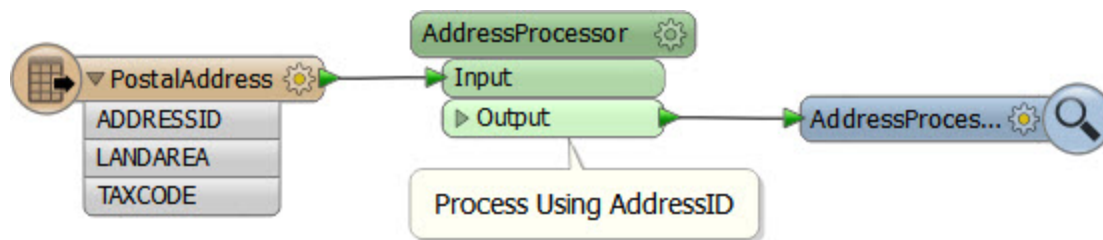
Therefore it's vital that there is some form of mechanism for protecting against problems of a mismatched schema of this type. In fact there are two ways this can be handled: FME can automatically take care of the schema, or the workspace author can handle it manually.

User Parameters

The other 'schema' items that need to be handled are published parameters.

If an FME transformer has been set up with a published parameter, and then that FME transformer is incorporated into a custom transformer, then there needs to be a way for that user input to be passed to the custom transformer.

For example, if a transformer with these parameters is used in a custom transformer, there still needs to be a method for the user to select values for these fields:

Automatic Schema Handling

When a custom transformer is created, one of the parameters in the Create Custom Transformer dialog is labelled Attribute References.
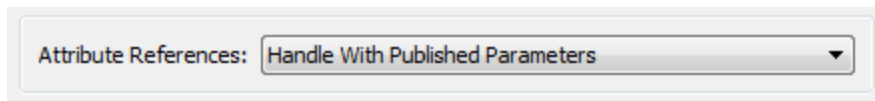
If this value is set to 'Handle with Published Parameters' then FME will automatically take care of attribute references in the custom transformer by turning them into published parameters. These published parameters become exposed in the parameters dialog for the custom transformer meaning they can be set wherever the transformer is used.

For example, here is a set of FME transformers for styling data as KML. In effect it is a cut-down version of the KMLDiagrammer transformer available on the FME Store.
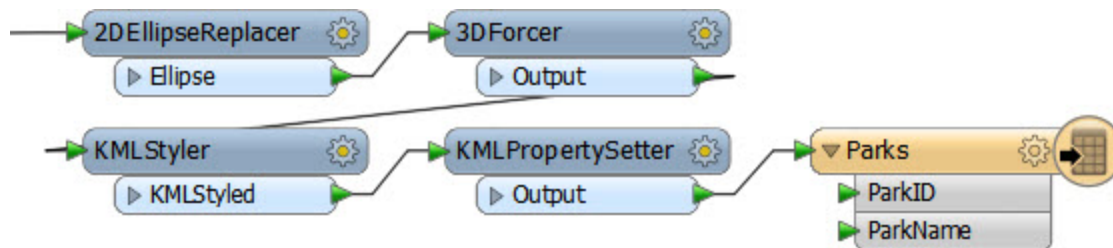
Incoming data is basically turned into a 3D column, where the width (x and y axes) and height of the column are set by attribute values, and the color through a published parameter:
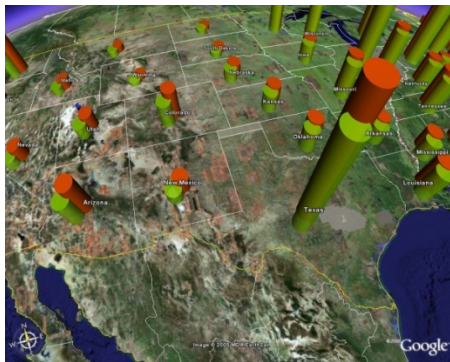
In this instance the attributes used are ParkCycleVisitors and ParkHikingVisitors (for the column x/y width) and TotalParkVisitors for the height (z axis extrusion) of the column:

The workspace author decides to turn this into a custom transformer, so he can use the same technique on other data, such as election results or planning applications. However, the question is, how can the transformer be set up to use an attribute called ElectionTurnout, when at present it is expecting TotalParkVisitors?

The answer is to use the 'Handle with Published Parameters' setting when creating the transformer:

Turn the above into a custom transformer and - with this new parameter set – the result is a transformer like this:

The attributes being referenced by the custom transformer have been exposed as published parameters, so that wherever the custom transformer is used, the user has the option to select their own attributes.

They don't have to rename their attributes to TotalParkVisitors, for example, in order to use the transformer. Also notice that the existing published parameter – Output Color – has also been exposed in this dialog, solving the second part of the schema handling conundrum.

If the user then opens the custom transformer definition and examines the Navigator Window, what they will see is as follows:

This illustrates how FME has automatically solved the attribute reference problem using published parameters. To make the custom transformer more generic, the workspace author will probably want to change the prompts on these parameters – and maybe reorder them – so they don't refer only to the current parks scenario:

Now, when used, the custom transformer has prompts that are a lot more generic, and will make sense when it is used elsewhere:
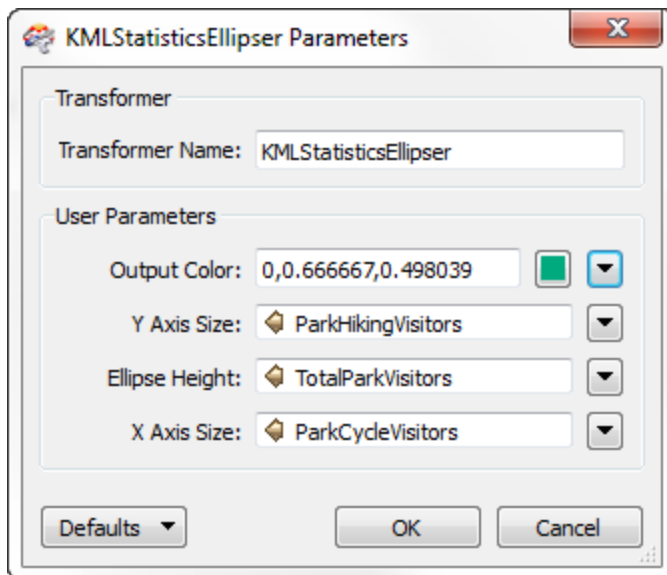
Post-Creation Schema Handling

The 'Handle with Published Parameters' setting is available when a custom transformer is created, but there also needs to be a mechanism for handling edits to a custom transformer, or where the custom transformer is created from scratch.

This is achieved using a schema-editing button on the custom transformer.

In a custom transformer definition, notice that input ports have a cog-wheel parameters icon, just like most other canvas objects:

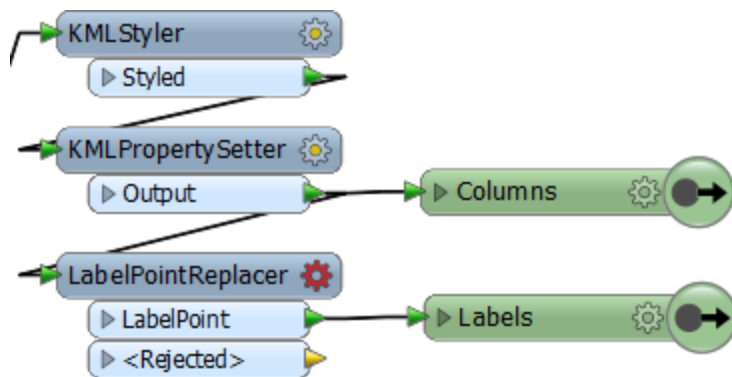Clicking this button opens a dialog in which the incoming schema can be defined.

For example, here our workspace author adds a new transformer to a custom transformer definition in order to create label features:

They've also added a separate output port – and renamed them too – which is an excellent show of FME Best Practice!
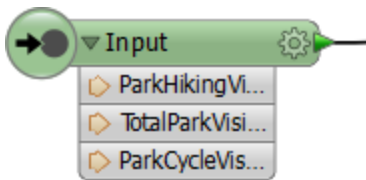
The author would like to use the Park Name attribute (ParkName) as the content for the label.

However, they cannot do that since ParkName was not used by the custom transformer when it was created, therefore FME did not need to expose it automatically:

That attribute is literally not available inside the custom transformer definition.

The solution is to click the icon to open the Input port parameters, and expose ParkName there:
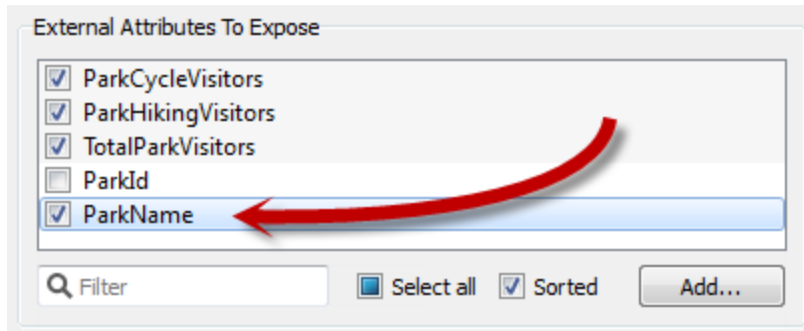
Now ParkName becomes available for use inside the custom transformer:
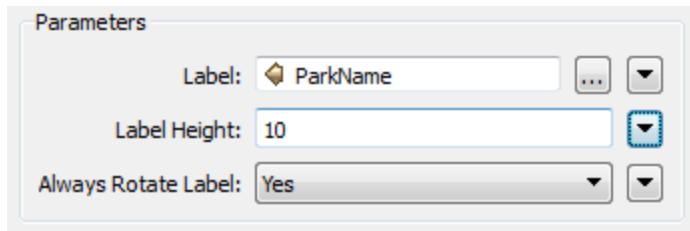
Additionally, back on the main canvas, the custom transformer has a newly exposed parameter ready to accept an attribute to use as the label content:
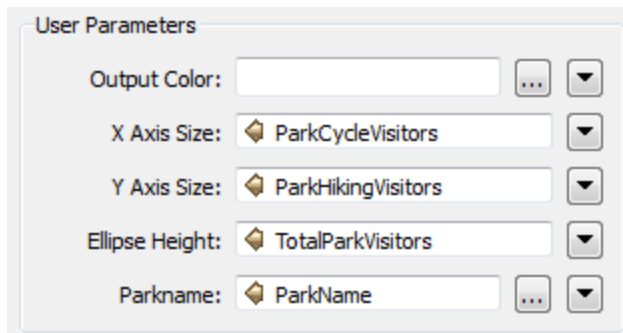
First Officer Transformer says…

"Basically, published parameters are the aerodynamic solution for handling schema inside a custom transformer. This automated behavior makes it easier for a new user, as they can use custom transformers without having to know about published parameters and how they operate.

Think of it as an auto-pilot for custom transformers. It's a perfect introduction for new users, and even experienced users will find it a nice break from flying manually.

I like to call it George!"

Exercise 3c Custom Transformer Schema Handling	
Scenario	FME user; City of Interopolis, Planning Department
Data	City Neighborhoods (KML)
Overall Goal	To set up an easy way for FME authors to calculate the average density of an item, for any set of polygon features.
Demonstrates	Custom Transformer Schema Handling
Starting Workspace	*C:\FMEData\Workspaces\DesktopAdvanced\Exercise3c-Begin.fmw*
Finished Workspace	*C:\FMEData\Workspaces\DesktopAdvanced\Exercise3c-Complete.fmw*

Having created a custom transformer in exercises 3a and 3b, we should check that the schema is going to be handled correctly.

1) Start Workbench

Open the workspace C:\FMEData\Workspaces\DesktopAdvanced\Exercise3c-Begin.fmw

If you check back to exercise 3a, you'll see that the transformer was created using the automatic schema handling parameter and the workspace will already be handling schema properly. That's how we were able to duplicate the transformer in exercise 3b and select one of the two attributes.

2) Set Parameter Prompts

One issue outstanding is that the prompt used by the transformer is not very generic. Let's fix that. Click on the tab labelled DensityEvaluator to switch the canvas to the custom transformer definition. Browse the Navigator window to find the published parameter that FME created.

Right-click on the parameter and choose Edit Definition. In the dialog that opens set the parameter prompt to: Density Attribute
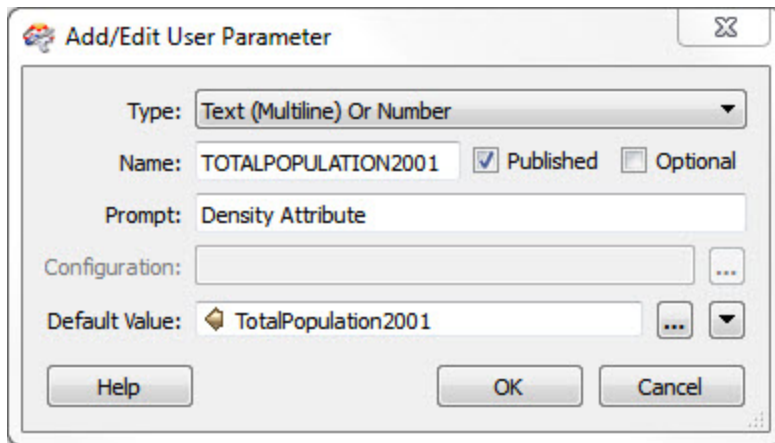
3) Implement Units Selection

At the moment this workspace is calculating the number of residents per square kilometer of land. However, other uses of this transformer might find different units to be more useful. Therefore we'll implement a method for users to be able to select units.

A user parameter will be the best way to achieve this, as the user can select the units manually and not need to supply it as an attribute.
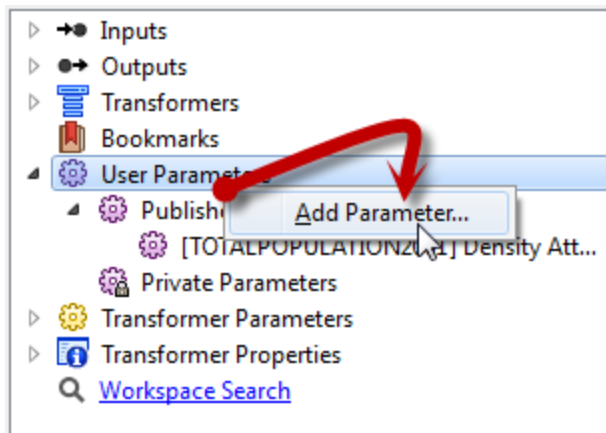
So, in the custom transformer definition, browse the Navigator window and right-click on the entry labelled User Parameters.

Select the Add Parameter option:

In the Add/Edit User Parameter dialog, set the following parameters:

Type	Choice with Alias
Name	DensityUnits
Prompt	Density Units

Uncheck the check box parameter labelled 'Optional.'

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. The first is a value of 1 with a display name of 'Sq Metres.'

The second is a value of 0.000001 with a display name of 'Sq Km.' Click OK to close that dialog.

Back in the Add/Edit User Parameter dialog set Sq Km as the Default Value.

Then click OK to close this dialog and add to the published parameter.

4) Implement Parameter

Now we've defined a published parameter, we have to use 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.
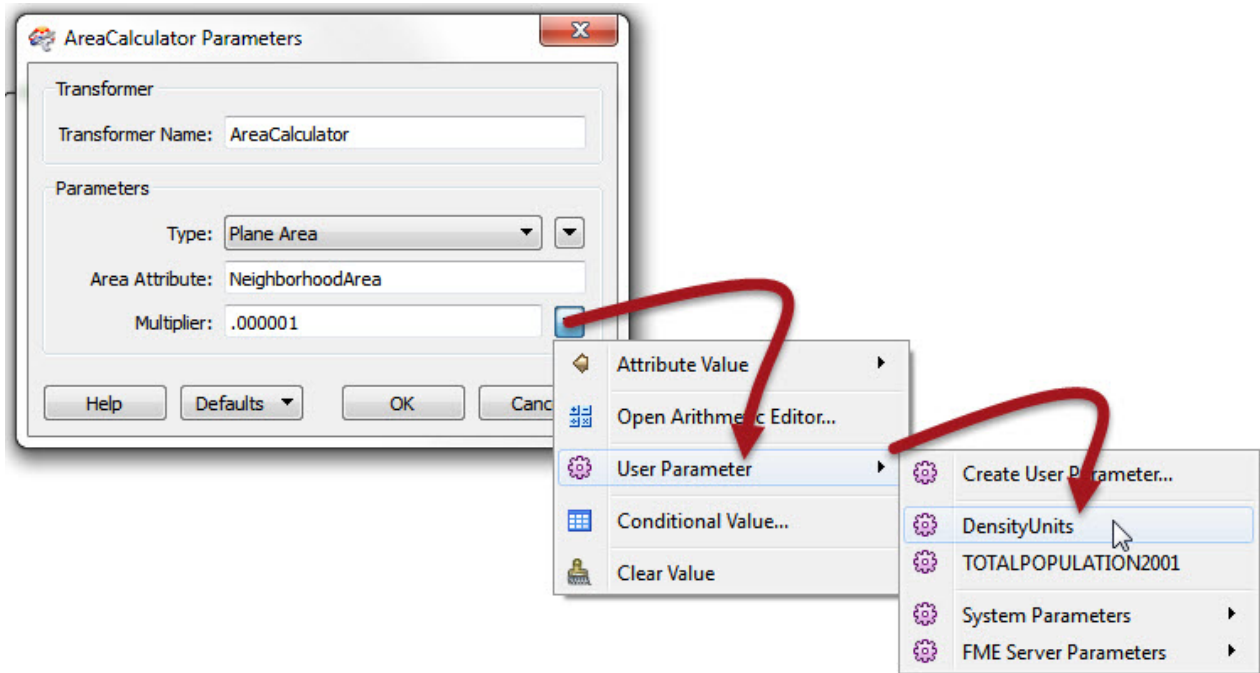
Click OK to close the dialog.

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.

5) Implement Weighting

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.

The weighting will come from an incoming attribute, which means we need to be able to handle this in the custom transformer's schema.

Because we don't have a weighting attribute here, return to the Main canvas tab and add a RandomNumberGenerator transformer in order to generate a test attribute:
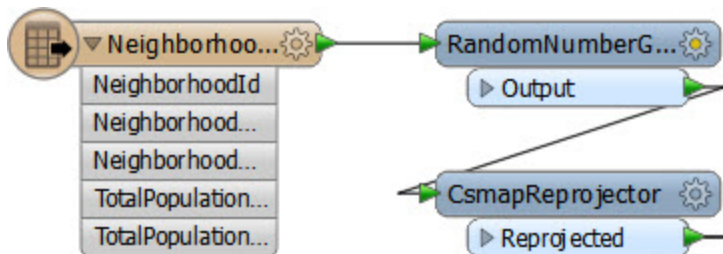
6) Set RandomNumberGenerator Parameters

Open the parameters dialog for the RandomNumberGenerator. For the purposes of this exercise set:
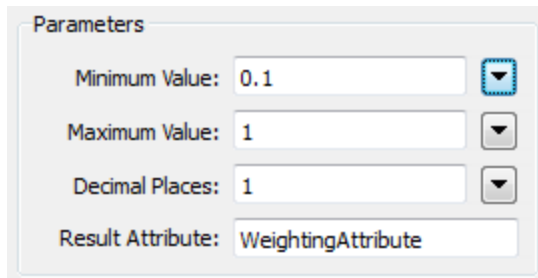
Minimum Value: 0.1

Maximum Value:1

Decimal Places: 1

Result Attribute: Weighting Attribute

7) Expose Attribute in Custom Transformer

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.
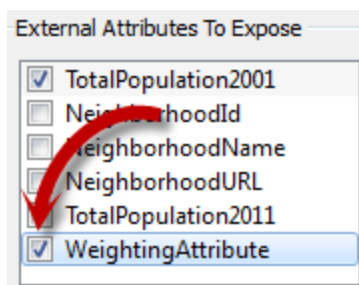
Put a checkmark against the WeightingAttribute attribute and then click OK to close the dialog.
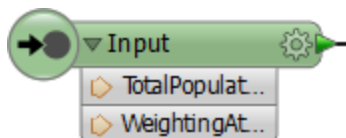
This will cause the attribute to be exposed in the custom transformer definition:

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.

Name: WEIGHTINGATTRIBUTE   ☑ Published ☑ Optional

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).

8) Duplicate ExpressionEvaluator

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:
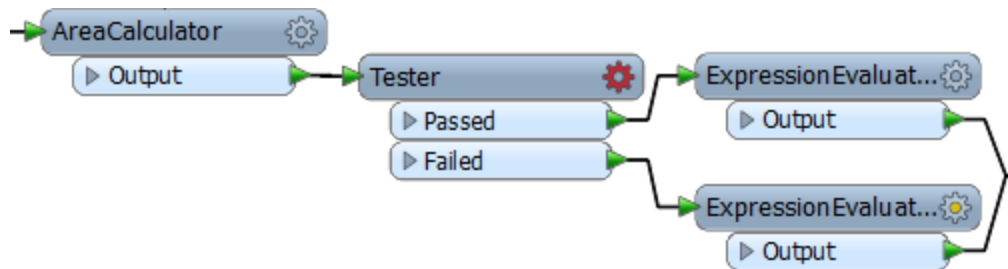
9) Set up Tester

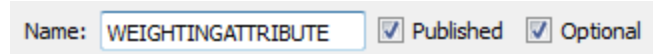Open up the Tester parameters dialog. Make a test for where WeightingAttribute > 0

	Left Value	Operator	Right Value	Negate	Mode
1	◈ WeightingAttribute	>	_k_ 0	☐	Automatic

10) 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:
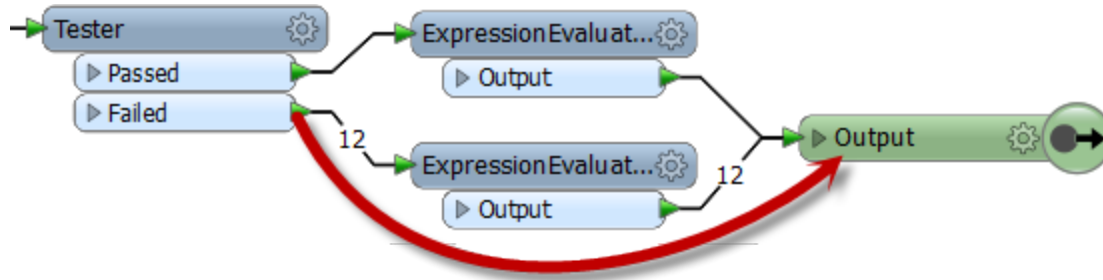
First Officer Transformer says…

"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."

Manual Schema Handling

For full control over how the attribute schema is handled in a custom transformer, the workspace author should create it using the setting 'Advanced – Fix Manually'.
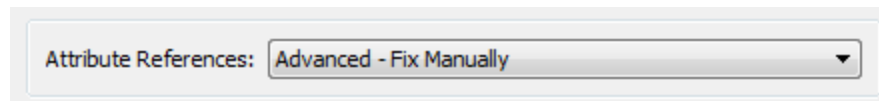
At that point the custom transformer has no published parameters, and therefore no user-definable options:
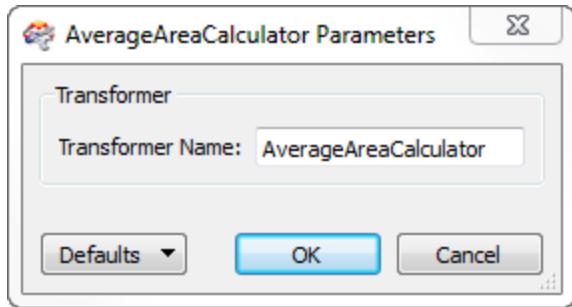
To expose a parameter or attribute, the workspace author must go to the transformer definition and manually create a published parameter:
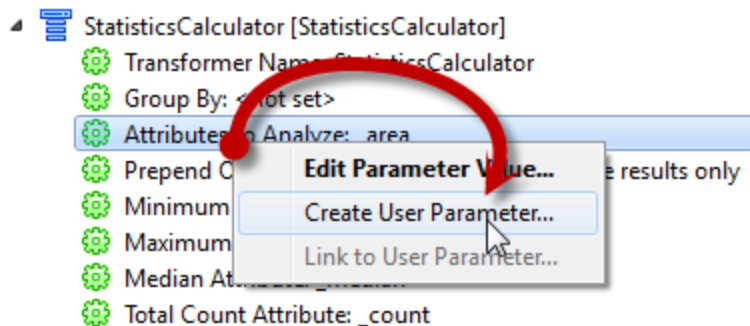
Then the parameter must be selected wherever it is going to be used:
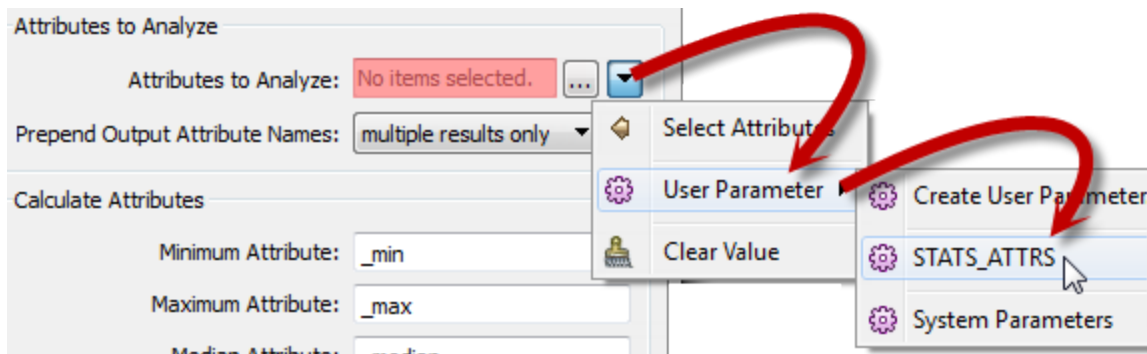
Compared to the automatic process, in manually defined schema handling it's more obvious – to an experienced user – what is happening. However, it's not such an easy process for a newer user to be able to set up.

Handling Outgoing Attributes

Besides incoming attributes, there is also the question of what attributes should emerge from the output of a custom transformer.
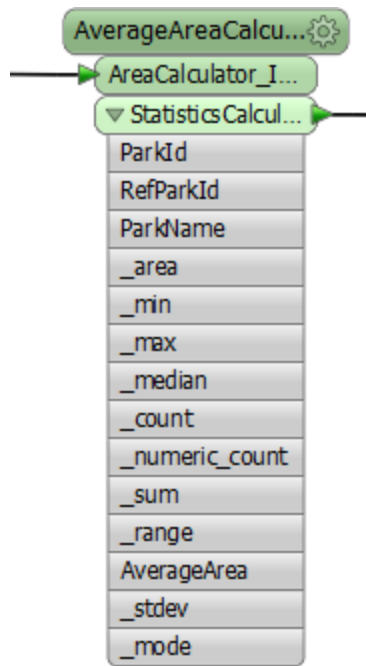
Best Practice suggests that we really don't want to output more attributes than are expected by the user.

For example, we would want to 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 the custom transformer is calculating the average area of a number of polygon features. The required output is an attribute called AverageArea.

However, 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:
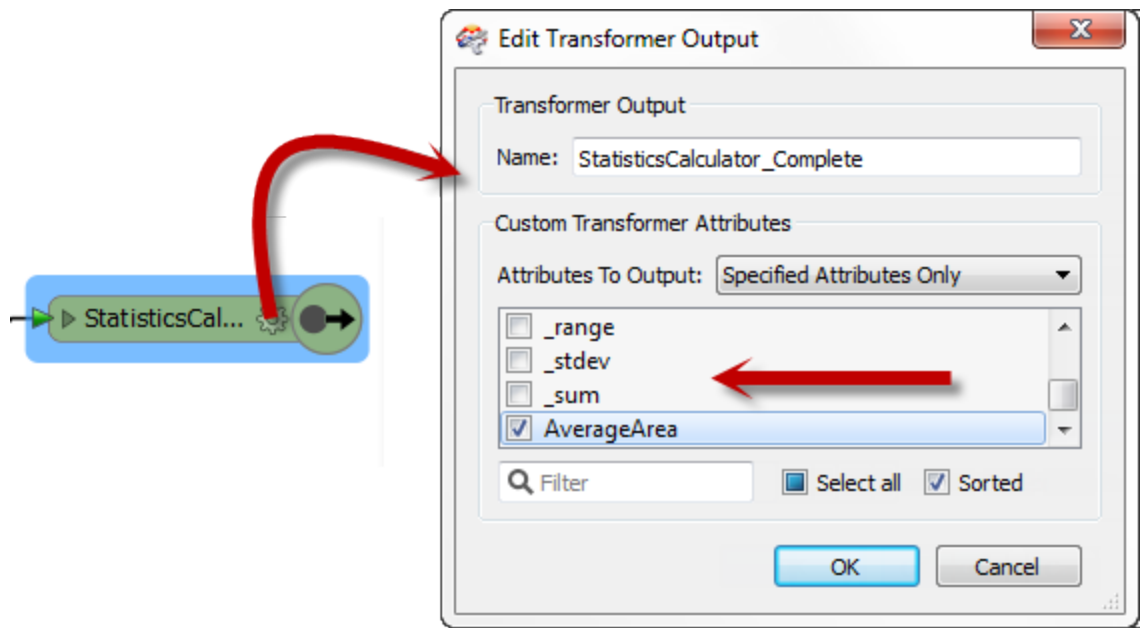
First Officer Transformer says…

"One suggestion is to prefix all temporary attributes inside a custom transformer with _temp. Then you'll know which to keep and which to clean up."

Exercise 3d Custom Transformer Manual Schema Handling	
Scenario	FME user; City of Interopolis, Planning Department
Data	City Neighborhoods (KML)
Overall Goal	To set up an easy way for FME authors to calculate the average density of an item, for any set of polygon features.
Demonstrates	Manual Schema Handling
Starting Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise3d-Begin.fmw*
Finished Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise3d-Complete.fmw*

Let's go back to the workspace – before creating the custom transformer – and this time create all our references manually.

1) Open Workspace

Open the workspace *C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise3d-Begin.fmw*.

As you'll see, this is the same as Exercise3a-Begin.fmw – we have a workspace ready to create a custom transformer.

2) Create Custom Transformer

As in Exercise 3a, 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.

This time, set the Attribute References parameter to "Advanced – Fix Manually" and click OK.

The custom transformer will now be created.

Name:	DensityEvaluator
Category:	Calculators
Description:	A custom transformer for the FME training course

▷ ☐ Use Extended Description

Attribute References: Advanced - Fix Manually
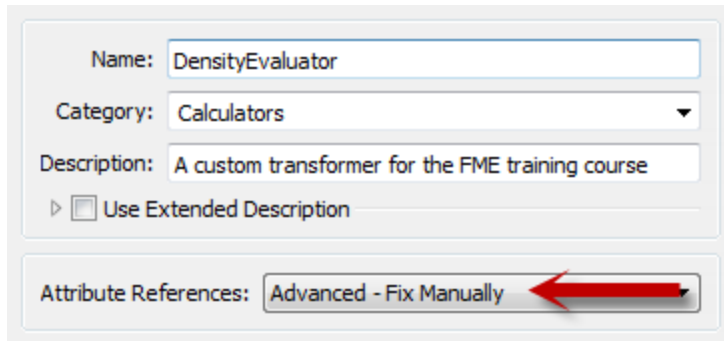
3) Create Published Parameter

Examine the custom transformer and this time you'll see there are no published parameters.

ExpressionEvaluat... ⚙
▷ Output
ExpressionE... ⚙
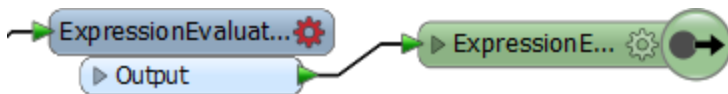
In fact, the ExpressionEvaluator will have a red parameters button, because it has no access to the required attributes:

So, in the custom transformer definition, locate the User Parameters section in the Navigator window. Right-click on the part labelled User Parameters and select Add Parameter.

In the dialog that opens, define a new published parameter as follows:

Add/Edit User Parameter

Type:	Attribute Name
Name:	DensityAttr ☑ Published ☐ Optional
Prompt:	Density Attribute:
Configuration:	
Default Value:	

Help   OK   Cancel
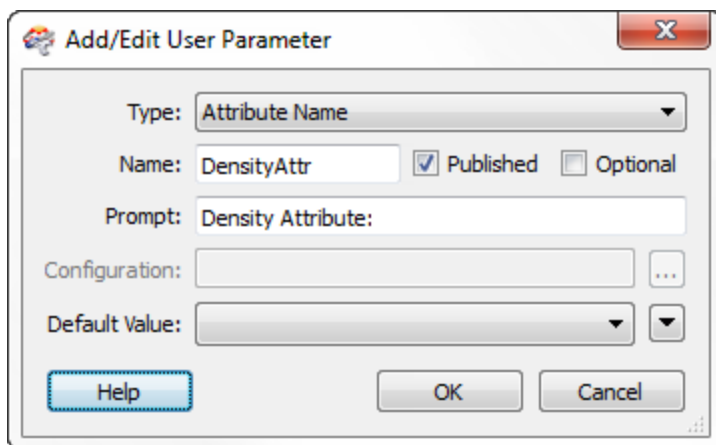
Type:Attribute Name

Name:DensityAttr

Prompt:Density Attribute

The Published checkmark should be turned on, but turn off the Optional checkmark, as this is a compulsory parameter for this transformer:

Click OK to create the parameter.

4) Apply Published Parameter

The parameter we need is now created, but not yet being applied. To do so open the parameters dialog for the ExpressionEvaluator.

You'll see the problem in the expression; TotalPopulation2001 is highlighted in red because it is not available to the transformer:

@Value(TotalPopulation2001)/@Value(NeighborhoodArea)

So, strip out the TotalPopulation2001 part of the expression:

@Value()/@Value(NeighborhoodArea)

Now add a reference to the newly created user parameter.

This can be done by locating the parameter in the operators panel and double-clicking it:
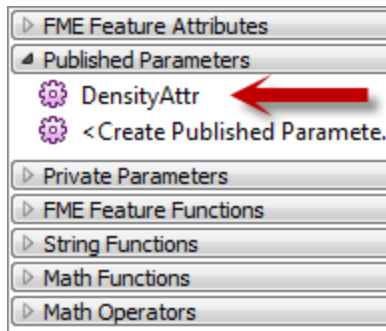
This will create an expression like so:

@Value($(DensityAttr))/@Value(NeighborhoodArea)

You should also change the Destination Attribute from PopulationDensity2001 to DensityResult.

Click OK to accept the new expression and close the dialog. Notice that the ExpressionEvaluator parameters button turns from red to blue, meaning all is now correct.

> *First Officer Transformer says…*
>
> *"In step 4 the @Value() part of the expression needs to remain, because the parameter only returns an attribute name, not a value.*
>
> *It's one reason why automated schema handling is easier to use."*

5) Check Parameter

Return to the main canvas window. The DensityEvaluator parameter button is marked red, but that's OK because we haven't yet picked an attribute to process.

Click the parameter button and select TotalPopulation2001 as the attribute to process.
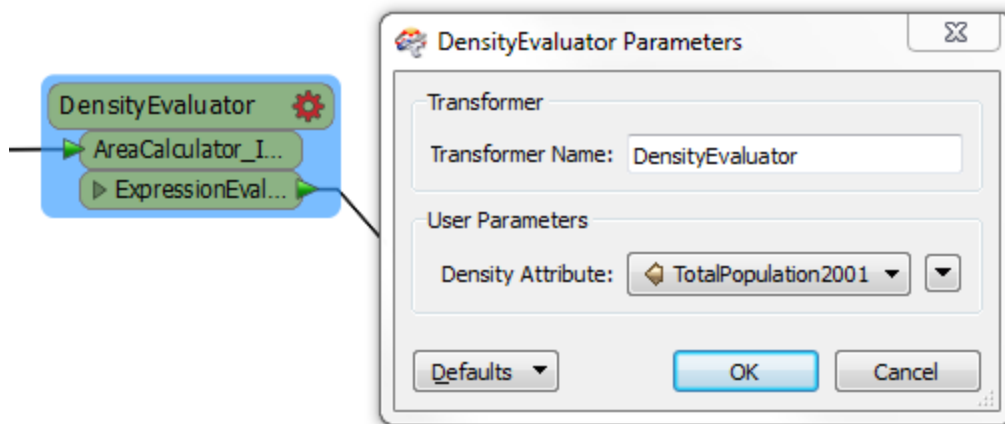
6) Add Second Custom Transformer

Now delete the remaining ExpressionEvaluator from the main canvas as we no longer need it.

Use Quick Add to add a second DensityEvaluator instance. Connect it as a second connection from the CsmapReprojector and select TotalPopulation2011 as the attribute to process.
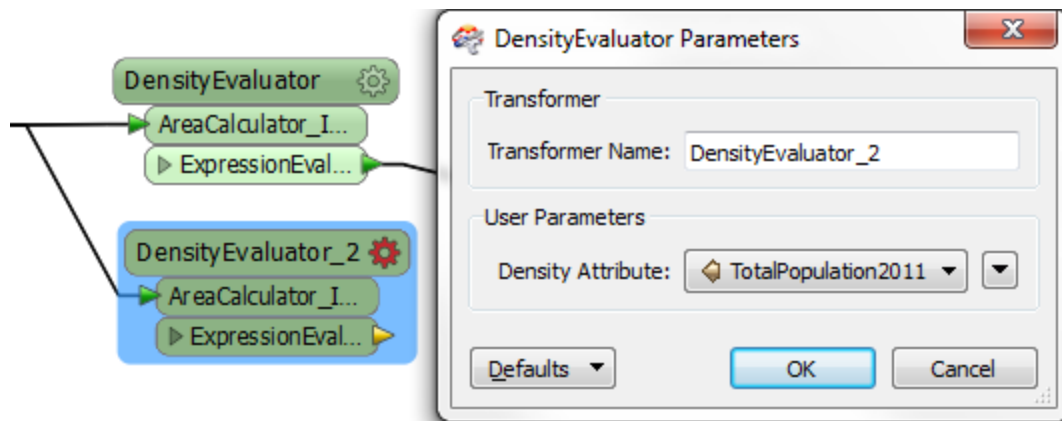
7) Run Workspace

Add an Inspector transformer to each of the custom transformers and run the workspace. Check the output to ensure the results are correct.

Custom Transformer Types

Custom Transformers come in two flavors: Embedded and Linked.

Custom Transformer Types

There are two types of custom transformers: Embedded and Linked.

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:

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 and have the advantage of needing no external files, as their definition is embedded into the workspace. However, an embedded 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

As a separate file, linked transformers are perhaps a little harder to understand and manage. However, a linked custom transformer is much easier to share among users: any number of users can use the same custom transformer definition. Additionally, maintenance is easier too because any changes made to the definition are automatically propagated to any workspace that uses it.

Exporting a 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:
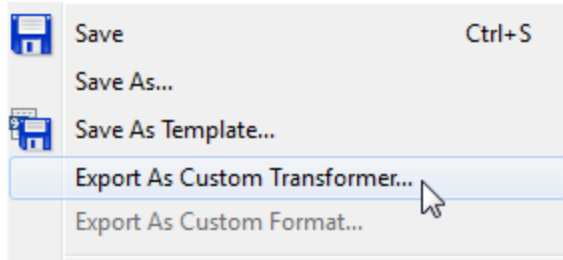
At this point a dialog opens in which you can confirm the transformer name and category, plus some other parameters including the save location:
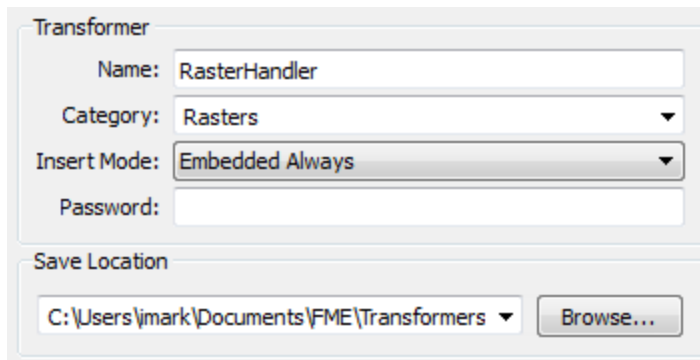
Clicking OK creates the custom transformer and opens it in a new instance of Workbench.

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.

When a linked transformer shows up in the transformer gallery – or Quick Add dialog – then it has a special icon to signify that you are about to use a linked version, rather than the embedded.
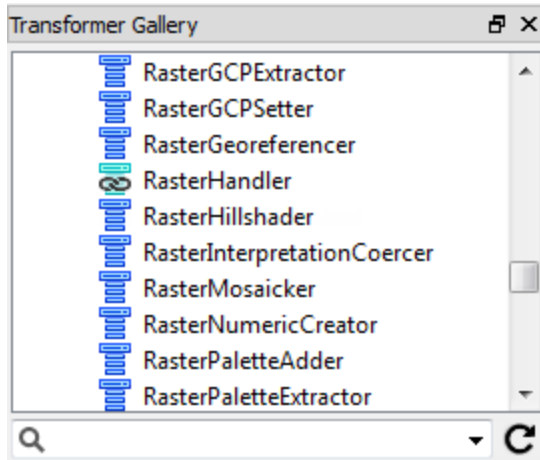
First Officer Transformer says…

"Note the refresh button on the transformer gallery. If a custom transformer doesn't show up when you expect, refreshing the gallery contents may help to locate it."

Insertion Mode

When exporting a custom transformer, the Insert Mode parameter specifies how the custom transformer can be used. There are four different modes.
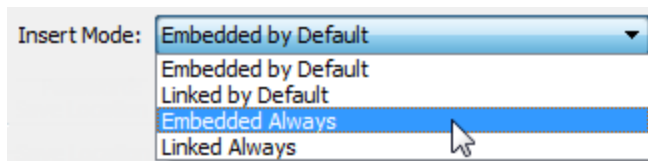
If a "by Default" option is used, then any instance of this transformer that is placed can be switched back and forth from embedded to linked.

If an "Always" option is used, then any instance of this transformer is fixed in either embedded or linked mode, and cannot be switched.

Which Mode to Use

When an author creates a custom transformer for a customer or user who is inexperienced in FME, then Embedded Always is a good choice, since it is easier to manage (no external files) and the user cannot accidentally change it.

When the custom transformer is intended to be shared among users, then Linked Always is a good choice, as any changes to the transformer definition are automatically applied to the end workspaces.

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.

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.

Similarly, to switch from Linked to Embedded simply right-click and choose the option Embed.

Of course, to be able to switch types requires that you have already exported the custom transformer!
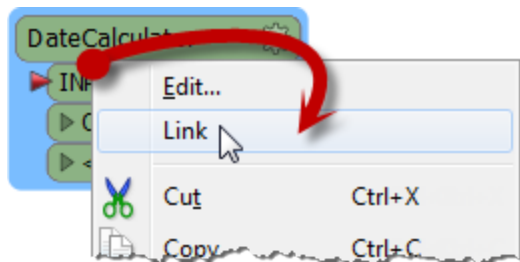
First Officer Transformer says…

"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."

Password

The password field (in the Export Custom Transformer dialog) 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, in case you wish to make edits!

Licensing

On the same topic, custom transformers can be licensed so that they cannot be used without the proper registration code. 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.
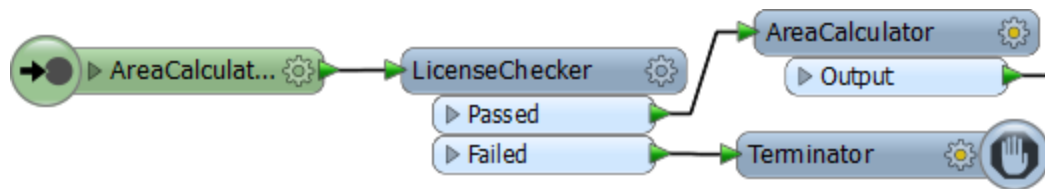
First Officer Transformer says…

"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."

For more information on obtaining licenses for a custom transformer, contact the Safe Software support team at www.safe.com/support.

First Officer Transformer says…

"Did you know that it is possible to nest one custom transformer inside another?

For example, download the PolylineAnalyzer custom transformer from the FME store and you'll find inside its definition it uses other custom transformers such as the VertexCounter and AzimuthCalculator. Look at the VertexCounter definition and you'll find it uses a custom transformer called the LoopFilter!

So nesting transformers – whether linked or embedded – is a perfectly valid technique."

Custom Transformer Versioning

Custom Transformer Versioning

Why use Versioning?
Creating a Versioned Custom Transformer
Editing a Specific
    Transformer Version
Updating a Transformer Version

FME Desktop Advanced Training 2015

FME includes functionality so that any single custom transformer can exist in a number of versions.

Why use Versioning?

Versioning means a single custom transformer can exist in any number of versions.

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.

However, a more important advantage relates to FME releases and new functionality.

For example, a custom transformer shared by many users could be updated to use new behaviour in FME 2015. If that custom transformer is versioned then the previous version remains available to users who have yet to update to 2015, while users who have updated to FME2015 can update their custom transformer to match.

Creating a Versioned Custom Transformer

Versioning can occur whenever you edit a linked custom transformer and then attempt to save it. In that situation a dialog will open to ask what you wish to do:
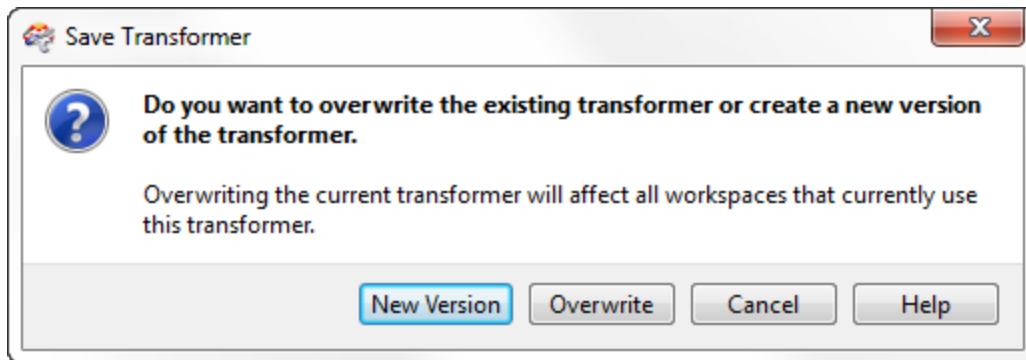
The two options are to overwrite the existing version or to create a new version. Creating a new version does not create a separate fmx file; instead it creates a separate version of the transformer in the same fmx file.

The title bar in Workbench also changes to illustrate that this is now a new version.

RasterHandlerLinked - Version 2 - FME Workbench

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."

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:
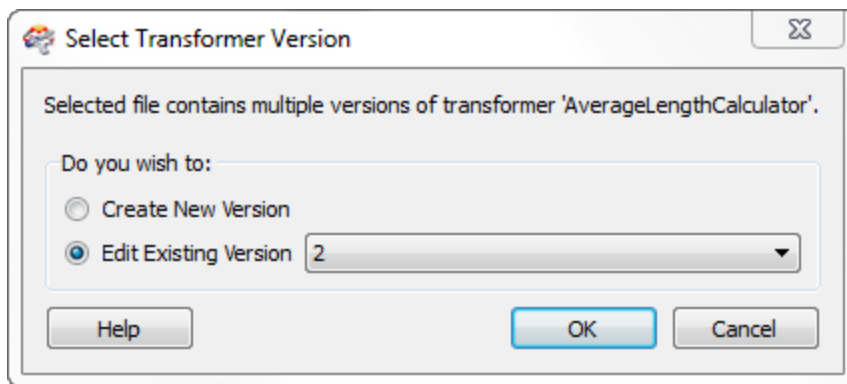
This way you are able to make edits on older versions, which is particularly useful when that version is tied to a particular FME release.

Updating a Transformer Version

Whenever a workspace contains a linked custom transformer, and FME detects that a new version is available, then an option appears on the context menu to allow an update to the new version:
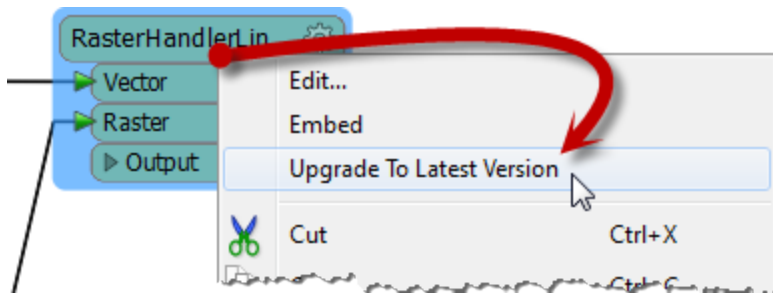
If you have the 'Show Transformer Version' option turned on under Tools > FME Options > Transformers, then the tooltip will now show the transformer version:
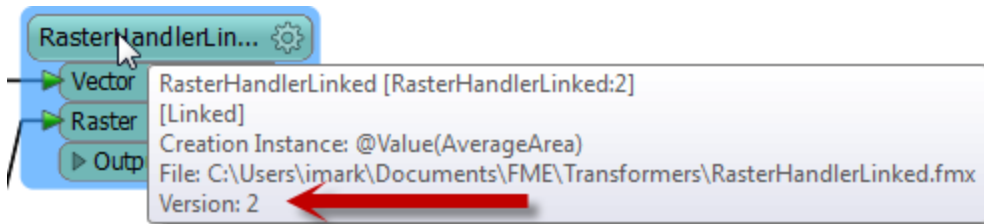
Note that if I was using a version of FME that is incompatible with the updates in version 2, then I would not be given the option to update.

Exercise 3e Custom Transformer Modes	
Scenario	FME user; City of Interopolis, Planning Department
Data	Bicycle Routes (Esri Shape)
Overall Goal	To create a custom transformer to calculate the average length of line features
Demonstrates	Linked Custom Transformers. Versioned Custom Transformers.
Starting Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise3e-Begin.fmw*
Finished Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise3e-Complete.fmw* *C:\FMEData2015\Workspaces\DesktopAdvanced\AverageLengthCalculator.fmx*

I notice that there's a custom transformer to calculate the average area of a number of polygon features. But there's no such transformer to calculate the average length of linear features.

Let's fix that situation by creating such a transformer.

1) Open Workspace

Open the workspace *C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise3e-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 workspace.

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 Length Calculator

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.
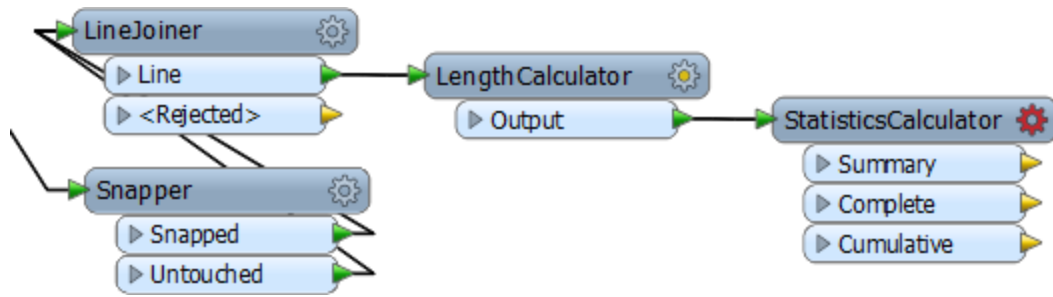
3) Create Custom Transformer

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:
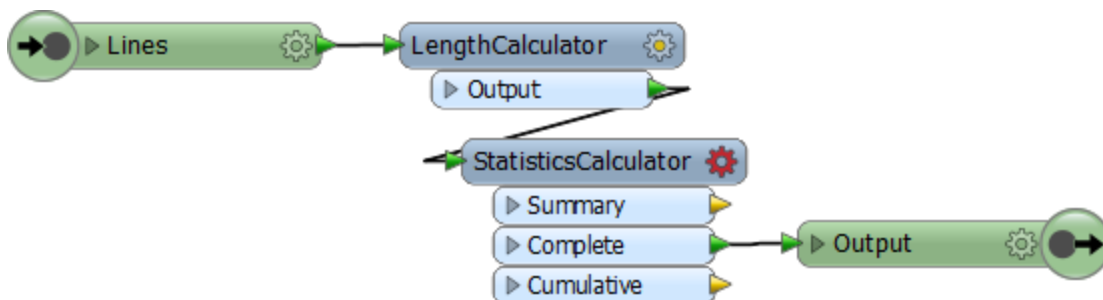
Now open the LengthCalculator parameters. Rename the Length Attribute to _temp_length.

Then open the StatisticsCalculator parameters. Set Attributes to Analyze to _temp_length

Clean up the output attributes by removing all of them except Mean Attribute – and that one should be renamed 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 _temp_length is not.

If you see other attributes (such as _max) then you obviously didn't clear them out of the StatisticsCalculator. That's OK. Just make sure they are unchecked in this dialog.
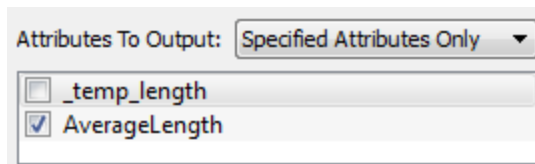
5) Run Workspace

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

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 (<user>\FME\Transformers).
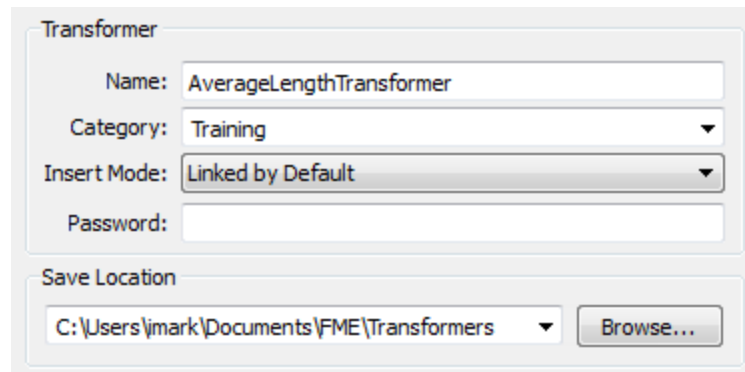
Click OK to close the dialog. The custom transformer is saved (as AverageLengthCalculator.fmx) and this file opened up in a new instance of FME Workbench.

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.
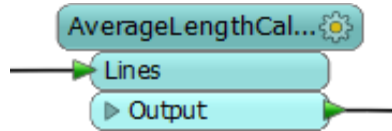
Notice that you can right-click and choose to embed the transformer, and then switch back to the linked version. In a real-life scenario, which you choose would usually depend on whether you are planning to share the transformer.

In embed mode, right-click the transformer and choose Edit. Then in the definition make a small change (like moving one of the objects). Back in the Main tab you'll find that you can no longer change back to Linked mode, because the two definitions are now different!

Delete the embedded transformer. You'll be prompted whether you wish to delete the definition too. Click Yes, but then add a new linked version back again.

First Officer Transformer says…

"It's important to realize that the definition of an embedded custom transformer can remain in the workspace, even if it's not used. Sometimes this is useful, sometimes not. You'll be able to tell if such a definition remains by looking in the Embedded Transformers section of the transformer gallery."

8) Examine Custom Transformer

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.
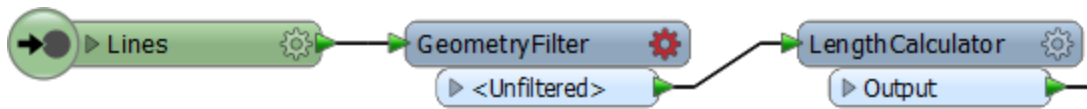
Open the parameters and select Line and Arc as the geometries to filter by. Then click OK to close the dialog.
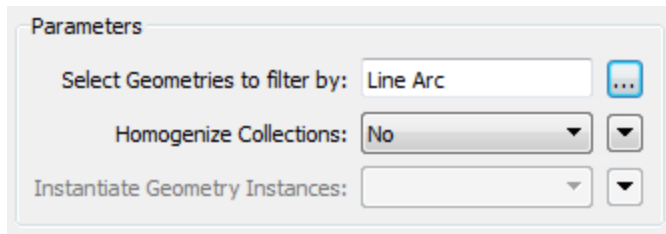
Adjust the feature mapping so that the Line and Arc ports are directed into the LengthCalculator. Add a second output port object by right-clicking on the canvas and selecting Insert Transformer Output. Call the newly placed port Rejected and connect the <Unfiltered> data to it, like so:
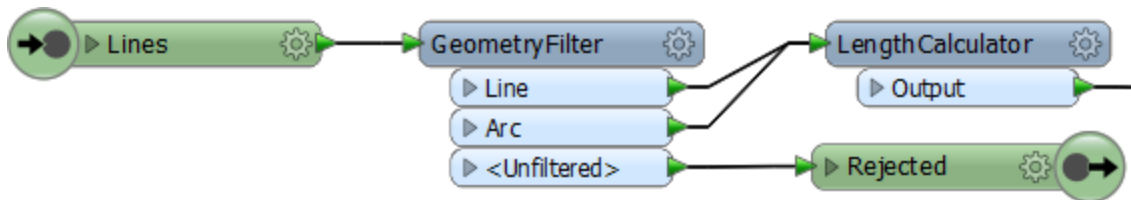
Now click the save button to save the custom transformer. You'll be prompted whether you want to create a new version. Click the button labelled New Version to do so.

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.
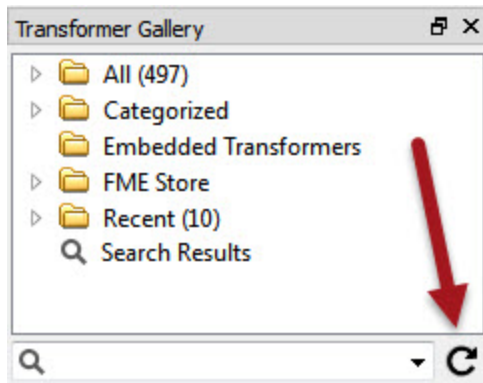
Now right-click on the AverageLengthCalculator custom transformer and there should be an option to upgrade to the latest version. Choose this option:
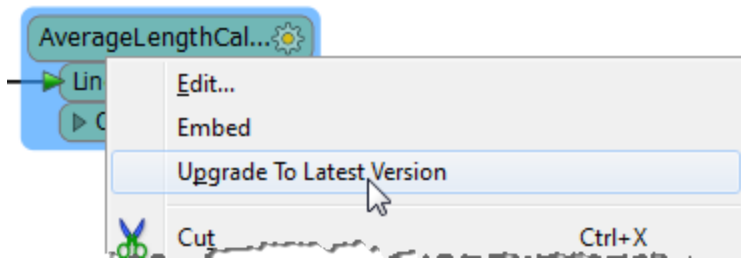
The transformer will be refreshed and updated, which you can tell by the presence of a Rejected port:
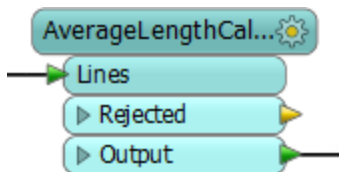
First Officer Transformer says…

"If you have an fmx file by itself, you can have FME install it into the correct folder by double-clicking on the file in a file browser.

Similarly, if you download a custom transformer from the FME Store, it will be stored in a specific location where it will be recognized by FME Workbench."

Custom Transformers and Parallel Processing

Custom Transformers and Parallel Processing

Setting up Custom Transformer Parallel Processing

FME Desktop Advanced Training 2015

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:

- Transformer Parameters
  - Advanced
    - Parallel Processing Level: No Parallelism
    - Parallel Process By: <not set>
    - Parallel Process Groups are Ordered: No

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:

- Transformer Parameters
  - Advanced
    - Parallel Processing Level: Moderate
    - Parallel Process By: (Linked to 'FME_PROCESS_GROUP_BY')
    - Parallel Process Groups are Ordered: No

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:
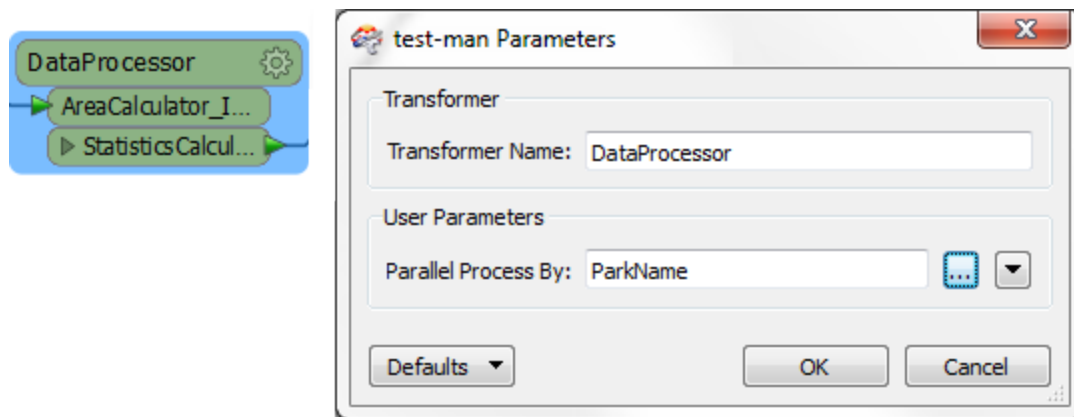
If, as an author, I don't want the end user to be setting the group-by, then what I can do is locate that published parameter, edit its definition, and unset the Published parameter:
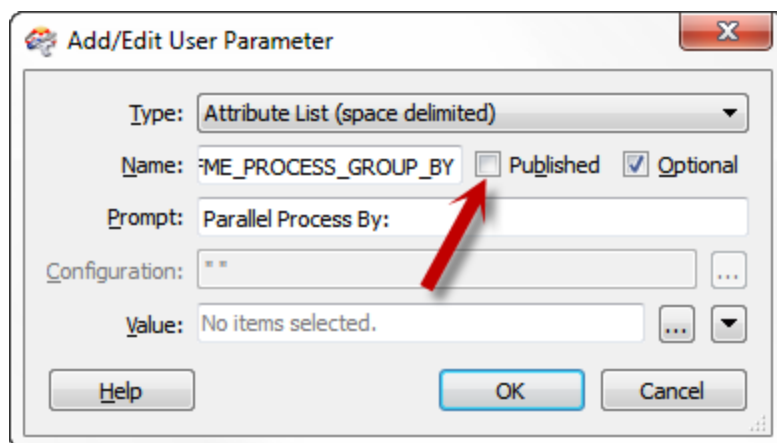
First Officer Transformer says…

"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 3f Custom Transformer Parallel Processing	
Scenario	FME user; City of Interopolis, Planning Department
Data	3D Point Cloud (ASPRS Lidar Data Exchange Format (LAS))
Overall Goal	Turn Point Clouds into Point features
Demonstrates	Parallel processing in custom transformers.
Starting Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise3f-Begin.fmw*
Finished Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise3f-Complete.fmw*

For this exercise we have been asked to convert point clouds to a vector point format that another department can use. We already have a workspace to do this, which nicely tiles and thins the data too so the destination datasets aren't overwhelmed.

However, the workspace takes time to run and it might be better to use parallel processing. Since none of the transformers used has a parallel processing parameter, we'll have to create a custom transformer.

1) Open Workspace

Open the workspace C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise3f-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)

Image Name	User Name	CPU	Memory (...	Description
System Idle Process	SYSTEM	70	24 K	Percentage of time the processor is idle
fme.exe *32	imark	12	66,988 K	FME EXE
explorer.exe	imark	10	109,864 K	Windows Explorer
fmedatainspector.exe *32	imark	00	213,180 K	FME Data Inspector
fmeworkbench.exe *32	imark	00	218,604 K	FME Workbench

Of course, you'll also see an fmeworkbench.exe process, which is the process running the Workbench interface. This isn't responsible for running a workspace; this is a separate process.

2) Create Custom Transformer

Now select the PointCloudThinner and PointCloudCoercer transformers and turn them into a custom transformer. Note; 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:
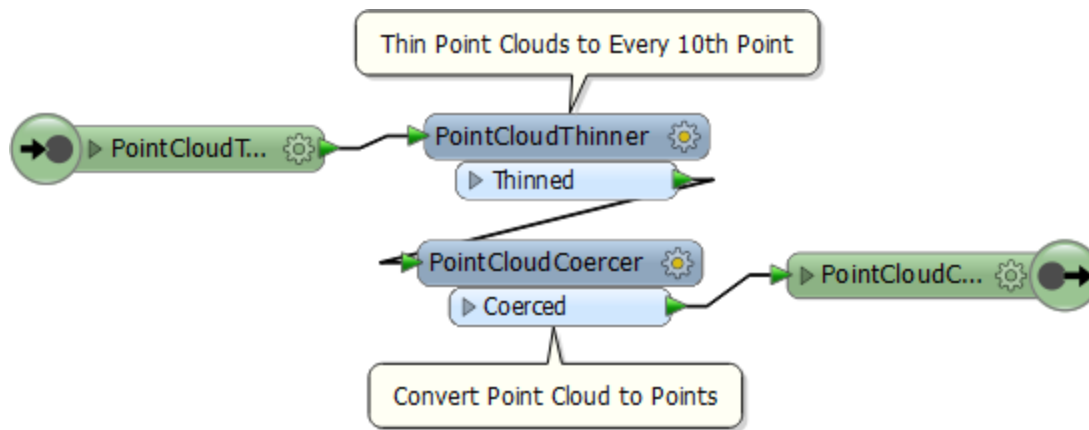
3) Set Parallel Processing

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.
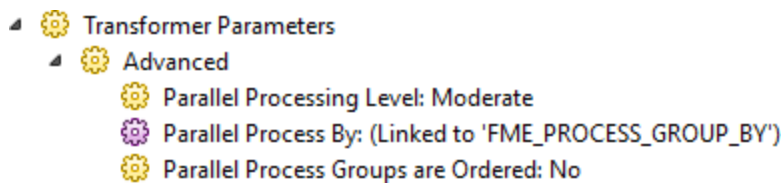
4) Set Process By

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.
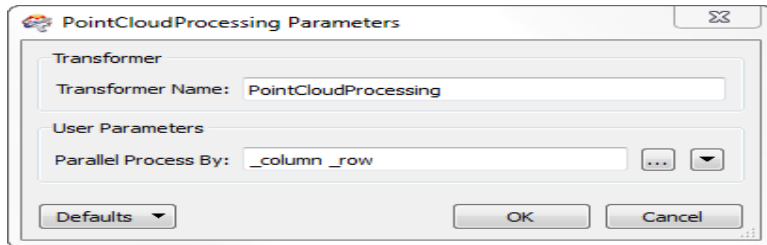
This means that each unique combination of _column and _row (i.e. each tile) will be run under a separate process, up to a maximum of one process per core processor.

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).

Image Name	User Name	CPU	Memory (...	Description
fmeworker.exe *32	imark	15	26,456 K	
fmeworker.exe *32	imark	15	26,460 K	FME EXE
fmeworker.exe *32	imark	14	26,460 K	
fmeworker.exe *32	imark	13	26,608 K	
fme.exe *32	imark	10	115,196 K	FME EXE
fmeworker.exe *32	imark	08	21,792 K	
fmeworker.exe *32	imark	04	12,648 K	
fmedatainspector.exe *32	imark	02	213,180 K	FME Data Inspector
fmeworkbench.exe *32	imark	02	202,304 K	FME Workbench

In moderate mode, you'll see up to one fmeworker process for each core. This time the translation should be complete is 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?

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.

> *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)."*

In FME, loops are only permitted inside a custom transformer.

> *First Officer Transformer says…*
>
> *"For an excellent example of looping in an FME workspace, check out the customer story at: www.fme.ly/LoopExample*
>
> *There the user uses a loop inside to place trees (the action) until a certain density is reached (the condition)."*

Setting up a Custom Transformer Loop

A loop in a custom transformer requires two components: the start and end point of the loop.

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:
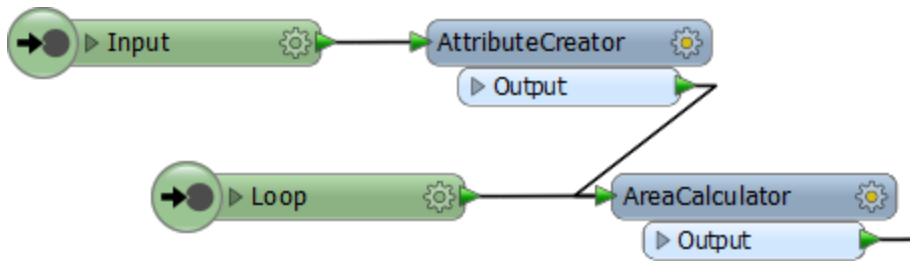
If you don't want the Loop port to appear on the transformer itself, like this:
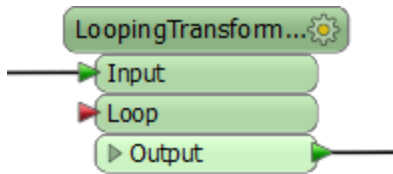
Then you simply have to open the Input port's parameters and 'unpublish' it:
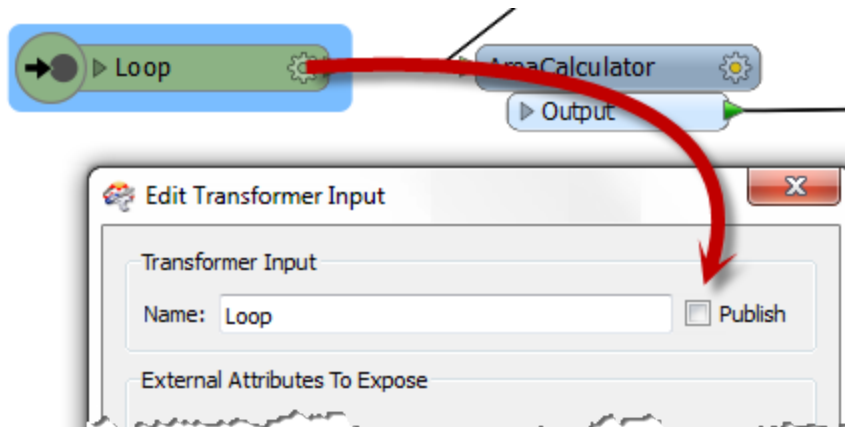
The end of a loop is identified by a Loop object. You can insert one by selecting it from the canvas context menu in a custom transformer:
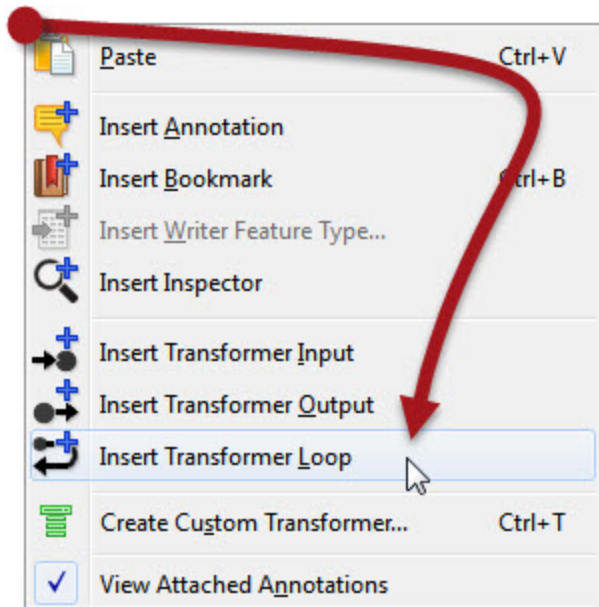
When you place a loop object you are asked which Input object is to be looped to:
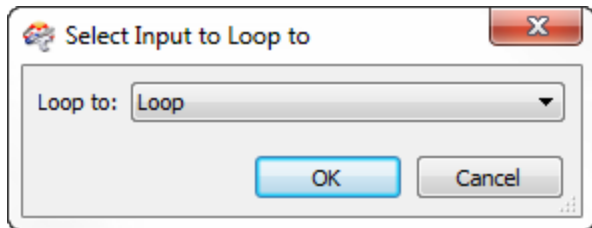
And then the loop is complete:
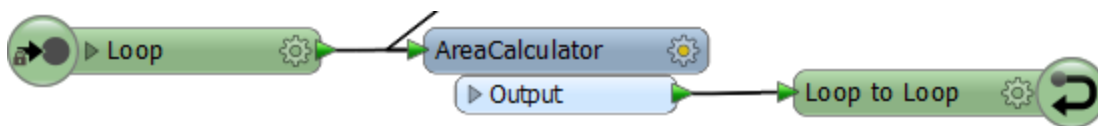
Of course, in most cases the loop needs a condition to stop processing. For example you might use a Tester to test for a particular case and then continue looping on a failure but exit when the test is successful:
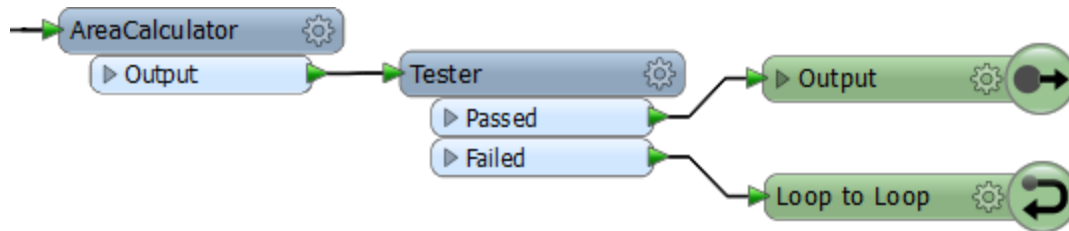
If you do create an infinite loop, FME won't run forever, and will eventually stop the process.

Loops and Custom Transformer Type

Loops can be implemented in both embedded and linked custom transformers. However, for technical reasons, blocking – or group-based – transformers can only be used in a linked transformer.

If you attempt to create a loop inside an embedded custom transformer that includes a group-based FME transformer, then you will receive an error message and be prompted to export the custom transformer.
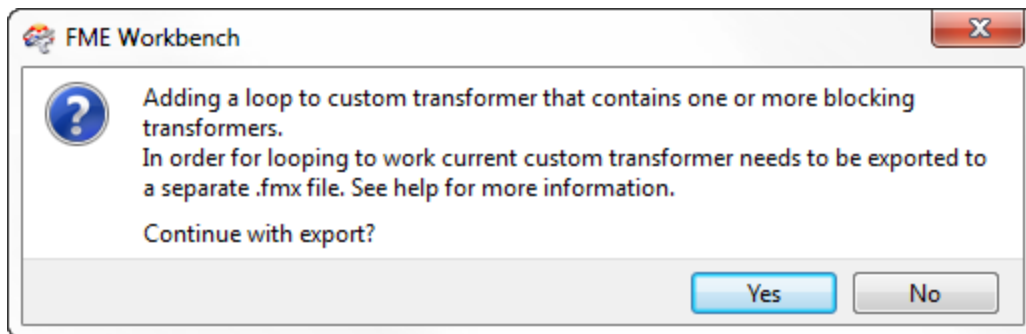
A linked custom transformer has a particular parameter (in the Navigator window) called Enable Blocked Looping:

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.

> *First Officer Transformer says…*
>
> *"Generally you only use a loop to repeat an action inside it. One-off actions should take place outside of a loop. For example, I wouldn't put my wheels up and down every time I circled an airport. That would be very inefficient. I'd do it once, either before I start, or after I finish, looping"*

Exercise 3g Custom Transformer Loops	
Scenario	FME user; City of Interopolis, Planning Department
Data	City Neighborhoods (KML) City Trees (CSV)
Overall Goal	Create a list of the top ten trees per neighborhood
Demonstrates	Custom transformer loops.
Starting Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise3g-Begin.fmw*
Finished Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise3g-Complete.fmw*

A colleague is trying to calculate the top ten tree species per neighborhood in the city of Vancouver. They've got to the point where they can merge the neighborhood and tree datasets together – and even create an FME list of the most-populous trees – but they are having problems extracting information from that list and turning it into attributes.

Having taken FME training(!) you realize this can be done using a custom transformer loop.

1) Open Workspace

Open the workspace *C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise3g-Begin.fmw*.

This is as far as your colleague got. Run the workspace to see what the output is. It will take about 90 seconds to complete (if your system is slower, or you just want it to go faster, put a Sampler transformer before the StringCaseChanger, to cut down the amount of trees being processed).

The output looks like this. It's the result of the ListHistogrammer that produces a nice, sorted list of the number of tree types per neighborhood.

⊿ **Attributes (438)**

_histogram{0}.count (encoded: utf-8)	255
_histogram{0}.value (encoded: utf-8)	KWANZAN FLOWERING CHERRY
_histogram{1}.count (encoded: utf-8)	228
_histogram{1}.value (encoded: utf-8)	PISSARD PLUM
_histogram{2}.count (encoded: utf-8)	122
_histogram{2}.value (encoded: utf-8)	RED MAPLE
_histogram{3}.count (encoded: utf-8)	109
_histogram{3}.value (encoded: utf-8)	AUTUMN BLAZE RED MAPLE
_histogram{4}.count (encoded: utf-8)	107
_histogram{4}.value (encoded: utf-8)	ENGLISH HAWTHORN
_histogram{5}.count (encoded: utf-8)	104
_histogram{5}.value (encoded: utf-8)	CRIMEAN LINDEN
_histogram{6}.count (encoded: utf-8)	91
_histogram{6}.value (encoded: utf-8)	EUROPEAN WHITE BIRCH

NB: *To see this you must query a feature and inspect the Information Window, not the Table View.*

A looping transformer is useful here because it can scan through the list, extract all of the data we need, and process it simultaneously. It's also useful because it can handle a variable number of list items; for example, if there are only 5 species of tree in a neighborhood, we only process 5 species (not 10).

2) Create Custom Transformer

Click anywhere on the canvas and press Ctrl+T to start creating a custom transformer. This time, because we're using loop functionality only available in a custom transformer, we'll create it from scratch and not define it first in the main canvas.

Call the transformer a TreeCounter or TreeIndexer and set up automatic attribute reference handling (i.e. Handle with Published Parameters).

3) Connect Custom Transformer

At the moment the transformer will be empty apart from an Input and Output port. However, if you open the Input port's parameters you'll notice there are no attributes available.

That's because the custom transformer is not connected to anything in the main canvas.

So the first task is to temporarily switch to the main tab and connect the custom transformer, like so:
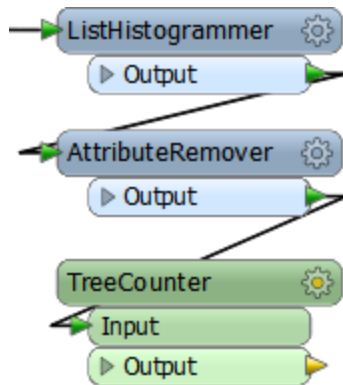
4) Define Inputs, Outputs, and Loop

In the custom transformer definition itself, the first task we'll do is create the input and output ports, and define the loop we're going to use too.

So, go to the custom transformer definition. Add a new Input object. Open its properties dialog, rename it to Loop and turn off the Publish setting (it doesn't need to appear as a data port).

Now place a Loop object (right-click on the canvas and choose Insert Transformer Loop). When prompted, set it to loop back to the Loop input port.

5) Expose Attributes

We'll need to use some attributes in here, so open the Input port parameters dialog and expose the list attributes _histogram{}.count and _histogram{}.value.
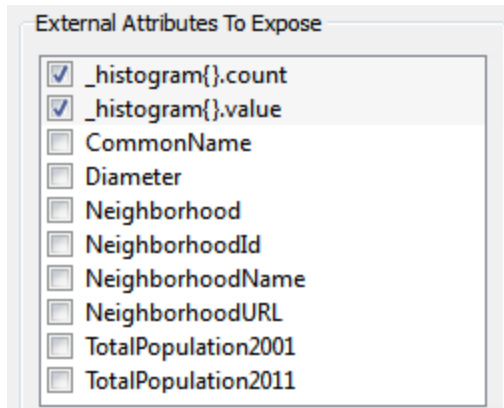
6) Define Loop Parameters

What we need to do is loop through each entry in the list, selecting a tree type, until we have either read the first ten elements or we've reached the end of the list.

To set up a condition for ending the loop we'll need to first count how many elements are in the list and secondly keep a count of the number of loops we've done, so we don't try to fetch information past the end of the list.

So, firstly add a ListElementCounter transformer to count the number of elements in the list. It will connect to the Input port (not the Loop port) because we only need to do this count once.

Set the attribute to scan the _histogram{} list and the Element Count Attribute to SpeciesCount.
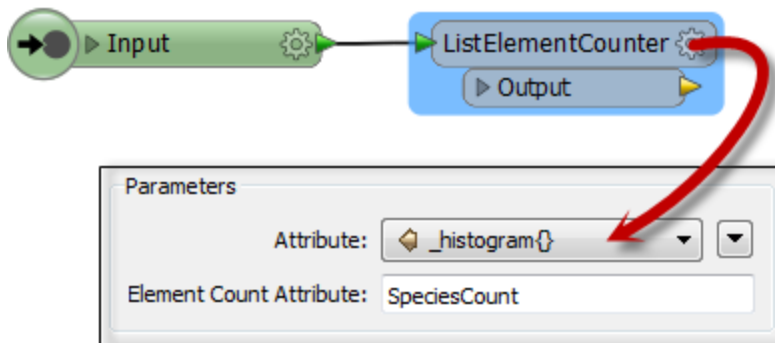
7) Define Loop Parameters - 2

Add an AttributeCreator transformer after the ListElementCounter. Use it to create an attribute called LoopCounter, with an initial value of zero (0).
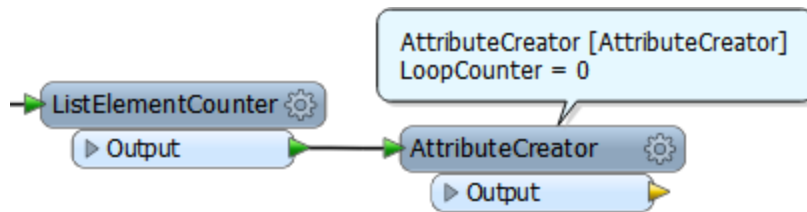
8) Define Loop Parameters - 3

Now add an ExpressionEvaluator transformer and connect it to the AttributeCreator Output port. Set it up to add one (1) to the value of LoopCounter, using the parameters provided to overwrite the existing value:

This is what will increment our loop counter each time we loop through the transformers:
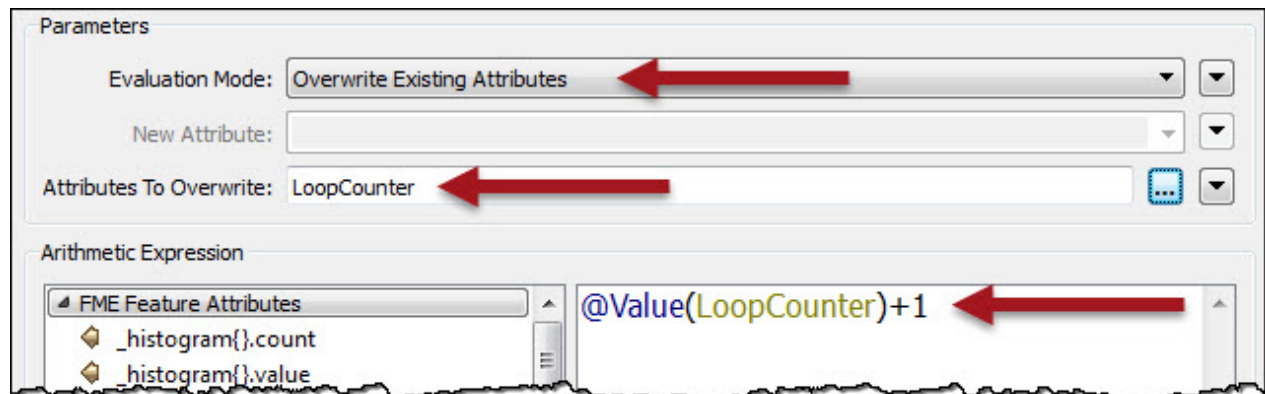
9) Set Condition

Add a Tester transformer connected to the ExpressionEvaluator.

Set it up to test for:

- LoopCounter = SpeciesCount **OR**
- LoopCounter = 10

	Left Value	Operator	Right Value	Negate	Mode
1	◇ LoopCounter	=	◇ SpeciesCount	☐	Automatic
2	◇ LoopCounter	=	k 10	☐	Automatic

Connect the Tester:Passed port to the Output port object.

Connect the Tester:Failed port to the Loop object.
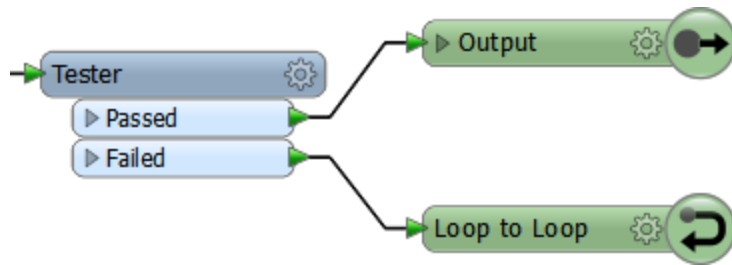
NB: If you don't see an attribute called SpeciesCount, check the ListElementCounter to make sure that is what you called the result of that action.

10) Connect Loop Input

Now connect the Loop input port to the ExpressionEvaluator. You should now have something that looks like this:
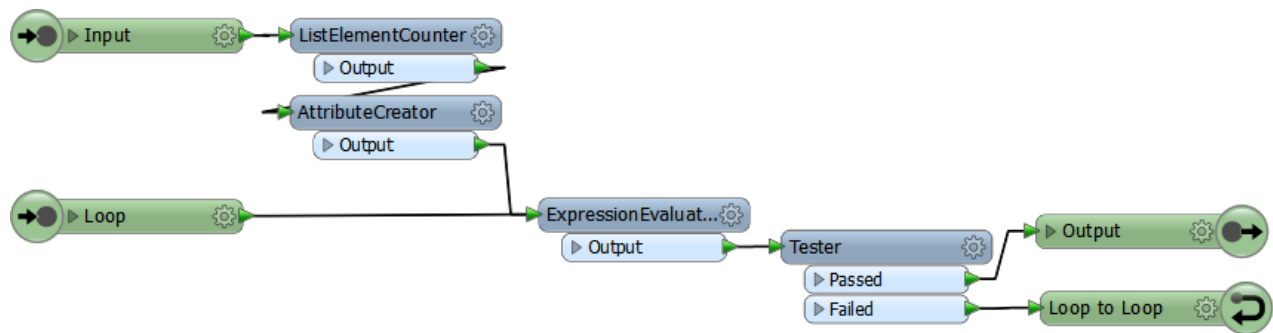
If you run this (and you certainly can do) it will run through the loop, up to a maximum of ten times per feature, and then finish the workspace.

What it won't do is process any data; in other words we've set up the looping structure but it's yet to do anything inside that loop.

11) Add ListIndexer

Add a ListIndexer transformer. This will fetch the next set of information in the list into the workspace. The ListIndexer needs to be connected before the ExpressionEvaluator, with connections from both the AttributeCreator and Loop input object.

The easiest way to connect this is to drag-connect it to the ExpressionEvaluator input port, like so:
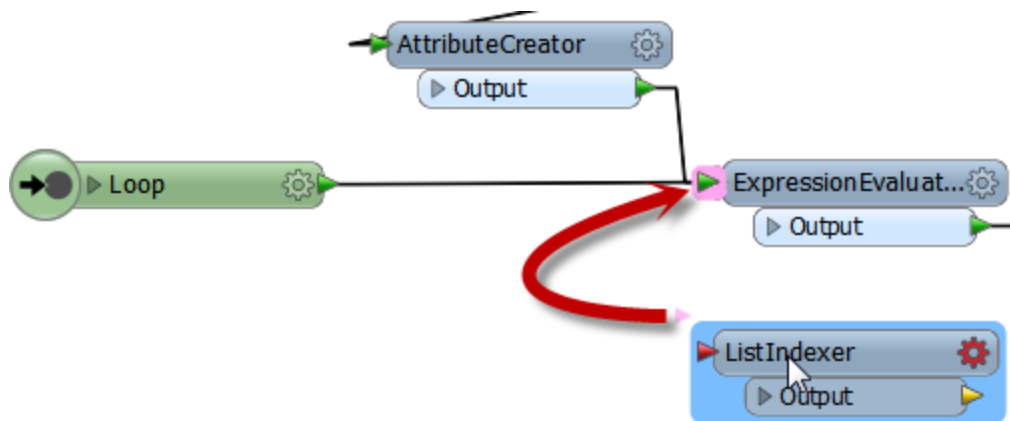
Open the ListIndexer parameters dialog. Set List Attribute to _histogram{} and the List Index should be set to the value of the LoopCounter parameter:
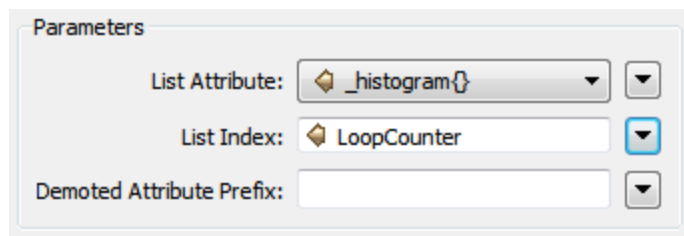
Now, for each iteration of the loop, the next set of list attributes (value and count) will be retrieved.

12) Add AttributeRenamer

Finally, add an AttributeRenamer between the ListIndexer and the ExpressionEvaluator.

Open up the parameters dialog. Set the Old Attribute parameter to the value attribute (i.e. we are renaming the value attribute). Set the New Attribute parameter to:

Species@Evaluate(@Value(LoopCounter)+1)

What this will do is rename the old attribute (value) to an attribute called SpeciesX, where X is the number of the loop, plus one. We add one to the value so the attributes don't start at zero!

This will give us something like:

- Species1: Oak

- Species2: Cedar

- Species3: Chestnut

- Etc.

In the Range Lookup Table,

13) Run Workspace

The custom transformer will now look something like this:
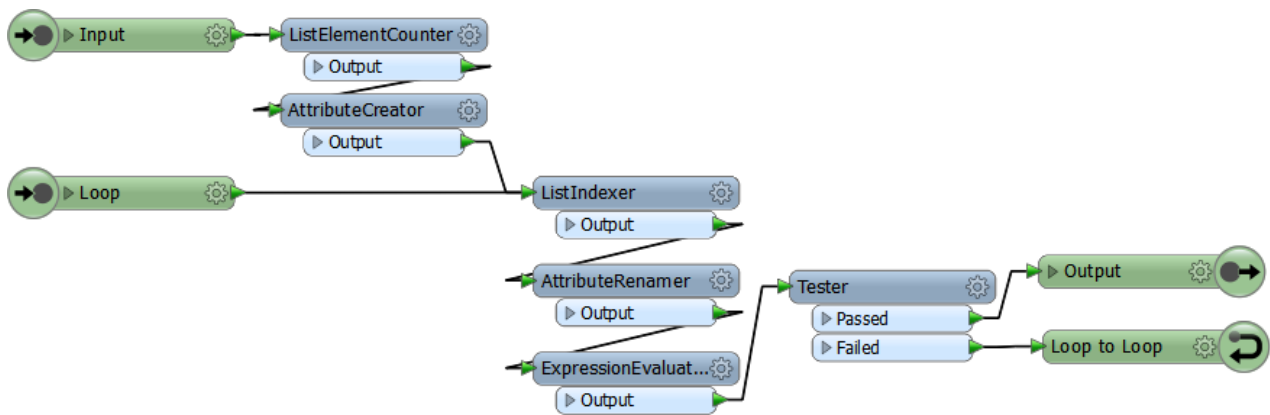
You may want to edit the Output port parameters so the only attribute it outputs back to the workspace is Species@Evaluate(@Value(LoopCounter)+1).
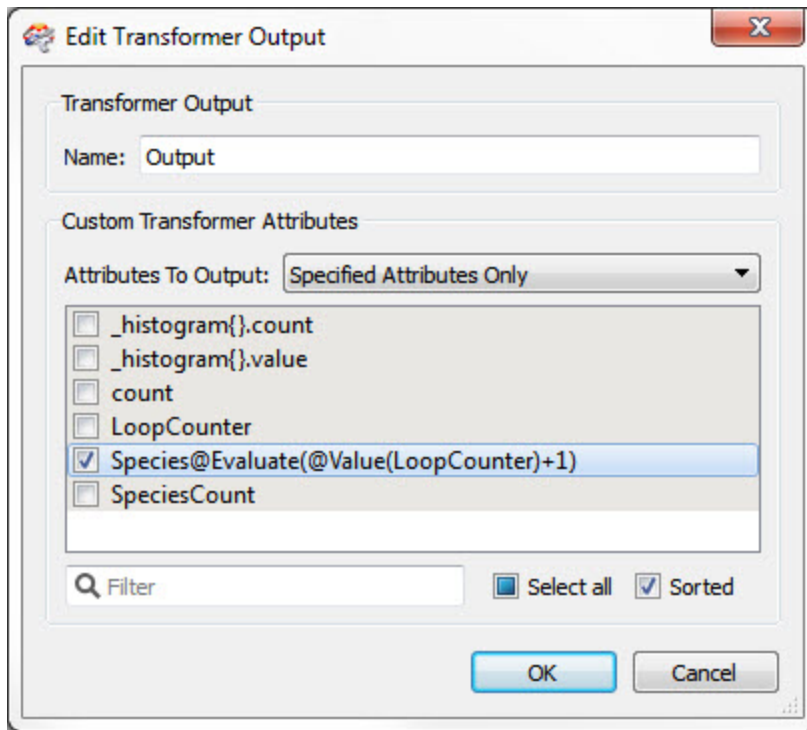
Return to the main tab and connect an Inspector transformer to the TreeCounter/TreeIndexer custom transformer, if there is not one already.

Run the workspace. Inspect the output. Querying a feature should produce something like this (you may need to click a "show all" link to expand the list fully):

NeighborhoodName (encoded: utf-8)	West End
NeighborhoodURL (encoded: utf-8)	http://vancouver.ca/green-vancouver/west-end.aspx
Species1 (encoded: utf-8)	KWANZAN FLOWERING CHERRY
Species2 (encoded: utf-8)	PISSARD PLUM
Species3 (encoded: utf-8)	RED MAPLE
Species4 (encoded: utf-8)	AUTUMN BLAZE RED MAPLE
Species5 (encoded: utf-8)	ENGLISH HAWTHORN
Species6 (encoded: utf-8)	CRIMEAN LINDEN
Species7 (encoded: utf-8)	EUROPEAN WHITE BIRCH
Species8 (encoded: utf-8)	NORWAY MAPLE
Species9 (encoded: utf-8)	BOWHALL RED MAPLE
Species10 (encoded: utf-8)	COMMON HORSECHESTNUT

First Officer Transformer says…

"Up to a point you could have done this by exploding the list with a ListExploder transformer and keeping the first ten features.

However, that wouldn't have helped with renaming the attributes. You would have had to rename them manually.

Plus, now we can change the number of items required by changing the "10" in the Tester to something else. For example, if I only want the first 8 trees then I would set it to 8. You could also set it up as a user parameter for the user to decide. Either way, this "setting" is easier to change when the transformer uses a loop like this."

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

Chapter 4 - Advanced Reading and Writing

Zip File Handling

Zip File Handling

Zip File Writing

Zip files are a convenient way to write output datasets that need to be handled as a single unit.

FME Desktop Advanced Training 2015

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:
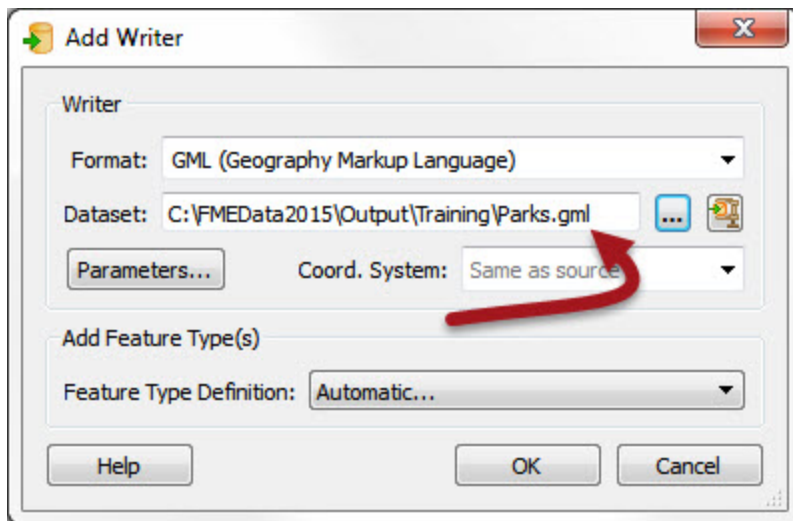
A small icon in the dataset field indicates the zipped status:
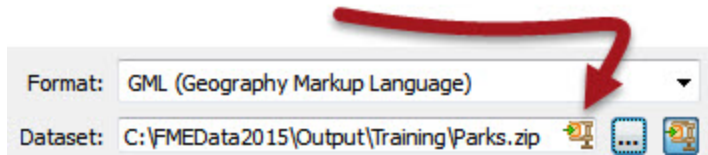
When the workspace is run the log file reports the zip creation:

```
|INFORM|MULTI_WRITER: Output will be zipped
```

And the output is, indeed, a zipped dataset:
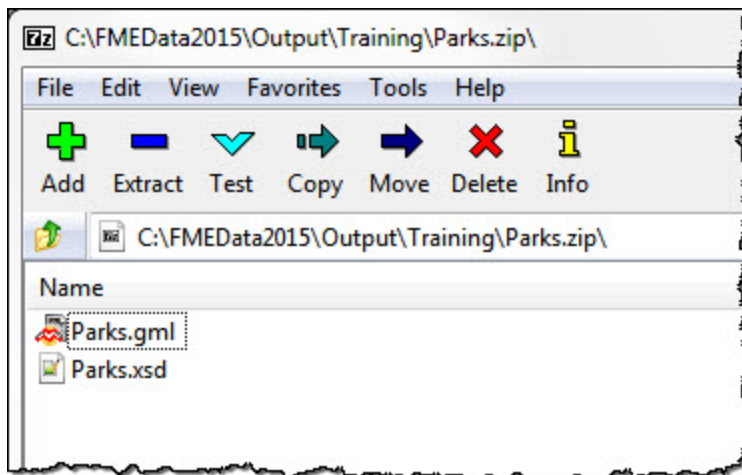
Sister Intuitive says…

"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 process it as a single entity. Here, a TCL or Python shutdown script will know the name of the file just written through published parameters (macros). With that information the file can be processed – for example, uploaded to an FTP site – with minimal difficulty."

Another way to force a zip file output is to simply click the new "Zip Output" button in the writer dialog:
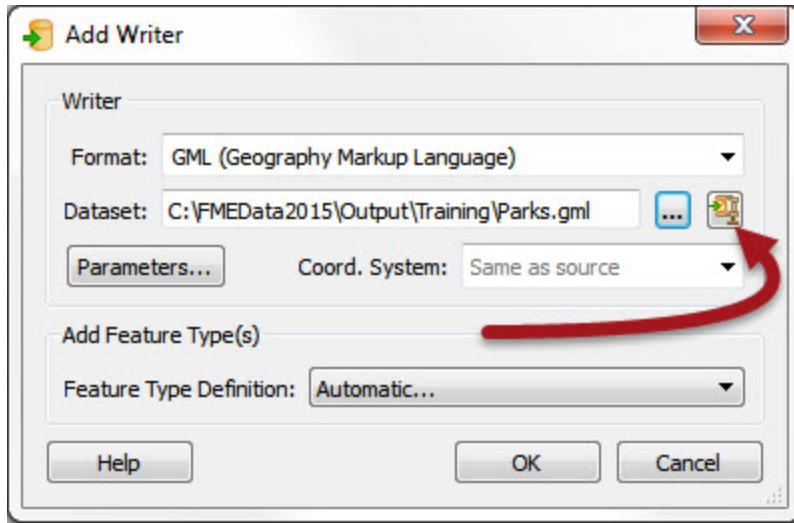
Doing so automatically sets up the output to be written with the given name, but inside of a similarly named zip file:
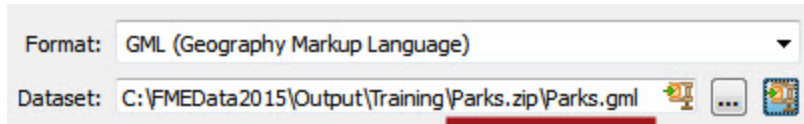
Sister Intuitive says…

"Although the above images show how to achieve a zipped output in the Generate Workspace dialog, the same functionality is available regardless of where and how you set the output file (for example, through the Navigator window). You just need to give it a .zip extension."

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, as the final step of a translation.

Because a fanout occurs as the data is being written, it does not cause multiple flows of data inside the workspace. Therefore this technique makes it easy to create groups with minimal impact on the workspace canvas.

Similarly to a group-by, the groups are defined by an attribute.

For example, here an author is "fanning-out" a set of data into multiple outputs depending on a feature's elevation attribute:
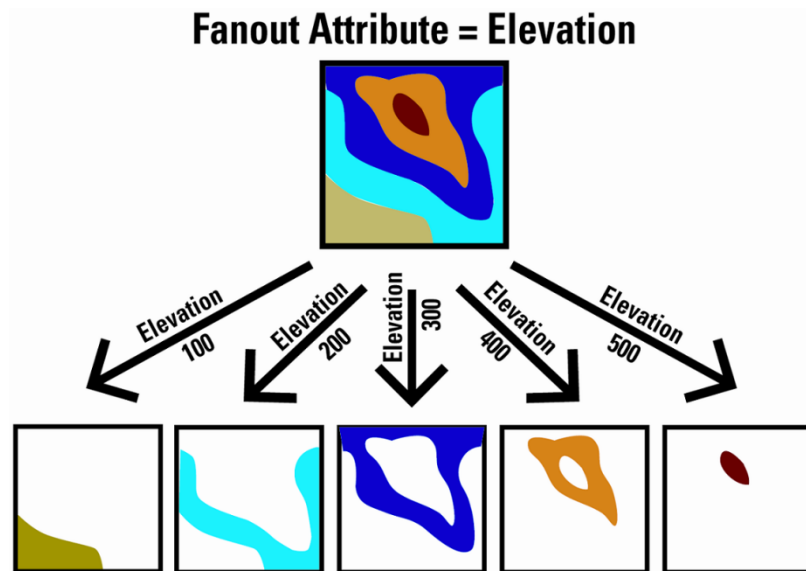
A major benefit of a fanout is the high degree of flexibility – and freedom from fixed-layer schemas – in return for minimal effort.

There are two types of fanout: Feature Type Fanout and Dataset Fanout.

> *Sister Intuitive says…*
>
> *"If this technique sounds familiar, that may be because it carries out a similar function – albeit in reverse – to the "Merge Feature Type" parameter."*

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:
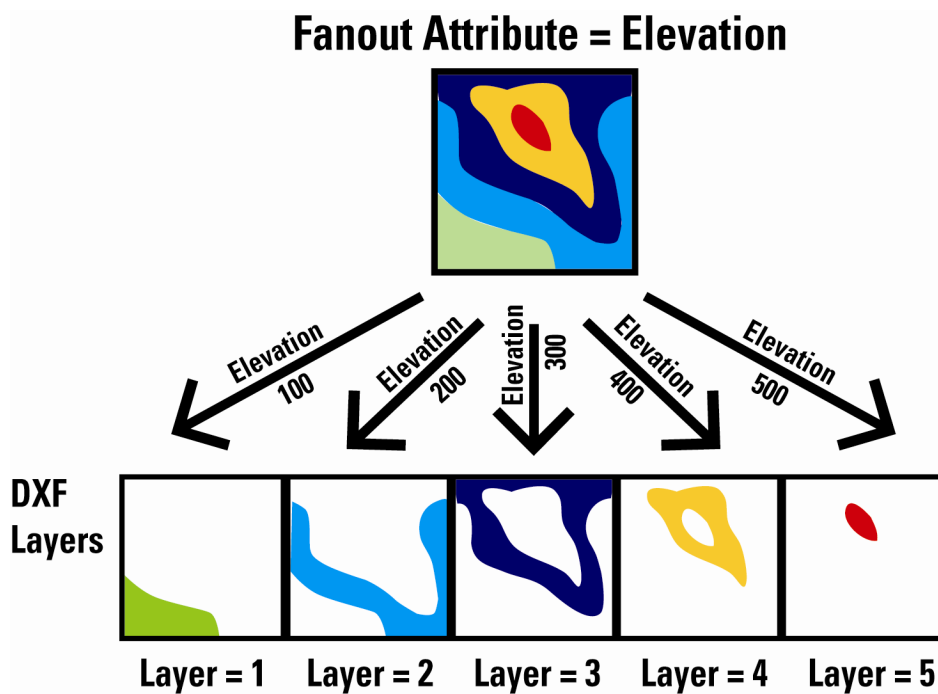
This results in a DXF dataset containing multiple layers of data.

A feature type parameter is defined using an option in the Feature Type Properties dialog:
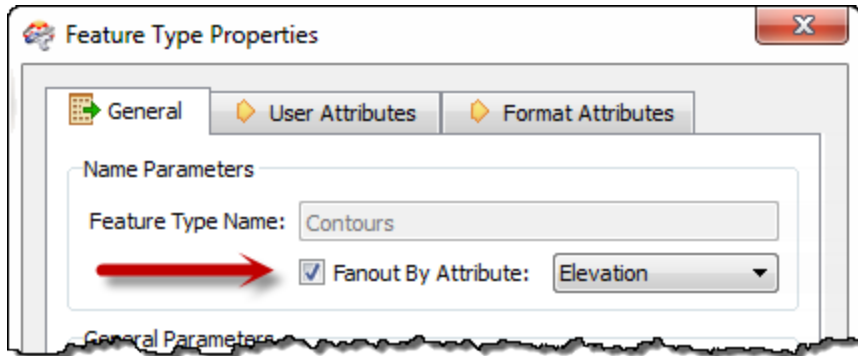
Dataset Fanout

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.
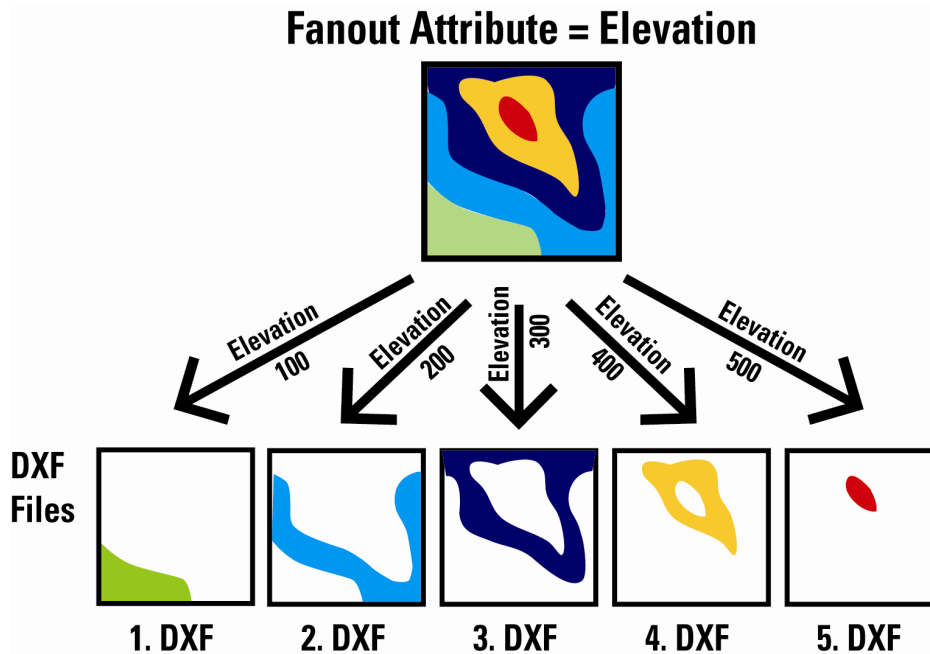
A Dataset Fanout is defined in the Navigator window in Workbench, under a Writer's Advanced Parameters.

Parameters
  Output Format: ACAD (Linked to 'GENERIC_OUT_FORMAT_GENERIC')
  Base filename: <not set>
  Advanced
      Fanout Dataset: Yes
      Fanout Directory: C:\FMEData2014\Output
      Fanout Prefix: <not set>
      Attribute to Fanout on: Elevation
      Fanout Suffix: <not set>

Like a Feature Type Fanout, there is a parameter to select the attribute to fanout on. However, there are also parameters to let the author define a prefix and suffix for the fanned-out file names.

Exercise 4a Fanouts and Zip Files	
Scenario	FME author; City of Interopolis
Data	Development Zones (MapInfo Tab)
Overall Goal	Convert zones to Shape, with a separate file per zone type
Demonstrates	Fanouts and zip file writing
Starting Workspace	None
Finished Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise4a-Complete.fmw*

You've been given a dataset of development zones and asked to separate each zone type into a separate Shape file and send it back with everything zipped together in a single file.

The requestor 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

The source data for this translation is a MapInfo TAB dataset:
C:\FMEData2015\Data\Zoning\Zones.tab

Inspect the source dataset in the Data Inspector. 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 Shape.

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.
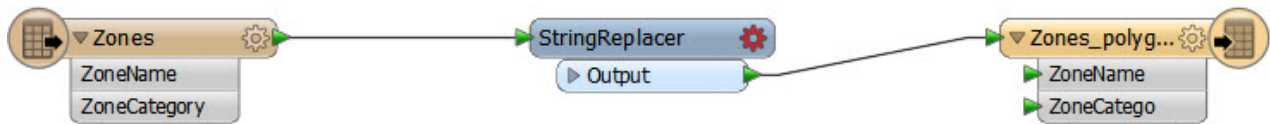
4) Set Parameters

Open the parameters dialog for the StringReplacer.
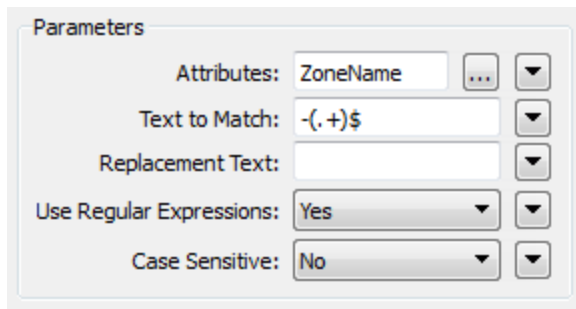
For the Attributes field, select ZoneName.

For the Text to Match enter: -(.+)$

Leave the Replacement Text field empty.

Select 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. Set the Fanout by Attribute checkbox on and select ZoneName as the attribute to fan out by:
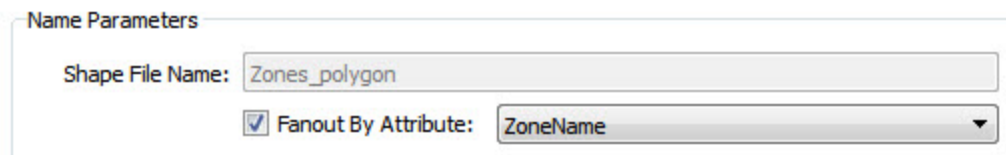
Click OK to close the dialog.

6) Save and Run Workspace

Save the workspace. Run the workspace using **File > Prompt** and Run.

When prompted manually change the Destination Folder to:
C:\FMEData2015\Output\Zones.zip.

Open the output folder in a file browser.

You should see the file Zones.zip. If you open it up there will be inside a Shape file for every zone type.
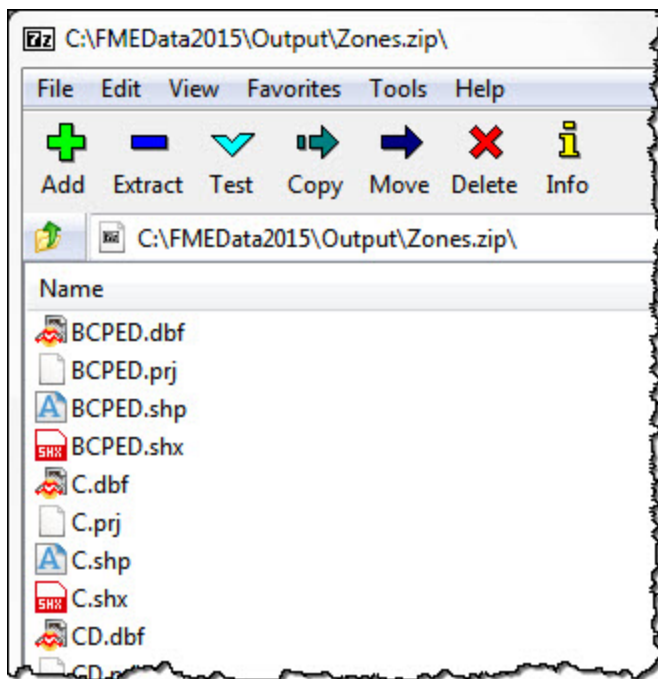
Sister Intuitive says…

"A feature type fanout results in multiple Shape files because each Shape file is a layer (feature type).

Why not repeat the exercise to get DWG files, in which case you'd need to use a Dataset Fanout."

The Generic Reader/Writer

The Generic Reader/Writer

Generic Formats
The Generic Reader
The Generic Writer

FME Desktop Advanced Training 2015

The Generic Reader and Writer allow FME workspaces to be freed from format restrictions.

Generic Formats

FME's generic tools are a Reader and Writer that are not tied to a particular format. 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

Whereas all other Readers are tied to a particular format, the Generic Reader is capable of handling almost any format.

A Generic Reader is simply added the same way that any other Reader is used; by specifying the format in the new Reader dialog:
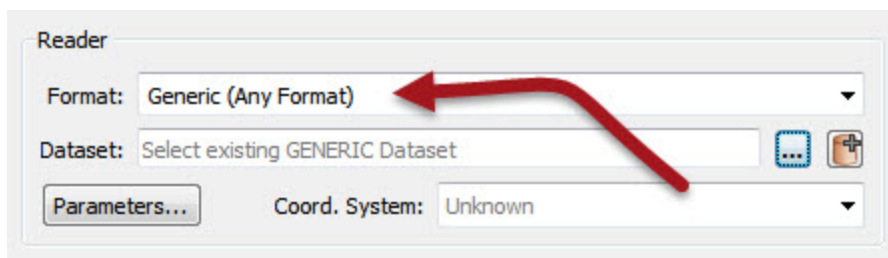
A parameter allows the format of the data being read to be specified, or it can be left up to FME to ascertain the format from the data extension of the source file.

Parameters
    Input Format: GUESS_FROM_EXTENSION
▷ Advanced

Now at run time, the source dataset can be set to GML:

Translation Parameters

User Parameters

Source Dataset: C:\FMEData2015\Output\Training\Parks.gml

OK   Cancel

Or MapInfo TAB:

User Parameters

Source Dataset: C:\FMEData2015\Data\Parks\Parks.tab

Or any other format of data you care to read (provided the data has the same schema). This makes for a very flexible workspace

Generic Reader Feature Types

However, the Generic Reader is not immune from the Unexpected Input Remover. Remember, this is the functionality 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 removed.

For this reason, you'll want to ensure that all potential layers are defined as feature types in the workspace; or that you have a Merge Feature Type set in the Feature Type Properties dialog:

With that setup, any layer of data can be passed into the workspace:

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 none of these.

So, for example, a user wishes to apply a particular parameter to a GML dataset read with the Generic Reader, how is it done?

In brief, the solution is to add a dummy GML Reader. In fact, a Resource Reader is the best solution, because it applies parameters without reading any data (more info on Resource Readers appears later in this chapter).

When the Generic Reader reads a dataset of GML format, it will now look to the parameters of the dummy Reader, and use those to set how it reads GML datasets.

For example, this workspace author uses a Generic Reader to read his GML data. It is a dataset of parks in the city but, sadly, the x/y axes are being read incorrectly:

There is a parameter to control the x/y axes in the GML Reader, but it doesn't appear in the Generic Reader. So, the author selects Reader > Add Reader as Resource from the menubar:
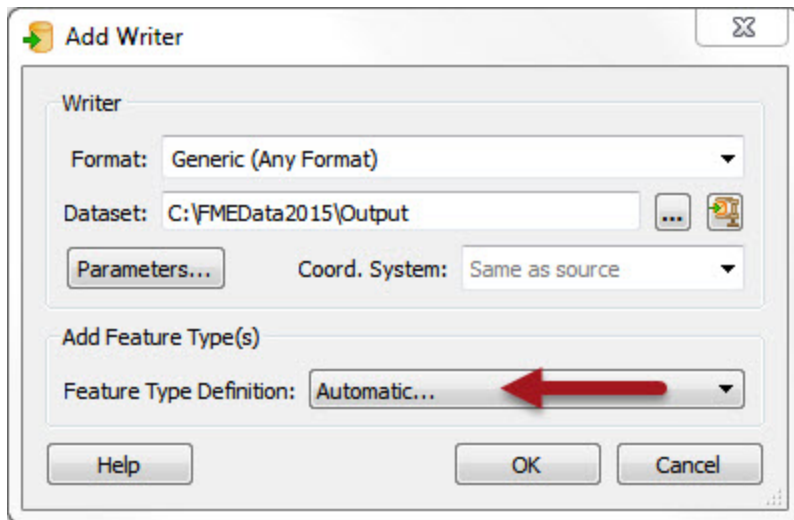
When prompted the user defines the format as GML and picks a GML dataset (it won't matter which).

In the Navigator window they locate the newly added Reader under Workspace Resources…
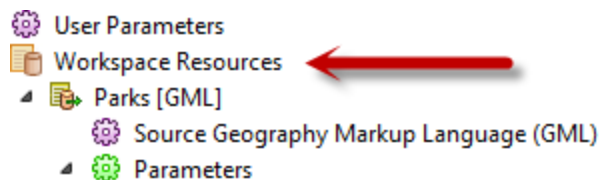
…and locates/set the GML Axis Order parameter:

⚙ Ignore xsi:schemaLocation in Dataset: No
⚙ GML SRS Axis Order: 2,1 ⬅
⚙ HTTP Username: <not set>

Now when the workspace is run, the Generic Reader uses this GML resource to determine the parameters to read GML data with. The parameter is applied and the data read correctly:

Sister Intuitive says…

"If the Input Format parameter is turned into a user parameter, then the user can specify at runtime what format of data is being read. This is especially helpful when the extension is – like .mdb – one used by multiple formats.

Additionally, the workspace can be made to test for the format being read and then filter the data to process different formats in different ways:"

Tester [Tester]
Test Description: ""
TEST $(GENERIC_IN_REAL_FORMAT_GENERIC) = GML
Condition: OR

Tester ⚙
▷ Passed
▷ Failed

GML Data

Not GML Data
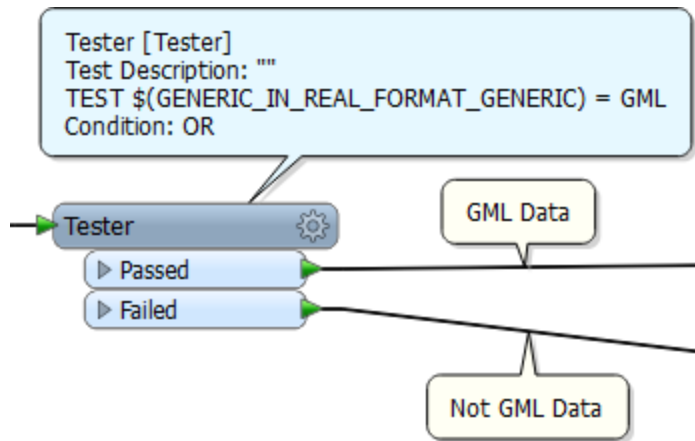
The Generic Writer

Similarly to the Generic Reader, the Generic Writer is a component of FME without a specific format.

A Generic Writer is simply added the same way that any other Writer is used; by specifying the format in the new Writer dialog:

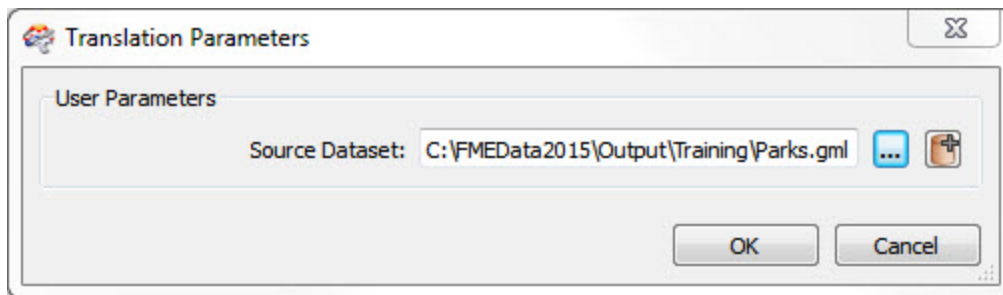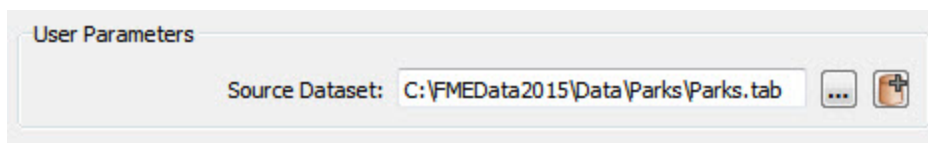When a workspace with a Generic Writer is run, the format of data written is determined by a parameter that can be set in the Navigator window:

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.

For example, here the user is reading a parks dataset (coincidentally also using the Generic Reader) and writing the data out to KML format using the Generic Writer:
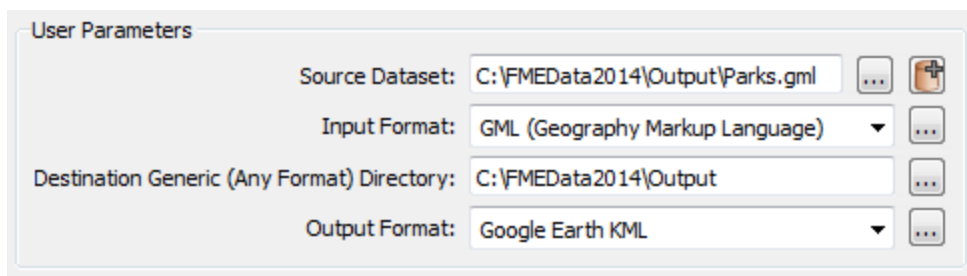
Semantic Translations and the Generic Writer

It's important to remember that FME is a semantic translator, carrying out transformations on output data to fit the definitions and rules of the destination format.

In other words, FME will automatically restructure data to fit the output format's rules on geometry and attributes (both names and values).

Therefore, the Generic Writer may produce slightly different results for different data formats.

Generic Writer Feature Types

The feature types defined for a Generic Writer are – like the Generic Reader – relatively inflexible. The layers that are defined will be the layers that are created, regardless of format.

Of course, there is a way to be more flexible – and creative – with the feature types that are written, and that's with functionality that we just covered: Fanouts!

If the Generic Writer uses a feature type fanout, 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, a particular output format in the Generic Writer can be controlled using parameters from a dummy Writer of the same format, which has been added to the workspace.

The difference is that this will be a "real" Writer and not a Resource Writer (of which there is no such thing). 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.

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.

To work with flexible attribute schemas requires the use of either Automatic Attribute Definitions or a Dynamic Translation."

Exercise 4b Generic Formats	
Scenario	FME author; City of Interopolis
Data	Community Mapping (Geodatabase)
Overall Goal	Create self-serve workspace
Demonstrates	Generic Writer
Starting Workspace	None
Finished Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise4b-Complete.fmw*

In this exercise your task is to create a workspace with which end-users can output a dataset of Community Mapping in a format of their choice. This would be an excellent workspace to use for an FME Server Data Download service.

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:\FMEData2015\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.

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:
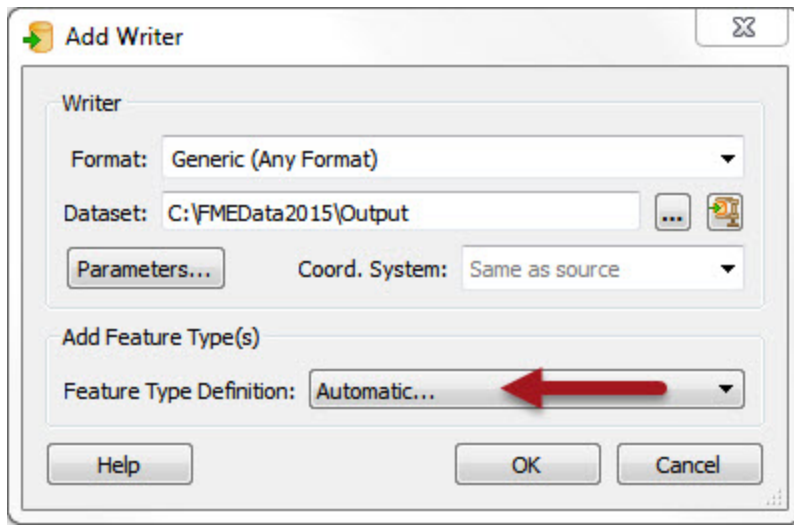
In the Feature Type Properties dialog set the Allowed Geometries field to fme_any. This will allow any data to be written to this feature type. The "output" part of the Writer refers to the output location, which may or may not be set in your workspace.
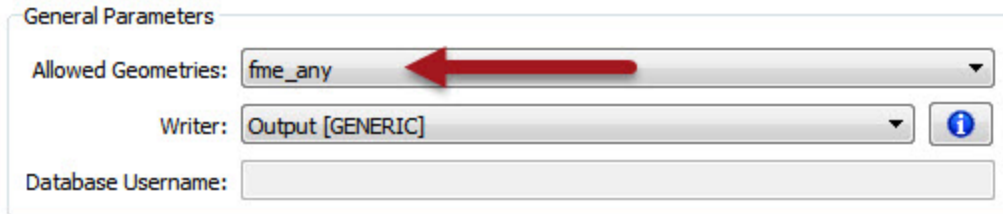
Check the User Attributes tab and you will see that the attribute definition is set to Automatic.
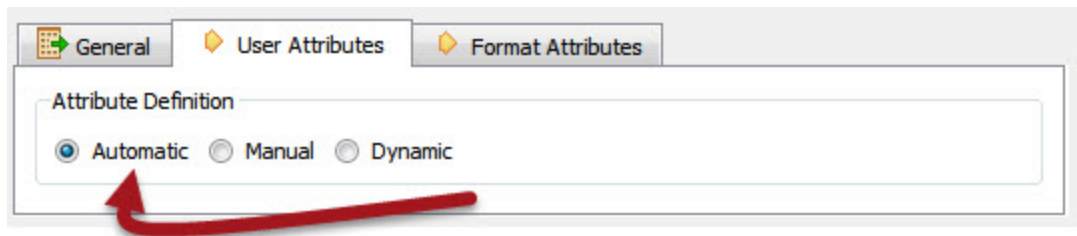
Click OK to close the dialog and add the new feature type. Connect it to the source feature type. When you make the connection the attribute schema will automatically be updated to match the connected Reader feature type:
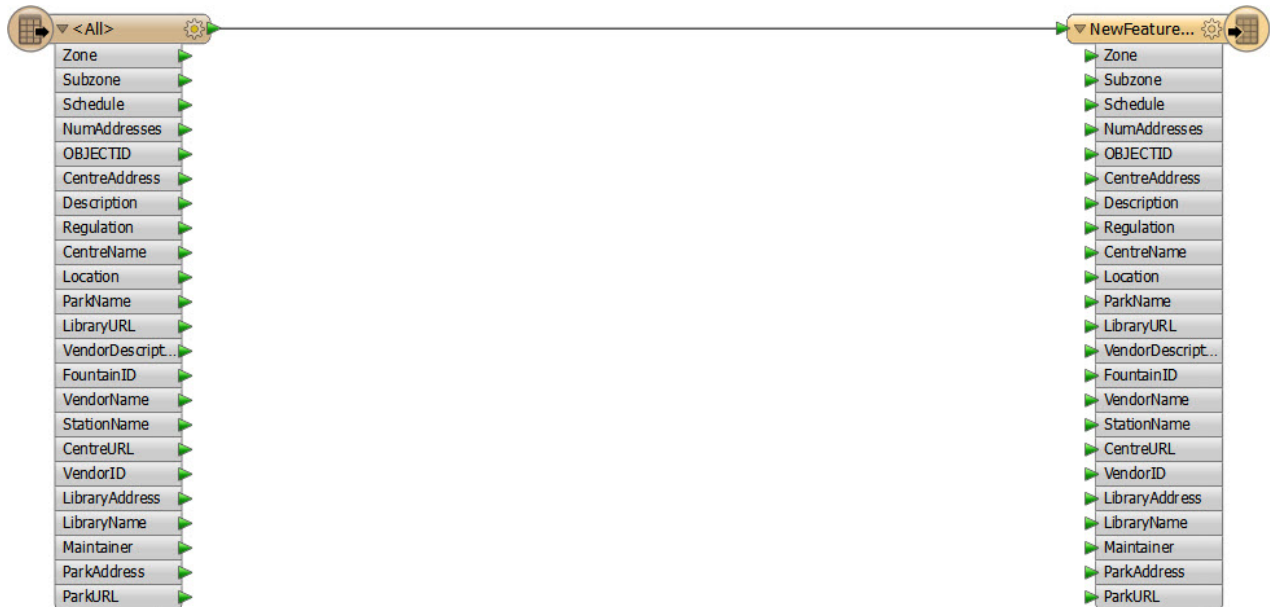
3) Check User Parameters

Look in the Navigator window at the user parameters that were created automatically with the Reader and Writer.

The parameter for source dataset we don't need (it will always be this dataset) so delete it.

Another parameter is Feature Types to Read. This is very useful because when the user runs the workspace they will be prompted which tables to read from the source Geodatabase, so keep this one.

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.
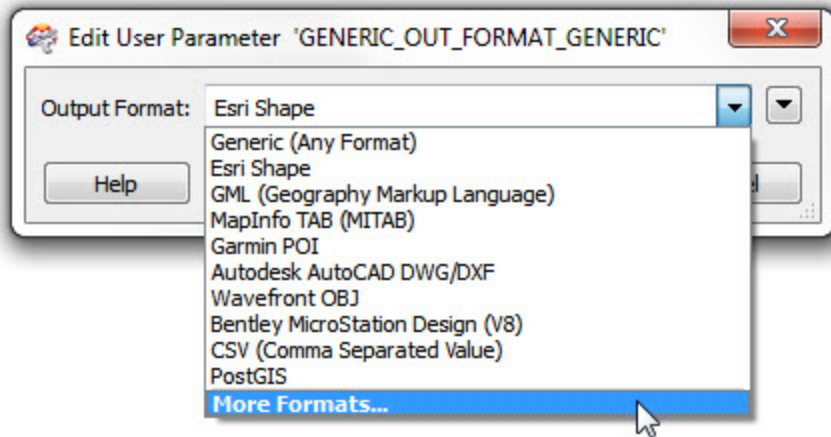
It wouldn't be fair to the end-user to expose so many formats, when they don't really need to see or select most of them. It would be better to restrict this list. So, delete this user parameter, and we'll create a new one.

4) Add User Parameter

Now add a new User Parameter, by right-clicking on User Parameters and selecting Add Parameter. In the dialog that opens…
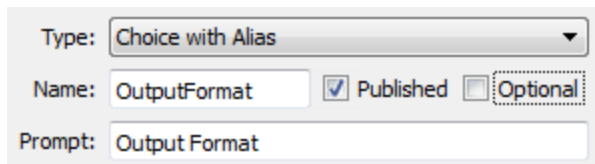
Set type to Choice with Alias.

Set OutputFormat as the name.

Set Output Format as the prompt.

Turn off the Optional checkbox.

For configuration 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.
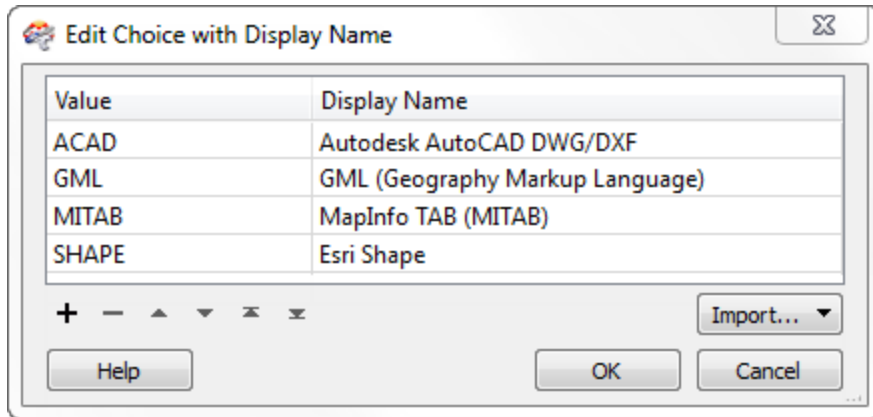
5) Link Parameter

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:
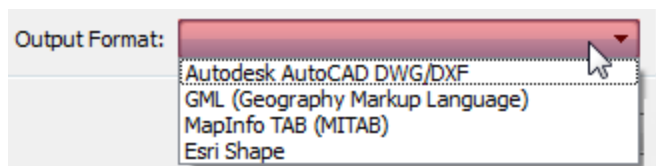
6) Expose Format Attribute

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:

7) Set Feature Type Fanout

Now open the properties dialog for the Writer feature type. Check the box for Fanout by Attribute and select fme_feature_type as the attribute to fanout by.
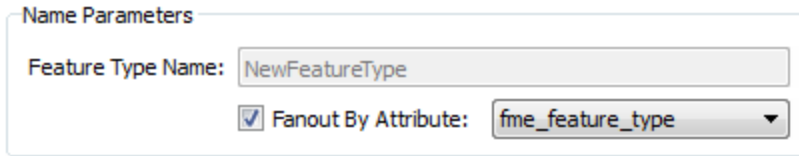
8) Save and Run Workspace

Save the workspace and then run it using Prompt and Run. When prompted, select some source tables to read (include at least the GarbageSchedule plus one other) and set an output folder. Set Esri Shape as the format to write.
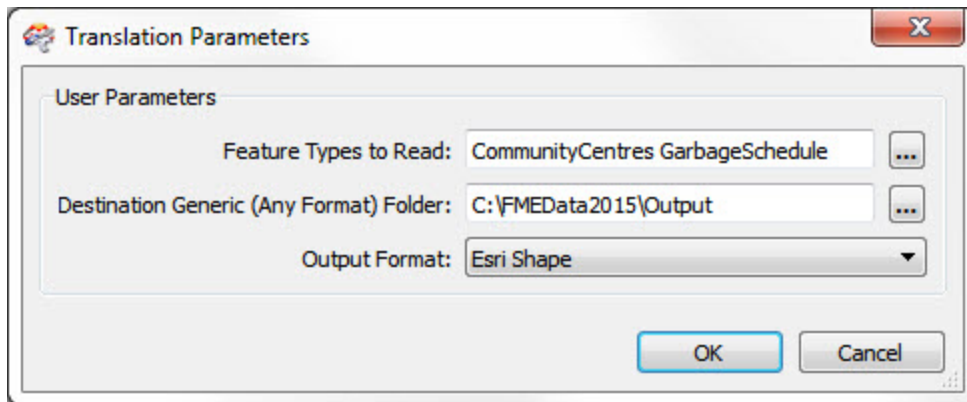
Examine the output folder. The selected tables have been written back to Shape format.

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 the source data. To avoid that you would need to add each table in the source separately (as we can see in the Advanced Task below).

Advanced Task

If the Writer feature type was set to a manual definition of the attribute schema, then we would be unable to use any other source data as the Generic Writer would be constrained by the fixed nature of this schema. However, since we made the attribute definitions automatic, it's now possible to run this workspace and use a different dataset... provided we add a new Reader too.

9) Add Reader

Select **Readers > Add Reader** from the menubar and add the following:

Reader Format: Esri Geodatabase (File Geodb API)
Reader Dataset: *C:\FMEData2015\Data\Addresses\Addresses.gdb*
Workflow Options: Individual Feature Types

There will be separate Reader feature types for addresses and postcode boundaries.

10) Connect Schema

In the workspace, disconnect the CommunityMap feature type from the Generic Writer, and instead connect the PostalAddress feature type. Notice that the schema will automatically update to match the Address data.
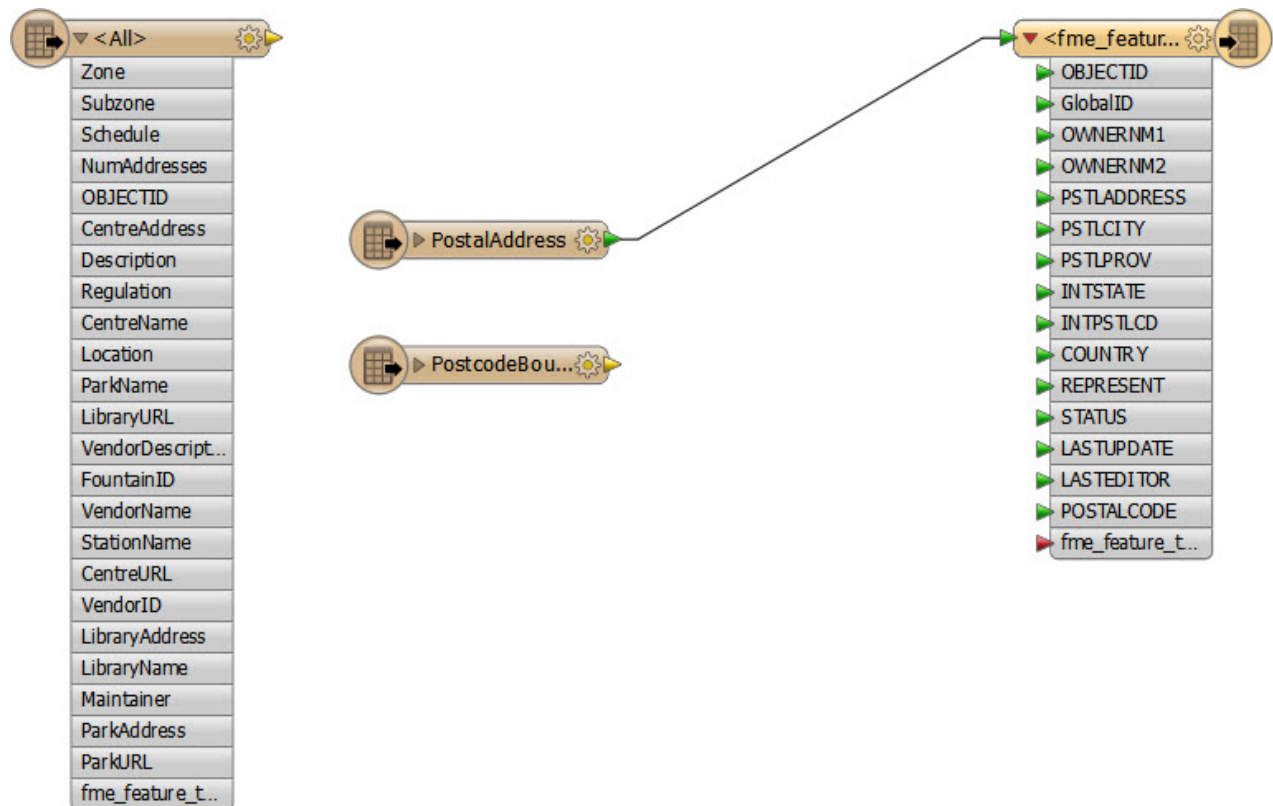
Run the workspace (setting a different output format such as GML)and inspect the output to prove that the Generic Writer is writing the format correctly, and that the automatic attributes setting is creating the correct attribute schema:

Feature Information		
	Features Selected: 1 of 1	
Property	**Value**	
Feature Type	PostalAddress	
Coordinate System	EPSG:26910	
Dimension	2D	
Number of Vertices	1	
Min Extents	490997.39840000001, 5458984.0091000004	
Max Extents	490997.39840000001, 5458984.0091000004	
▲ **Attributes (20)**		
COUNTRY (encoded: utf-16)	Canada	
fme_geometry (string)	fme_point	
fme_type (string)	fme_point	
GlobalID (encoded: utf-16)	{892FF2D2-B6E2-476A-B5F1-D753A9F0611F}	...
gml_id (encoded: utf-16)	id11e8c027-00e5-49b9-83c4-6034ec856599	...
gml_original_coordinate_system (encoded: utf-16)	EPSG:26910	
INTPSTLCD (encoded: utf-16)	CA-V6E 0C9	
INTSTATE (encoded: utf-16)	CA	
LASTEDITOR (encoded: utf-16)	GIS Technician	
LASTUPDATE (encoded: utf-16)	20141128000000	
OBJECTID (encoded: utf-16)	472	
OWNERNM1 (encoded: utf-16)	Nichelle Sancho	
OWNERNM2 (encoded: utf-16)		
POSTALCODE (encoded: utf-16)	V6E0C9	
PSTLADDRESS (encoded: utf-16)	1030 Robson St	
PSTLCITY (encoded: utf-16)	Vancouver	
PSTLPROV (encoded: utf-16)	British Columbia	
REPRESENT (encoded: utf-16)	Owner/Occupant	
STATUS (encoded: utf-16)	Current	
xml_type (string)	xml_point	
▲ ● **IFMEPoint**	490997.39840000001, 5458984.0091000004	
Name (encoded: utf-16)	pointProperty	

Dynamic Translations

Dynamic Translations are a way to create 'schema-less' workspaces.

What are Dynamic Translations?

In previous chapters all translations have involved 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 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."

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.
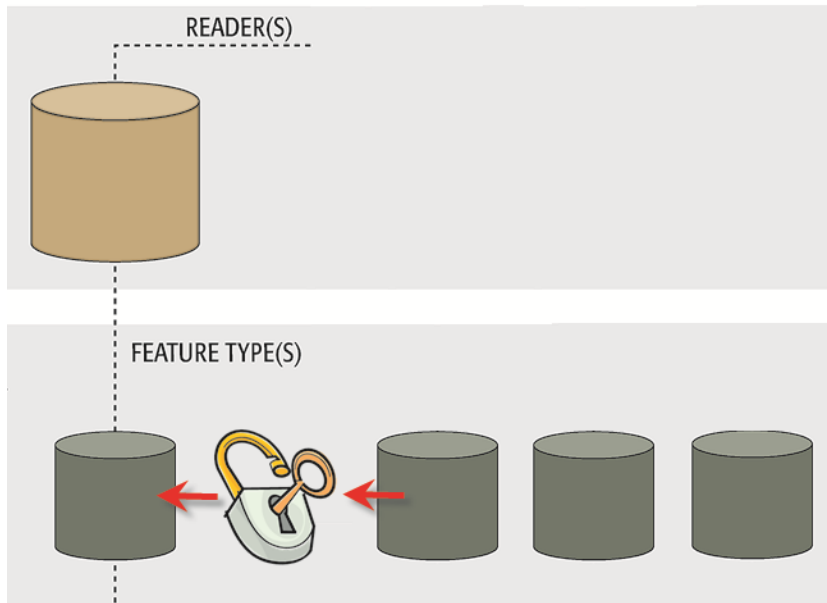
The Writer side of a dynamic workspace mimics the Reader part; feature types are written to the destination dataset, regardless of whether they have been defined in the workspace.

Additionally, all attributes and geometries are also written, regardless of whether they too have been predefined in a Writer feature type.
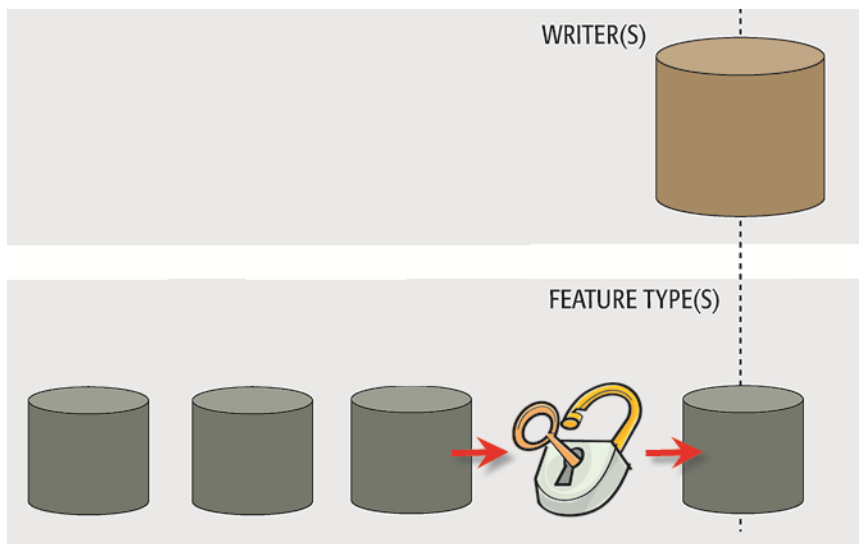
Creating a Dynamic Translation

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.
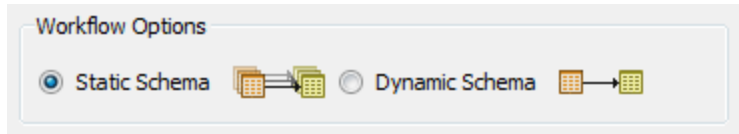
The Static Schema option is the default for a workspace including schema. Choosing the Dynamic Schema option creates a schema-less workspace with dynamic Readers and Writers.

The Add Writer dialog also has options for Static and Dynamic schema; so too does the Add Reader dialog, although there they are described a little differently.
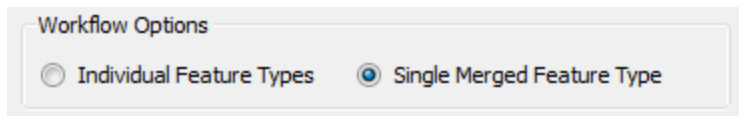
In this way it's possible to define individual readers and writers as 'dynamic'.

> *Sister Intuitive says…*
>
> *"You can use any Reader and Writer format in conjunction with a dynamic workflow; so don't get confused with the Generic Reader and Writer formats.*
>
> *The term "Generic" means "any format", while "Dynamic" means "any schema". A workspace may be generic, or dynamic – or both!"*

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.

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.

Exercise 4c Dynamic Translations	
Scenario	FME author; City of Interopolis
Data	Community Mapping (Geodatabase)
Overall Goal	Create self-serve workspace
Demonstrates	Dynamic Translations
Starting Workspace	None
Finished Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise4c-Complete.fmw*

In the previous exercise, a workspace was generated to translate a Geodatabase dataset into a number of formats using the Generic Writer.

However, now you feel it would be useful if that workspace could handle any source Geodatabase, not just the community maps dataset, without having to add new Readers or feature types every time.

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:

Reader Format	Esri Geodatabase (File Geodb API)
Reader Dataset	*C:\FMEData2015\Data\CommunityMapping\CommunityMap.gdb*
Writer Format	Generic
Writer Dataset	*C:\FMEData2015\Output\Training*
Parameters	
Output Format	Esri Shape
Workflow Options	Dynamic Schema

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 using Prompt and Run.

When prompted, select some source tables and set the output format.
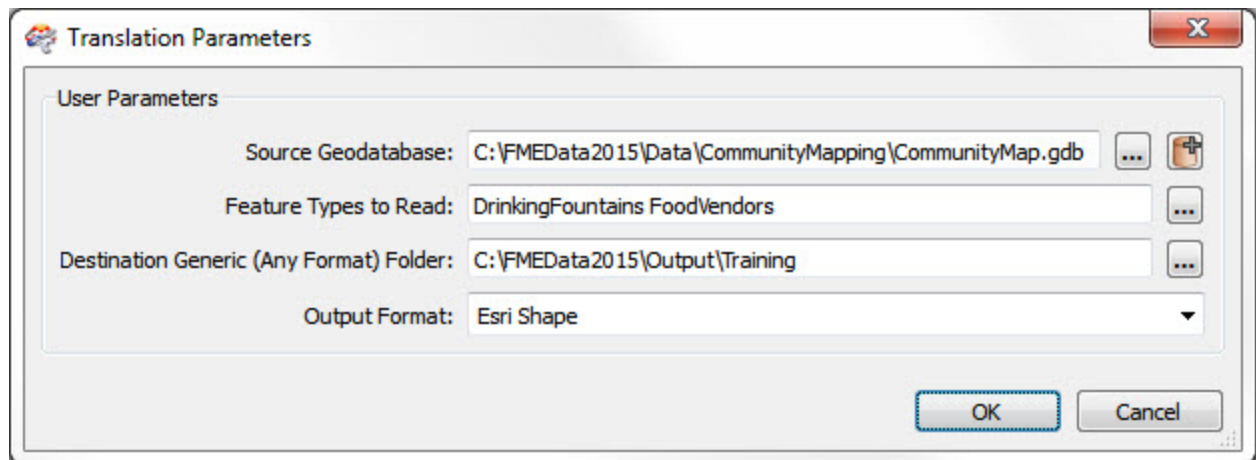
The workspace will run to completion. Check the output to ensure it is all correct.

4) Re-Run Workspace

Now run the workspace again.

This time click the browse button for the source Geodatabase and browse to
C:\FMEData2015\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.

Schema Handling in Dynamic Translations

Schema Handling in Dynamic Translations

Dynamic Translation Settings
Schema Sources
Schema Definition
Adding or Deleting Attributes

Dynamic Translations still use a schema, but it is not necessarily a carbon copy of the source dataset.

Dynamic Translation Settings

Opening the feature type properties dialog for a dynamic translation reveals the checkboxes that turns on this behavior.

For a Reader, all that is really happening is the merge feature type setting is turned on:
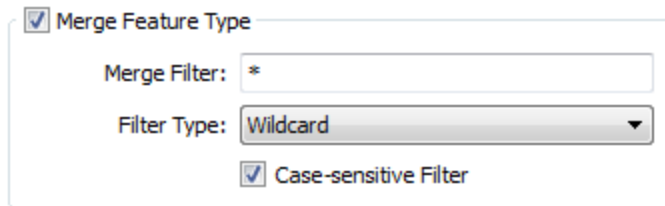
For a Writer, there is a special Dynamic option (it's not just a feature type fanout):
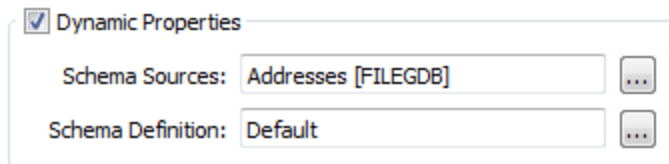
So, it is possible to create a dynamic translation by turning one or both of these settings on manually; there's no necessity to do so at the time the workspace is created.

The two extra parameters on the Writer are to control the source of the schema being used on the output dataset.

Sister Intuitive says…

"Yes, a dynamic translation is often used to handle a variety of source datasets, by automatically duplicating the incoming schema – feature types, attributes, geometry types – on the output.

However, it's also possible to dynamically select the outgoing schema from a completely different location, or from an attribute in the data, or even by constructing a virtual schema in the workspace!"

Schema Sources

The schema sources parameter allows the user to choose 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. In the case where a Reader is only required to provide a schema, and not data, then it would be added as a Resource Reader.

For example, here the author has opened that parameter and is changing the output schema from the default (input dataset) to a standard corporate schema that is defined in PostGIS and has been added as a Resource Reader:
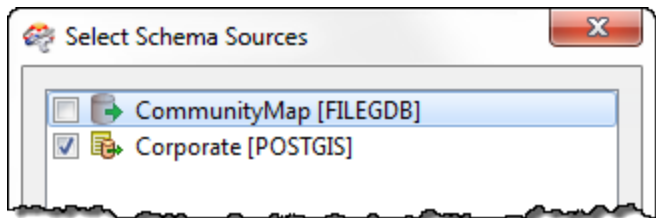
This is useful when you want to enforce a particular output schema, but it means that you have to ensure that the data being processed will match the feature types available in that schema.

If you write data to a dynamic Writer, and the feature types of that data do not match what is provided by the schema then it will be dropped:

```
|                          Features With No Schema defined
|=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
|DrinkingFountains                                                  113
|FoodVendors                                                         91
|GarbageSchedule                                                      6
|==================================================================
|Total Features NOT Written                                         210
|=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

Consider this behavior a sort of "Unexpected Output Remover".

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."

Schema Definition

The Schema Definition parameter opens a dialog with a number of options:

These options are where a user can become very creative with their dynamic workspaces.

Remember, a schema is composed of three basic parts: Feature Types, Attributes, and Geometry Types. These settings give the author control over each part of that schema.

By default, each part is obtained from the incoming schema, whether that's a real Reader or a Resource Reader. But these settings let the author pick alternative sources.

For example, the author could decide that the feature type name will be fixed (i.e. all output must match this set value), that the attribute definitions should come from feature types defined by an attribute value, and that geometry is set to a fixed type.

Here each incoming feature is assumed to possess an attribute called OutputLayer.

That attribute defines what feature type (layer) the data will be written to.

It also defines the Reader (or resource) feature type that the attribute schema will be taken from.

Finally, it specifies that the geometry of the output features must be area (polygon) features. Non-area features will be dropped.

Sister Intuitive says…

"Don't be confused by the Schema [Attribute] Definition setting. When you set it to 'Named By Attribute' it means that attribute defines the feature type names to use for the attribute schema.

It DOESN'T mean that the attribute schema is somehow defined inside that attribute."

Adding or Deleting Attributes

Even in a dynamic translation, sometimes you need to add or delete attributes from the output schema. 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.
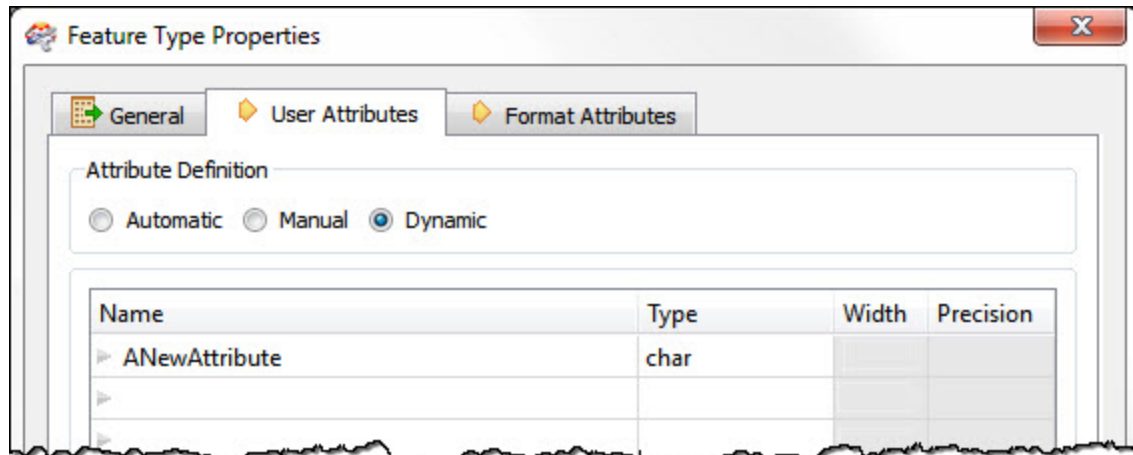
In other words, any attribute you add to the feature type definition will get added to all features output through there – regardless of source or resource schemas.

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:
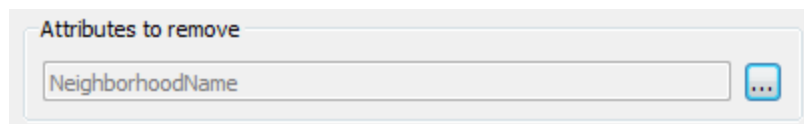
The edit […] button opens a dialog in which to select attributes that are in the source schema but that you don't want in the output. You can choose to select these existing attributes – or type in an entirely different value, if you know an attribute exists but is not exposed.

Exercise 4d Schema Handling in Dynamic Translations	
Scenario	FME author; City of Interopolis
Data	Community Mapping (Geodatabase)
Overall Goal	Create new Community Mapping dataset
Demonstrates	Schema Handling in Dynamic Translations
Starting Workspace	None
Finished Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise4d-Complete.fmw*

One of the datasets in the FMEData2015 folder is CommunityMap.gdb. This dataset is a series of base datasets used by the planning department for various community mapping tasks.

However, the planning department now wants to create a new community map dataset, with new data, but using the existing schema where possible. They also – for reasons that are unclear – want a format change to GML!

So, let's create a new workspace to handle that scenario.

1) Inspect Data

At the moment two datasets have been identified as being required in the new community mapping Geodatabase. They are:

Format: GML (Geography Markup Language)
Dataset: *C:\FMEData2015\Data\Emergency\FireHalls.gml*

Format: MapInfo TAB (MITAB)
Dataset: *C:\FMEData2015\Data\Parks\Parks.tab*

So, inspect these two datasets in the FME Data Inspector, to become familiar with them. 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

Start FME Workbench and begin by generating a workspace as follows:

Reader Format	GML (Geography Markup Language)

Reader Dataset	*C:\FMEData2015\Data\Emergency\FireHalls.gml*
Writer Format	GML (Geography Markup Language)
Writer Dataset	*C:\FMEData2015\Output\NewCommunityMap.gml*
Workflow Options	Dynamic Schema
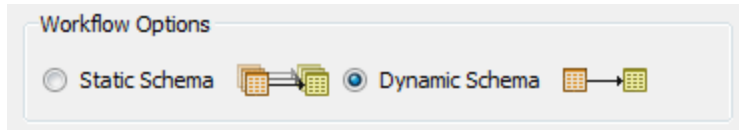
3) Add Reader

So far; so good. Now let's add a Reader for the new parks data.

Select Readers > Add Reader from the menubar. Add a Reader as follows:

Format	MapInfo TAB (MITAB)
Dataset	*C:\FMEData2015\Data\Parks\Parks.tab*
Workflow	Single Merged Feature Type

Connect the new Reader feature type up to the existing Writer feature type.

4) Add Resource Reader

One of the requirements was to use the existing schema where possible. With the firehalls it wasn't possible, since that never existed in the CommunityMaps before.

However, the parks dataset does exist in the current CommunityMaps dataset, so we'll need to use that schema.

So, select **Readers > Add Reader as Resource**.

When prompted, enter the following info:

Reader Format	Esri Geodatabase (File Geodb API)
Reader Dataset	*C:\FMEData2015\Data\CommunityMapping\CommunityMap.gdb*

Click OK and the Reader is added as a Resource:
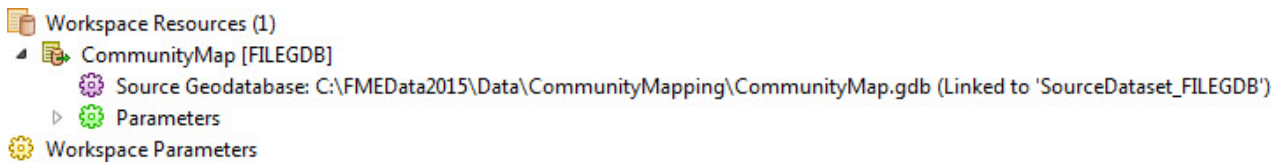
5) Adjust Dynamic Parameters

Now we need to make sure that resource is being used.

Open the properties dialog for the Writer feature type. Under Dynamic Properties click the edit […] button for Schema Sources.

Put a checkmark against CommunityMap and ensure Parks is NOT checked. Click OK.
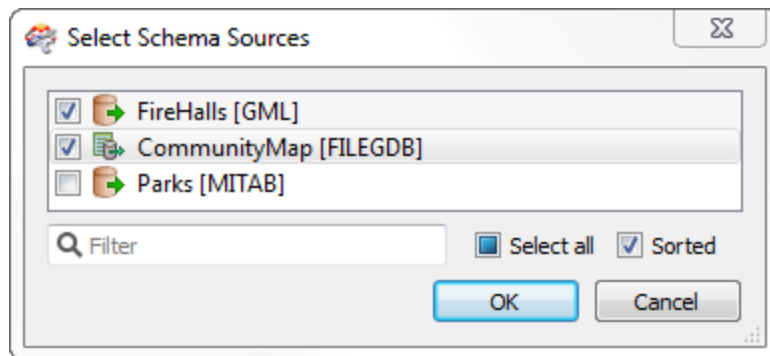
Now click the edit […] button next to Schema Definition.

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 this dialog and OK again 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 a schema of ParkName, ParkAddress, and ParkURL (even if there is no data to fill some fields yet). However, 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 Dynamic Properties click the edit […] button next to Schema Definition.
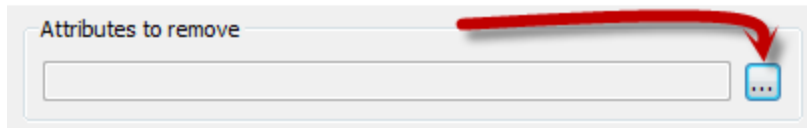
Click the […] button at the foot of the dialog next to Attributes to Remove.

In the dialog that opens, type OBJECTID into the first field. You won't be able to select it from the drop-down list because it comes from a resource reader, not a real reader.
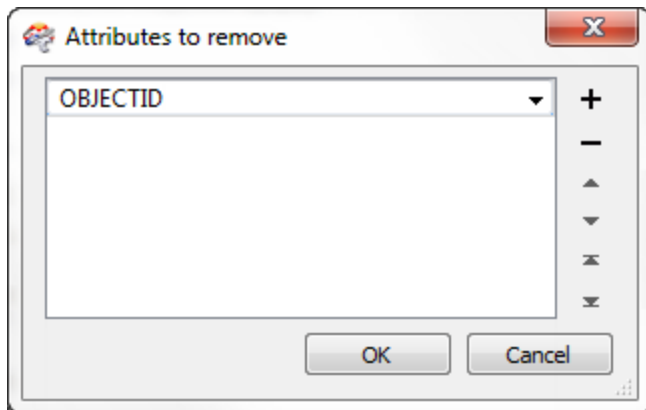
Click OK to close that dialog, then once more to close the Schema Definition dialog.

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.
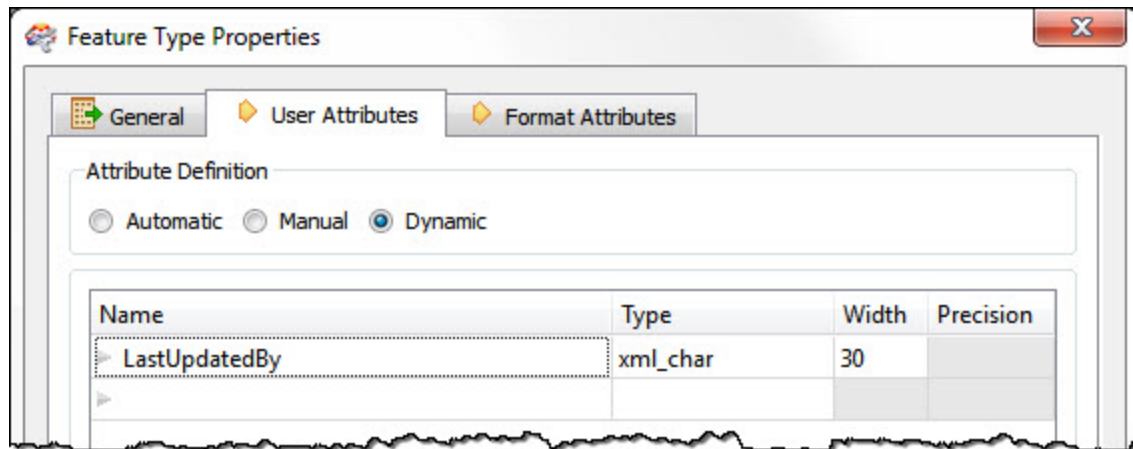
9) Re-Run Workspace

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.

Advanced Dynamic Attribute Schemas

The attribute schema in a dynamic translation can come from a multitude of places... or none!

In general, the schema for a dynamic translation will come either from 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:

	A	B	C	D	E	F
1	feature_type	attribute_name	attribute_type	geometry_type		
2	Parks	ParkId	fme_int32	fme_polygon		
3	Parks	ParkName	fme_varchar(40)			
4	Addresses	AddressId	fme_int32	fme_point		
5	Addresses	StreetNumber	fme_int32			
6	Addresses	StreetName	fme_varchar(40)			
7	Addresses	City	fme_varchar(40)			

Sheet1 / Sheet2 / Sheet3

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):
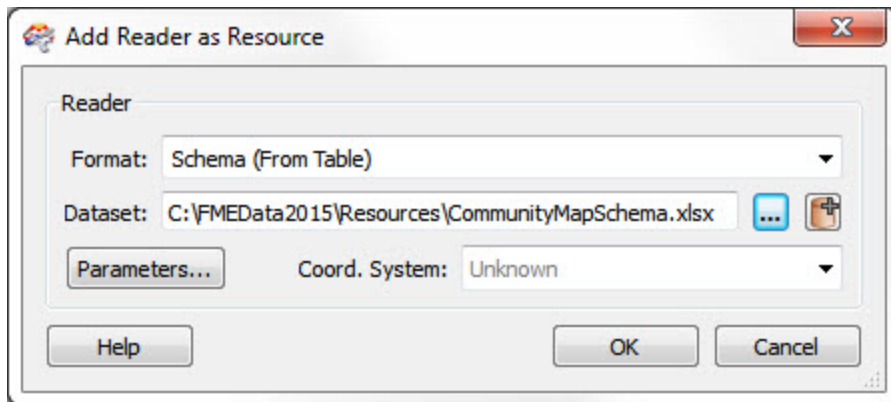
In the parameters dialog for this Reader, there are parameters that specify which fields in the table represent which parts of the schema:
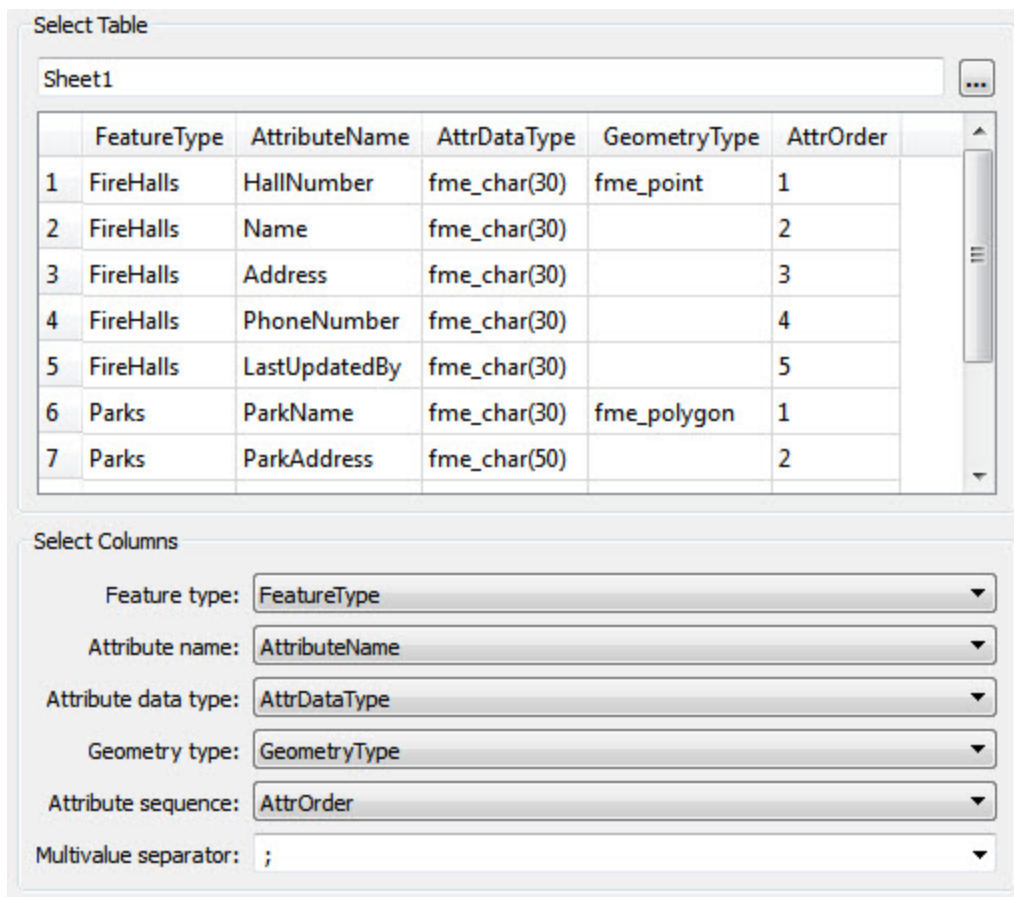
Geometry type is optional, but used in this example.

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:

The incoming attributes must then 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:

When a feature with such attributes is sent to a dynamic Writer, then this schema is used in preference to any other.

> *Sister Intuitive says…*
>
> *"The dynamic properties dialog still requires a Reader to be selected, even if you're not using its schema for the output. My advice is to add a Null format Reader and use that anyway. Then you can be sure that nothing else is being accidentally used!"*

The AttributePivoter is a very interesting transformer that makes use of this capability. Features logged after the AttributePivoter demonstrate a schema defined like this:

```
Attribute(string)    : `attribute{0}.fme_data_type' has value `fme_char(255)'
Attribute(string)    : `attribute{0}.name' has value `HoursOpen'
Attribute(string)    : `attribute{1}.fme_data_type' has value `fme_real64'
Attribute(string)    : `attribute{1}.name' has value `2 Average'
Attribute(string)    : `attribute{2}.fme_data_type' has value `fme_real64'
Attribute(string)    : `attribute{2}.name' has value `2 Count'
```

It makes sense for this transformer to do this because it creates data in a way that we are less likely to know the schema of in advance. If you have a dynamic writer and send this data to it then the output will be in this schema.

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:

- Character Fields:fme_varchar(width), fme_char(width), fme_char
- Integer Fields:fme_uint8, fme_int16, fme_uint16, fme_int32, fme_uint32, fme_int64, fme_uint64
- Numeric Fields:fme_decimal(width,decimal), fme_real32, fme_real64
- Date-Time Fields:fme_datetime, fme_time, fme_date
- Other Fields:fme_buffer, fme_boolean

Exercise 4e Advanced Attribute Schemas in Dynamic Translations	
Scenario	FME author; City of Interopolis
Data	Community Mapping (Geodatabase)
Overall Goal	Create new Community Mapping dataset
Demonstrates	Table-Based Schemas for Dynamic Translations
Starting Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise4e-Begin.fmw*
Finished Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise4e-Complete.fmw*

In Exercise 4d, we 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 do that, 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:\FMEData2015\Resources\CommunityMapSchema.xlsx.

If you don't have Excel then open it in the FME Data Inspector and switch to Table View.

The table looks like this, with Firehalls, Parks, and Zones feature types defined.

	FeatureType	AttributeName	AttrDataType	GeometryType	AttrOrder
1	FireHalls	HallNumber	fme_char(30)	fme_point	1
2	FireHalls	Name	fme_char(30)	<missing>	2
3	FireHalls	Address	fme_char(30)	<missing>	3
4	FireHalls	PhoneNumber	fme_char(30)	<missing>	4
5	FireHalls	LastUpdatedBy	fme_char(30)	<missing>	5
6	Parks	ParkName	fme_char(30)	fme_polygon	1
7	Parks	ParkAddress	fme_char(50)	<missing>	2
8	Parks	ParkURL	fme_char(100)	<missing>	3
9	Parks	LastUpdatedBy	fme_char(30)	<missing>	4
10	Zones	ZoneName	fme_char(30)	fme_polygon	1
11	Zones	ZoneCategory	fme_char(30)	<missing>	2
12	Zones	LastUpdatedBy	fme_char(30)	<missing>	3

2) Start Workbench

Open the workspace C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise4e-Begin.fmw

This is the workspace you created in Exercise 4d.

3) Delete CommunityMap Resource Reader

We no longer need the CommunityMap Resource Reader, so locate it in the Navigator window, right-click on it, and choose Delete.
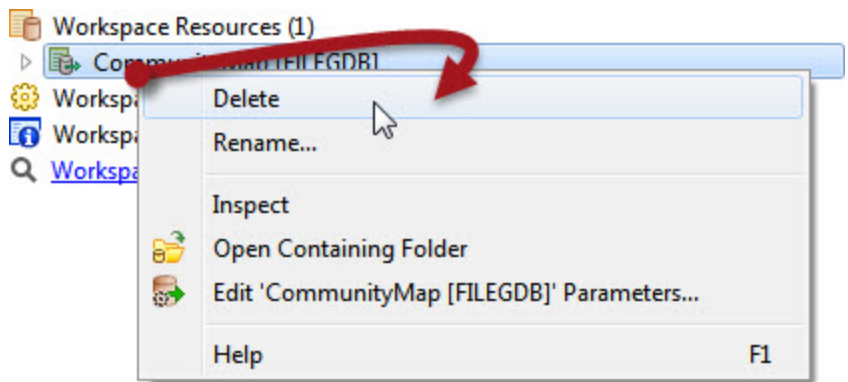
When prompted, click OK to confirm that all references relating to this dataset will also be removed.

4) Add Excel File as Reader Resource

Now select **Readers > Add Reader as Resource**. In the dialog that opens choose:

Reader Format	Schema (From Table)
Reader Dataset	*C:\FMEData2015\Resources\CommunityMapSchema.xlsx*

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:

Select Table

Sheet1

	FeatureType	AttributeName	AttrDataType	GeometryType	AttrOrder
1	FireHalls	HallNumber	fme_char(30)	fme_point	1
2	FireHalls	Name	fme_char(30)		2
3	FireHalls	Address	fme_char(30)		3
4	FireHalls	PhoneNumber	fme_char(30)		4
5	FireHalls	LastUpdatedBy	fme_char(30)		5
6	Parks	ParkName	fme_char(30)	fme_polygon	1
7	Parks	ParkAddress	fme_char(50)		2

Next select the appropriate fields to match to the required parameters (for example Feature Type = FeatureType).

Select Columns

Feature type:	FeatureType
Attribute name:	AttributeName
Attribute data type:	AttrDataType
Geometry type:	GeometryType
Attribute sequence:	AttrOrder
Multivalue separator:	;

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].
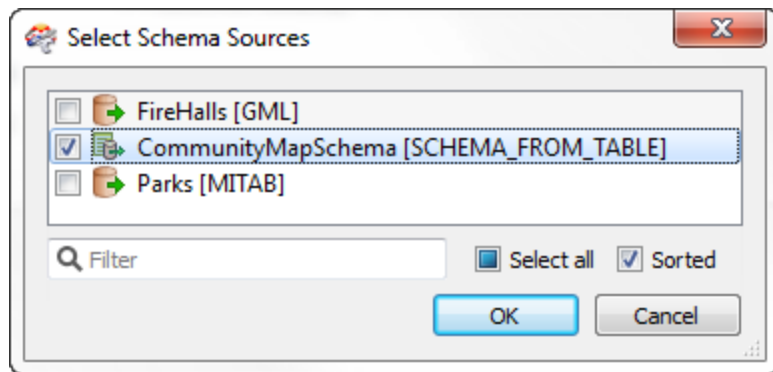
Click OK and OK again to close these dialogs.

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:

Reader Format	MapInfo TAB (MITAB)
Reader Dataset	*C:\FMEData2015\Data\Zoning\Zones.tab*

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.

	ZoneName	ZoneCategory	LastUpdatedBy
1	M-2	Industrial	\<missing\>
2	M-1	Industrial	\<missing\>
3	M-2	Industrial	\<missing\>
4	M-1	Industrial	\<missing\>
5	M-1	Industrial	\<missing\>
6	M-1A	Industrial	\<missing\>
7	M-1	Industrial	\<missing\>
8	M-2	Industrial	\<missing\>
9	M-2	Industrial	\<missing\>

Advanced Task

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).

The planning team have decided they should rename some attributes, so open the spreadsheet and rename the following for the FireHalls feature type:

- Name to HallName
- Address to HallAddress
- PhoneNumber to HallPhone

Save the spreadsheet.

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.

	HallNumber	HallName	HallAddress	HallPhone
1	1	\<missing\>	\<missing\>	\<missing\>
2	2	\<missing\>	\<missing\>	\<missing\>
3	3	\<missing\>	\<missing\>	\<missing\>
4	4	\<missing\>	\<missing\>	\<missing\>
5	6	\<missing\>	\<missing\>	\<missing\>

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.

	A	B
1	OldAttrName	NewAttrName
2	Name	HallName
3	Address	HallAddress
4	PhoneNumber	HallPhone

In sheet 2 of the spreadsheet, enter:

OldAttrName	NewAttrName
Name	HallName
Address	HallAddress
PhoneNumber	HallPhone

Then save the spreadsheet.

Add a SchemaMapper transformer to the workspace, with both output ports connected to the output feature type.
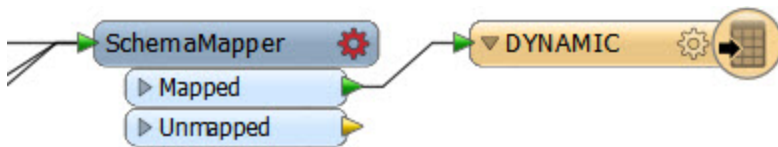
Open the parameters dialog. For the format select the edited Excel file:
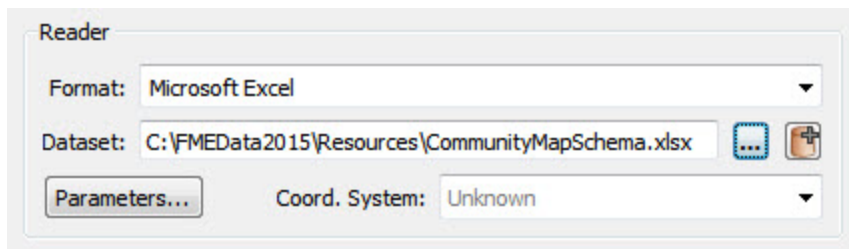
Click the parameters button. Under Sheets to Read, turn off Sheet1 and make sure Sheet2 is selected. Click OK to close the dialog.
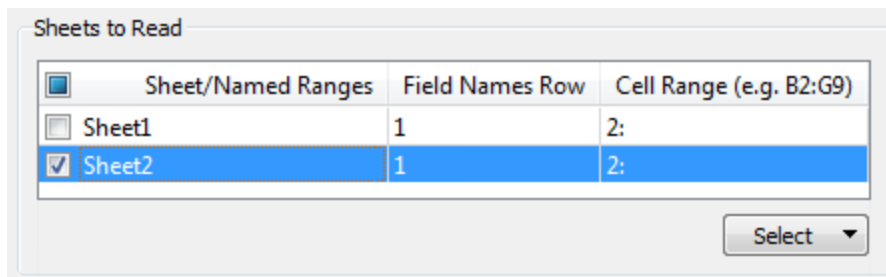
Click the Next button. In the next dialog, select Add > Attribute Map.
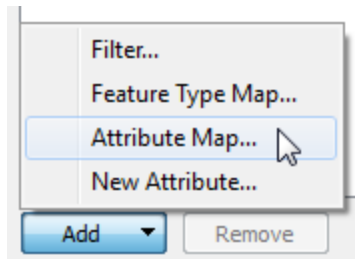
When prompted, select OldAttrName as the source field and NewAttrName as the Destination field. Check the box to Remove the original attributes (i.e. this is a renaming, not copying):
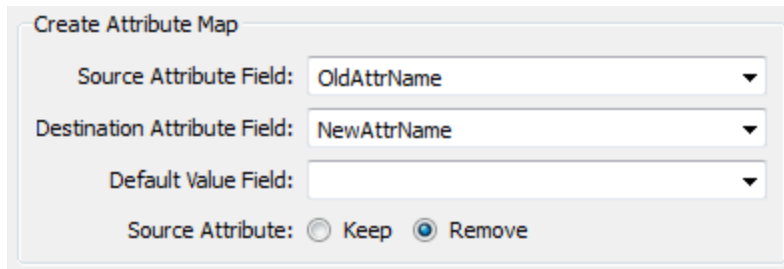
Click OK to close this dialog, then click Finish. Now save and run the workspace again.

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.

	HallNumber	HallName	HallAddress	HallPhone	LastUpdatedBy
1	1	Vancouver Fire ...	900 Heatley Av	604-665-6001	<missing>
2	2	Vancouver Fire ...	199 Main St	604-665-6002	<missing>
3	3	Vancouver Fire ...	2801 Quebec St	604-665-6003	<missing>
4	4	Vancouver Fire ...	1475 W 10th Av	604-665-6004	<missing>
5	6	Vancouver Fire ...	1001 Nicola St	604-665-6006	<missing>
6	7	Vancouver Fire ...	1090 Haro St	604-665-6007	<missing>
7	8	Vancouver Fire ...	895 Hamilton St	604-665-6008	<missing>
8	12	Vancouver Fire ...	2460 Balaclava St	604-665-6012	<missing>

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.

Chapter 5 - Advanced Attribute Handling

Constructing Values

Constructing Values goes one step further than setting an attribute with a simple, fixed value.

FME has various functions that allow the workspace author to create a simple, fixed value for an attribute.

For example, here an author is using the AttributeCreator transformer to set the value of an existing attribute and to create a new attribute with a fixed value:
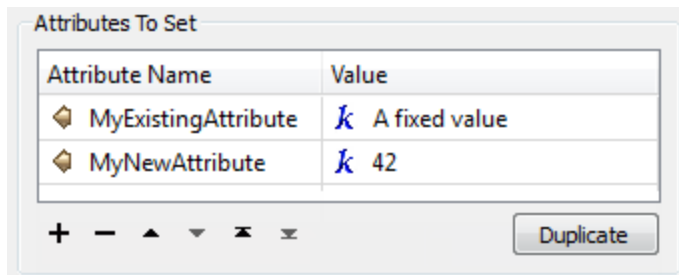
However, in many cases the workspace requires a value to be constructed out of a number of existing values, or perhaps an arithmetic calculation, and FME is capable of doing that as well.

There are, basically, two methods to constructing attribute values: *Transformer Construction* and *Integrated Construction*.

Transformer Construction

Transformer construction is a basic method of constructing attribute values. The author sets simple values in a transformer, but uses two or more transformers in a series in order to build a more complex value.

For example, here the workspace author needs to create a label in their spatial data.

First they create an attribute value by calculating a number, concatenate it onto a string attribute, trim the result, and then use that as the content for a LabelPointReplacer transformer.
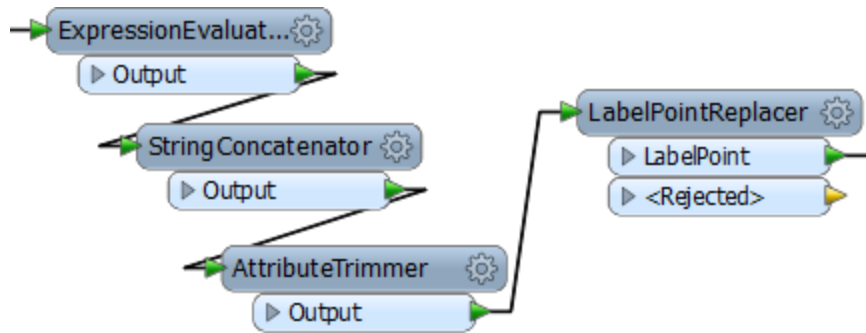
This is an acceptable way of working that is self-documenting and easy to understand; but it is also a fairly crude method and results in the use of more transformers than is perhaps necessary.

Integrated Construction

Integrated construction is a more elegant method of constructing attribute values. Instead of using a string of transformers, the author uses functionality built into a single transformer's parameter dialog window.

In short, the attribute value is constructed at its point of use; it's a more efficient method and reduces the number of transformers cluttering up the workspace canvas.

Most transformer parameters have one of two pieces of inbuilt functionality that allow this to take place: a *Text Editor* or an *Arithmetic Editor*.
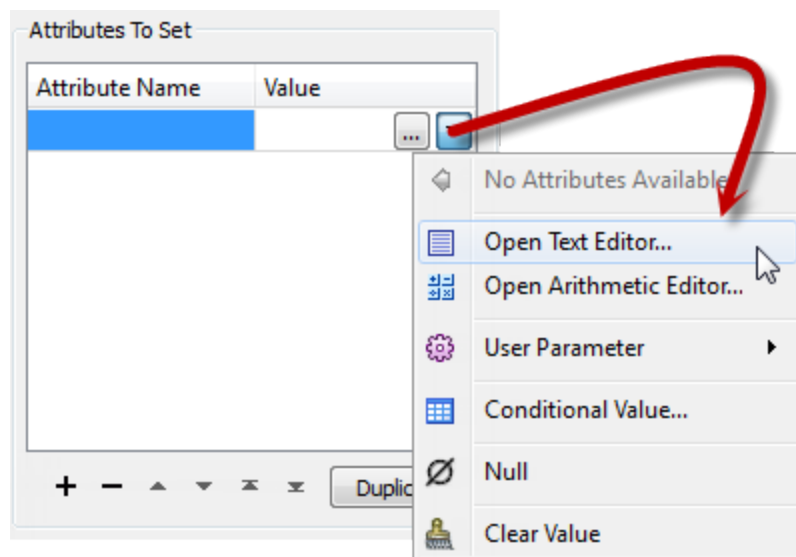
The text editor allows values to be constructed from strings. It includes access to all FME feature attributes and user parameters, plus special characters, FME Functions, and various string manipulation related functions.

The arithmetic editor allows values to be calculated from arithmetic calculations. Again, it includes access to all FME feature attributes, user parameters, and FME Functions; but additionally it gives access to a series of mathematical functions and operators.

Professor Lynn Guistic says…

"Although we're talking about attributes here, the same functionality applies to any transformer parameter.

For example, the label parameter for the LabelPointReplacer can be constructed using a text editor instead of a series of transformers.

We're only discussing it here in terms of "attributes" because traditionally you would construct an attribute and supply it to the parameter. Now everything is built into that parameter."

Text Editor

The Text Editor is a tool for string-based construction of attributes. In it, the workspace author can construct values out of fields such as the following:

- String Constants
- FME Feature Attributes
- User or System Parameters
- Special Characters

The text editor window is a free-form dialog into which the author can construct almost any form of string. Here, for example, the author is constructing a string that describes a cell phone tower. Such a string could be applied as a label, written as an attribute value, or used in many other ways.
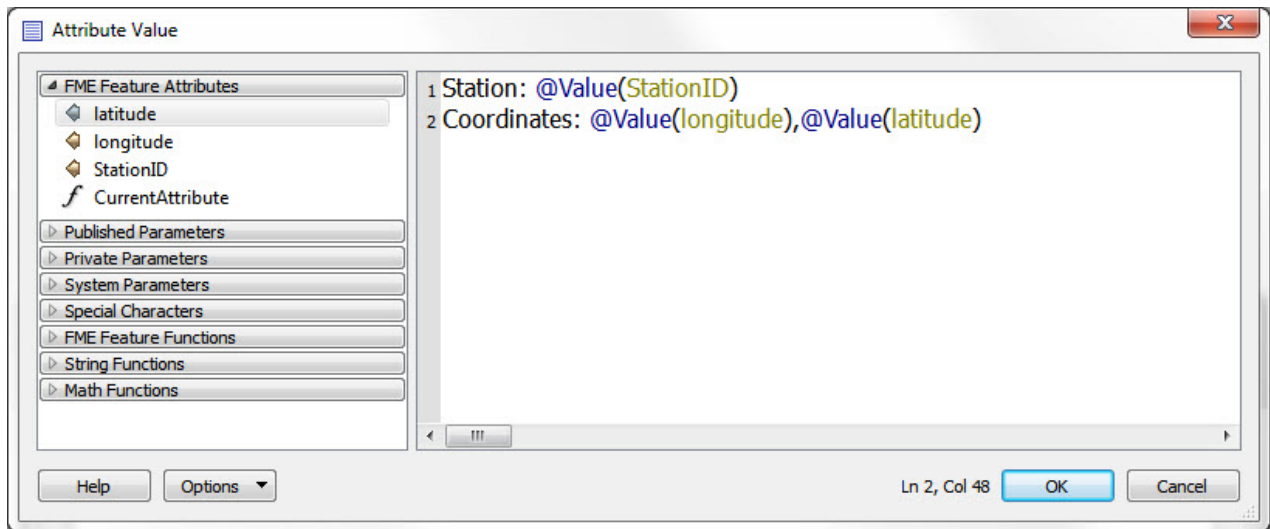
Constant values can be entered directly into the expression field.

Fixed items are generated by using the left-hand menu. In addition to the ability to use attributes, parameters, and special characters, the advanced editor allows the author to make use of special functions.

These are added to the expression by locating them in the menu and either double-clicking that entry or dragging it into the expression:
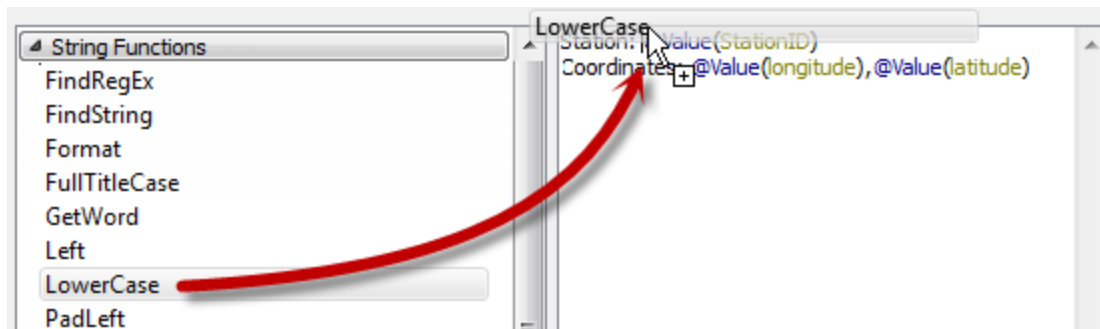
FME Feature Functions are the underlying functions used by FME transformers. By accessing these, the author is reaching directly into core FME functionality.

All FME functions are of the form `@functionname()` where the part inside parentheses is a function parameter.

For example the function `@Value(Roads)` returns the value of the attribute called Roads.

```
⊿ FME Feature Functions
    Abort
    Area
    CoordSys
    Count
    Dimension
    Evaluate
    GeometryType
    Length
    NumCoords
    UUID
    Value
    XValue
    YValue
    ZValue
```

Not all functions have parameters, or those parameters are not always compulsory; for example the function @Area() does not need to take a parameter.

> *Professor Lynn Guistic says…*
>
> *"FME functions can be used in the text editor to calculate values on the fly, instead of using a transformer.*
>
> *For example, an author wants to create an attribute whose contents include the area of the feature in the form "<area> sq. metres"*
>
> *They could calculate the area with an AreaCalculator and use @Value(_area) sq. metres or they could cut out that transformer and go directly with: @Area() sq. metres"*

The editor also has a set of string functions and mathematical functions. The string functions carry out operations like formatting, extracting, trimming, and padding. The mathematical functions are the usual advanced arithmetic functions such as int, cos, and log.

The string functions are mostly based around Tcl, while the arithmetic functions around C.

Arithmetic Editor

The Arithmetic Editor dialog is a tool for building arithmetic expressions.

Like the text editor, the arithmetic editor has the ability to calculate values using constant values, feature attributes, parameters, and other functions.

However, the arithmetic editor works entirely with numbers to calculate a numeric value. The implications of this are:

- All attribute values used must be numeric, or converted to numeric inside the expression by using a function.

- The only FME functions and string functions available are those that return a numeric value; for example @Area(), @Count(), and @StringLength()

- The value returned to the workspace will be a numeric type, not a string.

For example, here a workspace author is calculating the property tax for a set of features using the attributes TaxBand and AssessedValue, in conjunction with the @Area() function.
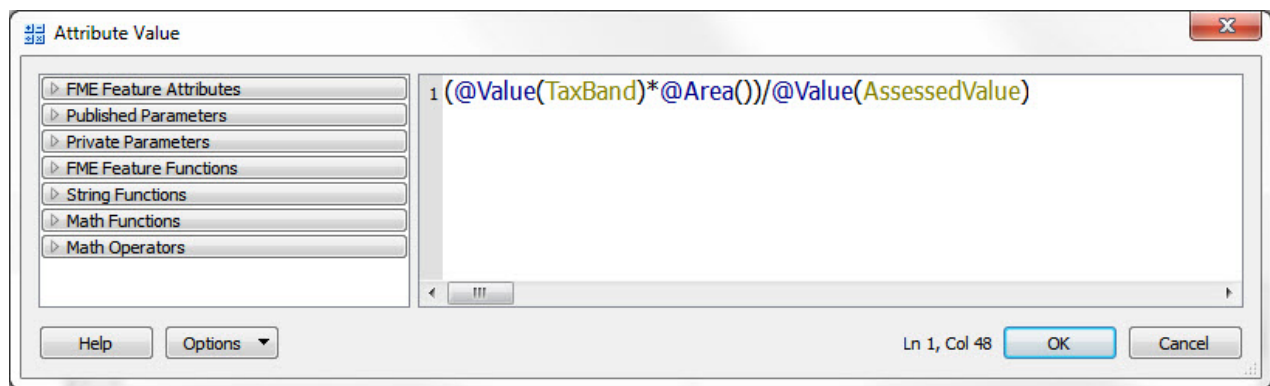
Professor Lynn Guistic says…

"At this point two questions are probably uppermost in your mind.

Firstly, why do some transformer parameters have options for both the text and arithmetic editor (like the AttributeCreator) whereas others have just the text editor or just the arithmetic editor?

That's easy. The editor is matched to the type of value expected by that parameter. For example, the LabelPointReplacer transformer has Label and Label Height parameters. The Label parameter can access the text editor, because it expects a string. The Label Height parameter can access the arithmetic editor because it expects a number.

The AttributeCreator transformer allows access to both because it doesn't rely on a particular type. It is happy with either numeric or string values.

Secondly, you're probably wondering why there are math functions inside the text editor and how you can use them.

That's simple. The FME function @Evaluate() carries out a mathematical calculation. So if you need to calculate a numeric value as part of a string, then use that function; like so:"

The property: @Value(AddressId)
is @Evaluate(@Value(DistanceInMetres)*3.2808) feet in length

Advantages and Disadvantages

There are some obvious advantages and disadvantages to using integrated construction of attributes, compared to a sequence of transformers.

A sequence of transformers is self-documenting.
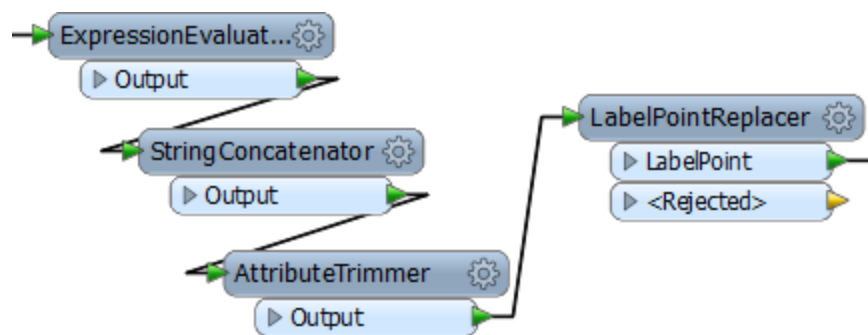
A casual user could look at the workspace and understand what is happening.

Integrating this into a single parameter in a single transformer leads to a more closed workspace. A casual user would have to investigate the workspace closely to understand how it works.

On the other hand, integrated functions produce much more clean and elegant solutions. They take up less space on the canvas and are generally less cluttered.

So the decision on which method to use depends as much on your confidence in using FME, and how much the workspace might need to be shared with others, as anything else.

Exercise 5a Integrating Attribute Construction	
Scenario	FME author; City of Interopolis
Data	Traffic Signals (DWG)
Overall Goal	Create a KML dashboard linked to FME Server
Demonstrates	Integrated attribute construction.
Starting Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise5a-Begin.fmw*
Finished Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise5a-Complete.fmw*

The Engineering department at the City of Interopolis has invested in FME Server and is now creating a KML dashboard to provide access to the information FME Server provides.

In this workspace they translate a DWG dataset of traffic signals to KML. The KML output will contain a URL that points to a workspace on FME Server. When the URL hyperlink is clicked it will cause a workspace to run and FME Server to return information about the traffic signal.

Today, however, your task is not to work on the Server workspace; rather it is to help create the hyperlink that points to that workspace.

1) Inspect Source Data

Inspect the source data for the workspace. You can find it at:

Format   Autodesk AutoCAD DWG/DXF
Dataset   C:\FMEData2015\Data\Engineering\TrafficSignals.dwg

When reading this data it's preferable to set the parameter Group Entities By to 'Attribute Schema.'

Notice how each traffic signal is represented by a point feature with a text object that specifies the ID number for the signal:

2) Start Workbench

Open the workspace C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise5a-Begin.fmw.

The first part of the workspace concerns getting the ID number from the text label and onto the point feature. It's not our main point of concern, so we can pretty much ignore this part:
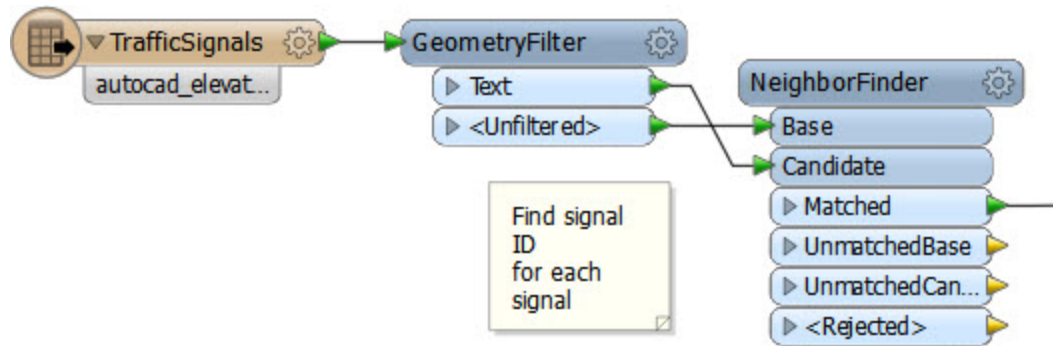
The part of the workspace that does concern us is the last four transformers:
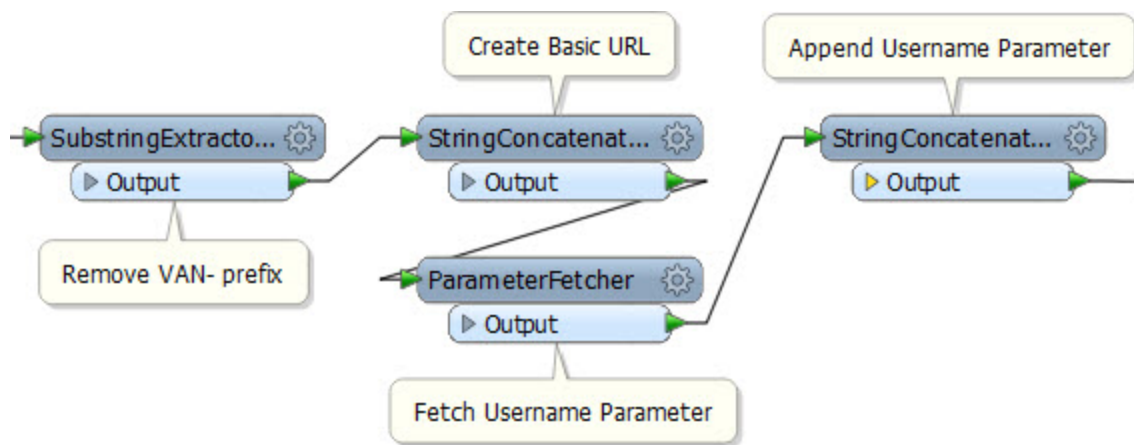
Inspect each transformer in turn, examining what it does and what parameters are set.

You should quickly see that this section is constructing a hyperlink that points to FME Server. From what we've learned about integrated attribute construction, we know that it should be possible to improve upon this design.

3) Place AttributeCreator

Place an AttributeCreator transformer with a parallel connection out of the NeighborFinder matched port.

It's important to ensure it is connected in to the current flow of data, else no attributes will be available.
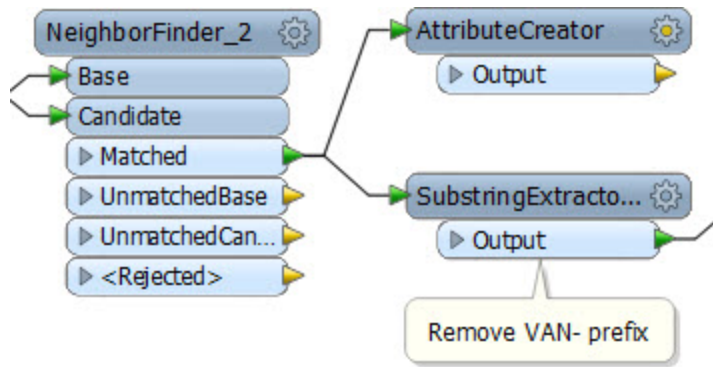
Open the parameters dialog. For attribute name enter SignalURL (to match the output schema).

Under Value, click the drop-down menu arrow and choose the option Open Text Editor:
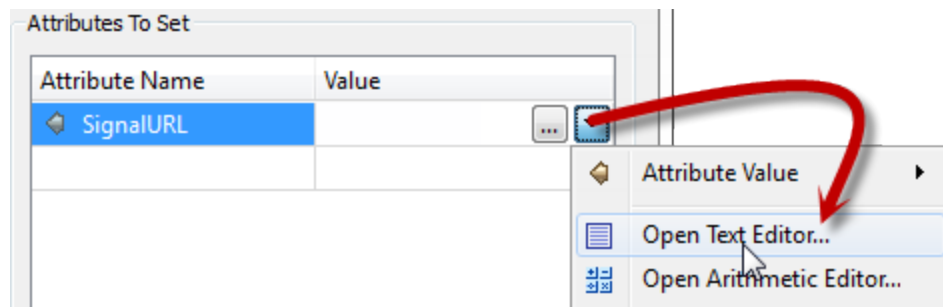
4) Create Attribute Value

In the expression dialog enter the text:

http://interopolis.org/fmedatastreaming/Engineering/signalinfo.fmw?signalid=

If it's easier, this can simply be copied from the existing StringConcatenator transformer; although it doesn't really matter if you get it wrong because it's just an exercise(!)

Now we need a function to replace the SubstringExtractor transformer. In the left-hand set of menus, click on String Functions to expand the list of available functions.

Locate the substring function and double-click it to insert it into the expression.

```
◢ String Functions
   FindRegEx
   FindString
   Format
   FullTitleCase
   GetWord
   Left
   LowerCase
   PadLeft
   PadRight
   ReplaceRegEx
   ReplaceString
   Right
   StringLength
   Substring
   TitleCase
```
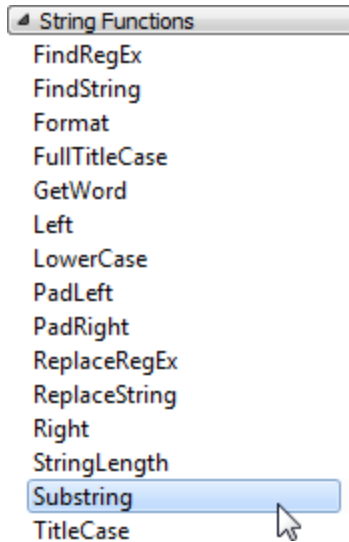
The expression will now look like this, with the <STRING> part pre-selected:

http://interopolis.org/fmedatastreaming/Engineering/signalinfo.fmw?signalid=@Substring(<STRING>,<INT>,<INT>)

The string we want to process is the value of the attribute fme_text_string. So, in the left-hand set of menus, click on FME Feature Attributes to expand the list of available attributes.

Locate the attribute fme_text_string and double-click it to insert it into the expression.

```
◢ FME Feature Attributes
   ◈ _angle
   ◈ _candidate_angle
   ◈ _candidate_label_angle
   ◈ _closest_base_x
   ◈ _closest_base_y
   ◈ _closest_candidate_x
   ◈ _closest_candidate_y
   ◈ _distance
   ◈ autocad_elevation
   ◈ fme_rotation
   ◈ fme_text_size
   ◈ fme_text_string
 ▷ Published Parameters
```
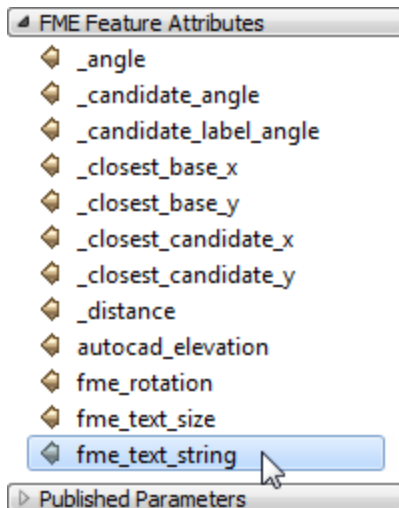
The latter part of the expression now looks like this, with the attribute filled in:

/signalinfo.fmw?signalid=@Substring(@Value(fme_text_string),<INT>,<INT>)

The values of the two <INT> fields are the same as used in the SubstringExtractor transformer. In this case the values are 4 (the fifth character in) and -1 (the end of the string).

So, manually select each <INT> in turn and replace them with 4 and -1. The expression now looks like this:

/signalinfo.fmw?signalid=@Substring(@Value(fme_text_string),4,-1)
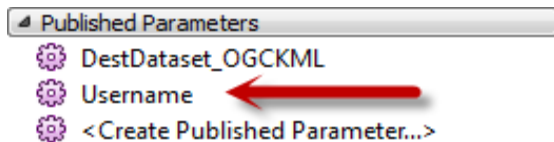
On the end of the expression, manually type &username=

/signalinfo.fmw?signalid=@Substring(@Value(fme_text_string),4,-1)&username=

And, finally, in the left-hand menus click on Published Parameters to expand the list of available parameters.

Double-click on Username to insert that user parameter into the expression:

The last part of the expression now looks like this:

/signalinfo.fmw?signalid=@Substring(@Value(fme_text_string),4,-1)&username=$(Username)

Notice how attributes are denoted by an @Value() function, whereas parameters are denoted as $(<parametername>)

Click OK to close the dialog, and then OK again to close the main AttributeCreator dialog.

5) Clean Workspace

Now clean up the workspace by connecting the newly created AttributeCreator to the output feature type, and deleting the other four transformers that already existed.
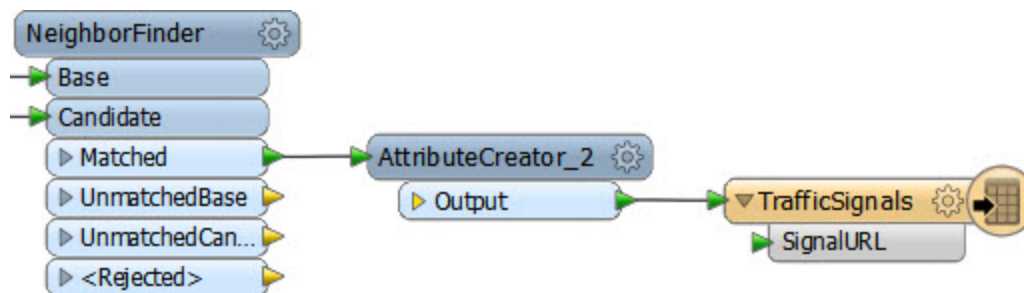
Add a little annotation to denote that this transformer now creates the URL:

Save and run the workspace to confirm that it does still work correctly.

Compare the original workspace to the new one. Is one "better" than the other? If so, why?

What are the advantages and disadvantages of each workspace?

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

The Basic FME Desktop training course includes a section on Conditional Filtering, and explains how to apply conditions to data in order to split or divide it into multiple streams.

In this example, the AttributeFilter is being used to divide data on the basis of a postal code attribute value:
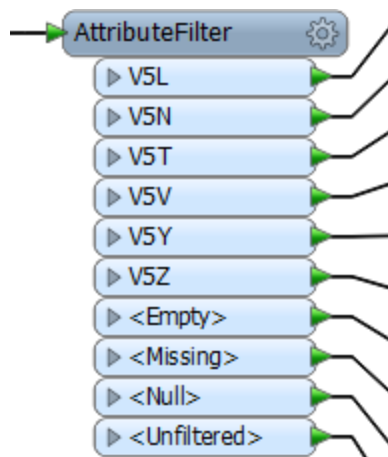
Similarly, a Tester or TestFilter transformer can be used to divide data in various ways.

So filtering transformers divide data up into separate streams in the workspace. The question is: what happens when the scenario requires a new attribute to be created based upon the original attribute value?

Technically, the following works, but it is far from elegant:
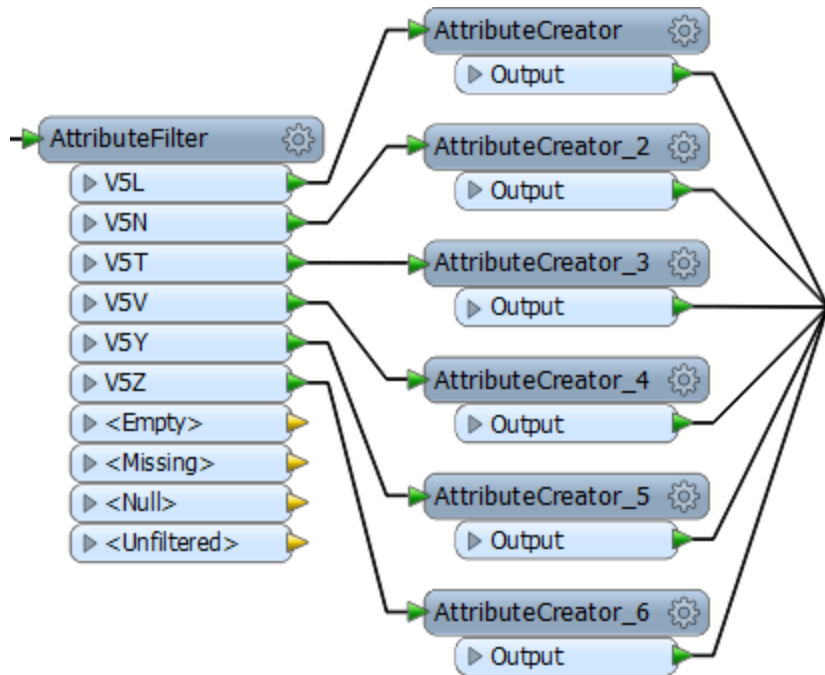
There are just far too many AttributeCreator transformers. Imagine if there were 100 values to handle! The workspace would be enormous!

In that scenario, the solution would be to use a simple AttributeValueMapper transformer:
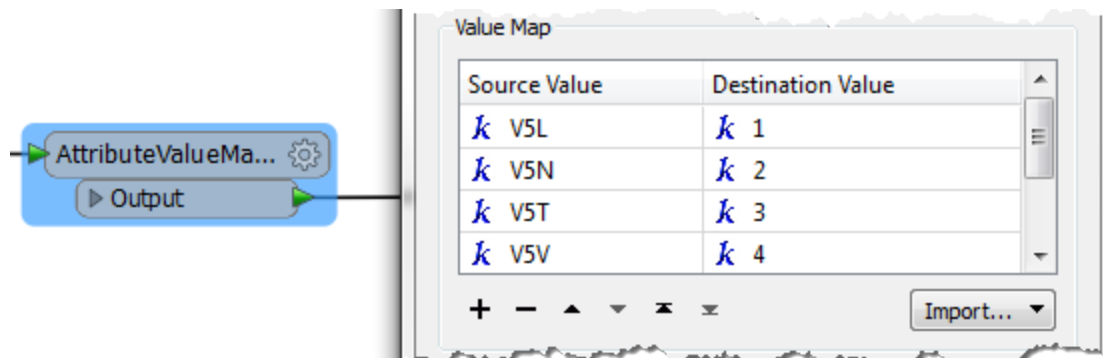
However... what if the original attribute required a more complex condition to carry out the mapping? In that scenario something even more advanced would be required: Conditional Attribute Values.
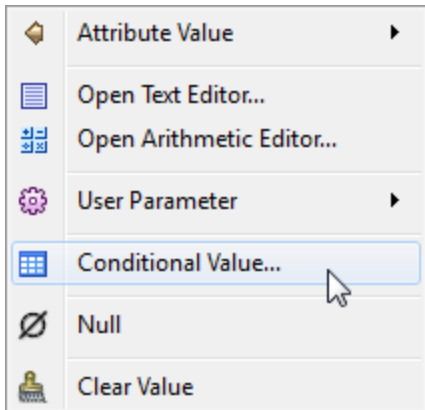
What are Conditional Attribute Values?

Conditional attribute values are when the author sets an attribute value, but conditional upon a number of tests 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 AttributeCreator, it appears like so:
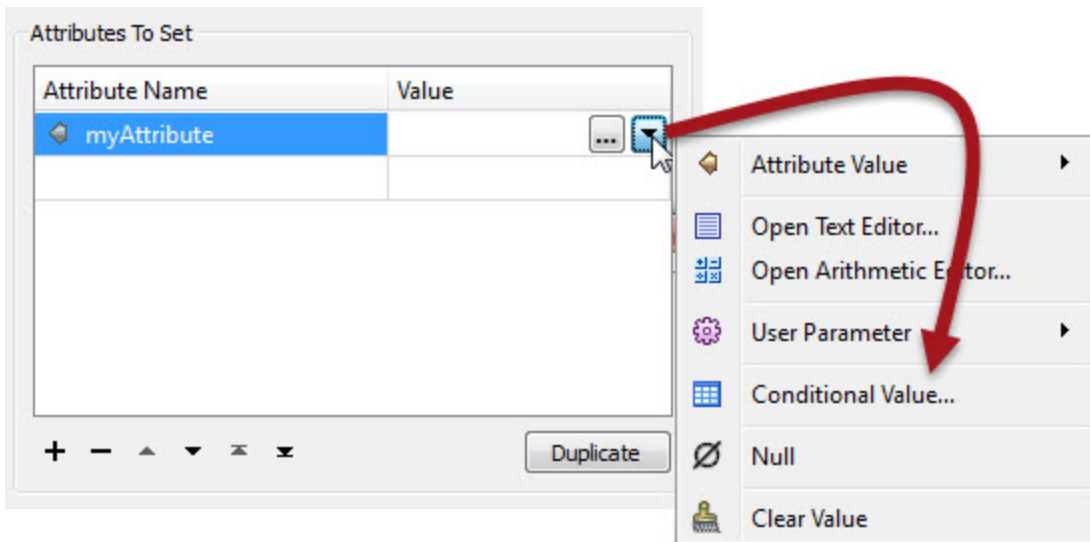
The Conditional Definition dialog looks like this:
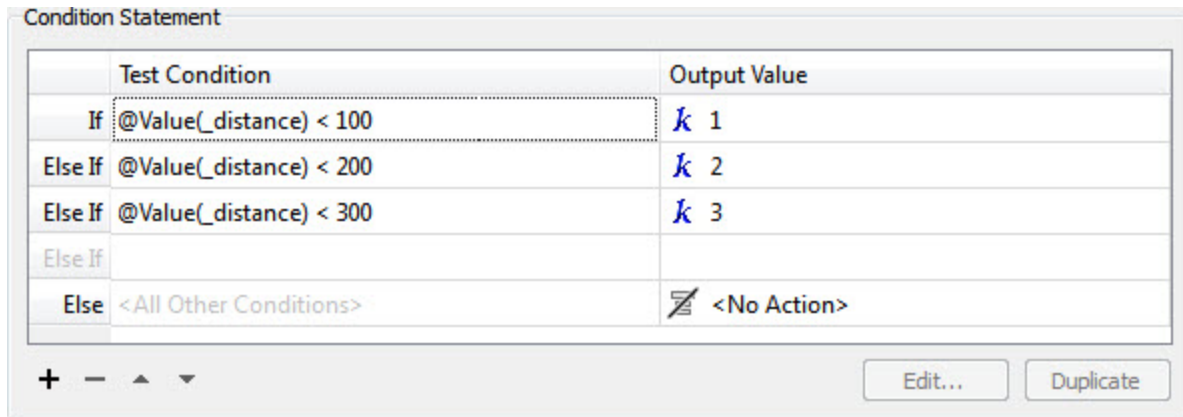
Like the AttributeValueMapper, a series of conditions map to different values. However, in contrast to the AttributeValueMapper, this dialog allows much more complex conditions than a simple 1:1 mapping. That's because full test capabilities are built into this dialog.

The test capabilities are accessed by double-clicking in the Test Condition field. That opens up a dialog almost identical to that used by the Tester and TestFilter. A number of complex conditions can be defined in order for this test to be passed.

The output value for a passing feature can be set here at the foot of the dialog, or back in the Conditional Definition dialog:
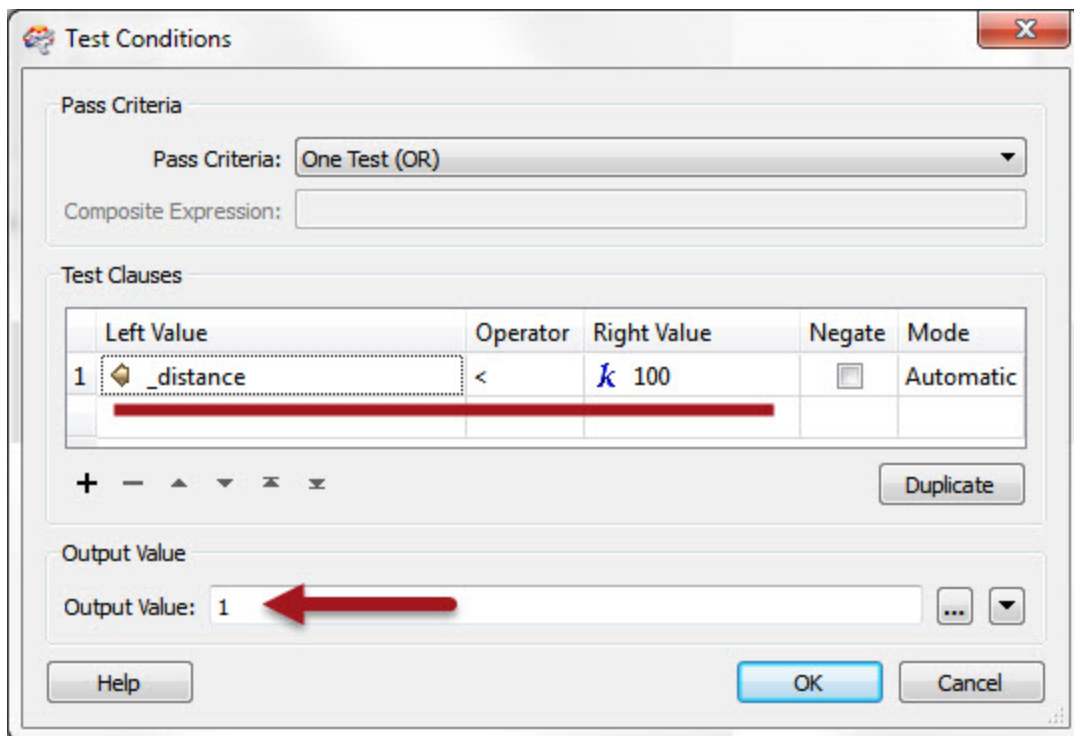
The Output Value field does not necessarily need to be a value. It can be any action on the usual drop-down menu, including Null or Stop Translation.

When the conditions are set then the original dialog – in this case an AttributeCreator – 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, when the conditions are more complex than can be handled in a simple AttributeValueMapper or RangeValueMapper transformer.

Professor Lynn Guistic says…

"If you're using the ?: operator in an arithmetic editor, then well done you! But that's now a scenario where you could instead use the conditional values technique."

Exercise 5b Conditional Attribute Values	
Scenario	FME author; City of Interopolis
Data	Addresses (Geodatabase) Coastal Zones (GML) Elevation (CDED/USGS)
Overall Goal	Calculate tsunami flood risk for addresses
Demonstrates	Conditional Attribute Values
Starting Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise5b-Begin.fmw*
Finished Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise5b-Complete [1-3].fmw*

A colleague is working on a workspace to calculate the flood risk due to tsunami 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:

		Elevation (m above sea level)		
		0-10m	10-25m	25-60m
Distance from shoreline (m)	100m	1	2	3
	200m	2	3	4
	300m	3	4	5

Your colleague has created the workspace up until the point at which the calculations need to start, and has asked for your assistance in finishing the project.

1) Start Workbench

Open the workspace *C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise5b-Begin.fmw*.

To find out what data we are dealing with, add Inspector transformers throughout the workspace and then run it - or else use Run > Run with Full Inspection from the menubar.

You'll probably want to inspect the source features 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.
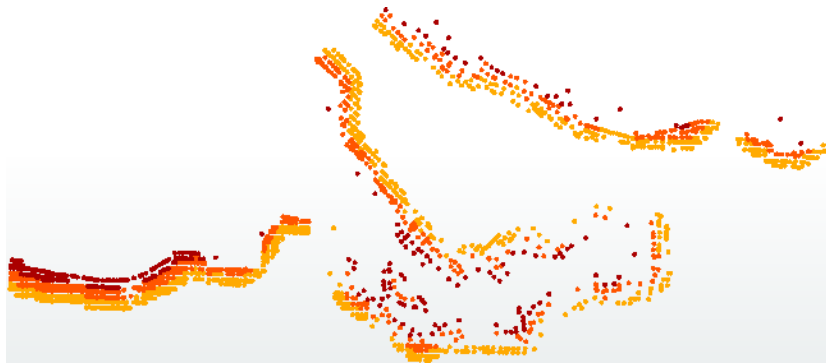
You'll see how the addresses are assigned a zone denoting their distance from the shoreline, and also possess an elevation.

2) Assess Methodology

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 consider:

- Can the data be mapped directly, or does it need filtering first?

  The zone is fairly easy to handle, 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. So mapping elevation values would need 60 x 5 = 300 combinations!

- Which transformers would you use?

  If you want to filter the data then the Tester and TestFilter transformers seem to be the most obvious candidates, with maybe the AttributeRangeFilter. To map data, either the AttributeValueMapper or AttributeRangeMapper appear to be the best. But as a whole, we're looking to set attribute values, so why not just use the AttributeCreator?

- 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? Consider also the need to maintain the workspace. For the purposes of this exercise we'll assume that all methods have an equally efficient performance.

Try placing a few likely transformers into the workspace and see which might be suitable. Which will do the job, without needing too many transformers? Would conditional attribute values work?

3) Select Methodology

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 one-step process, but – because the data is being filtered – still 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.

The Simple Filtering Method

This is a two-step process involving an AttributeFilter and several AttributeRangeMappers.

4) Place AttributeFilter

Place an AttributeFilter connected to the AttributeRenamer.
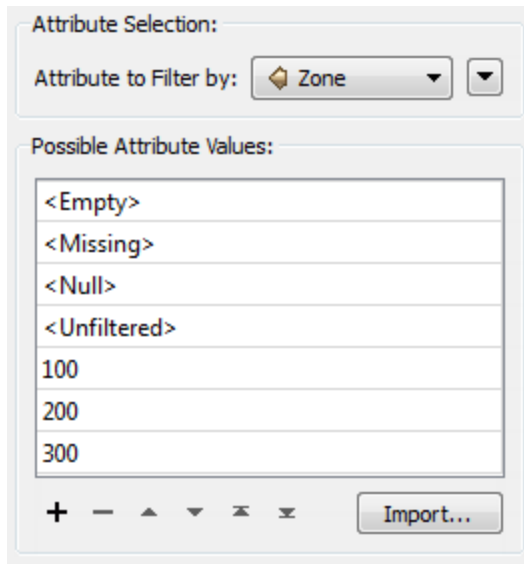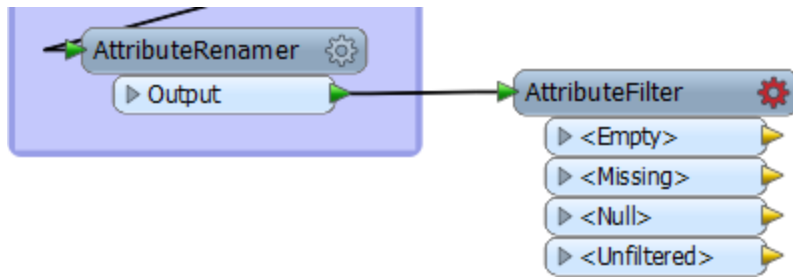
Open the parameters dialog.

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.

5) Add AttributeRangeMapper

Add an AttributeRangeMapper transformer and connect it to the 100 output port of the AttributeFilter.
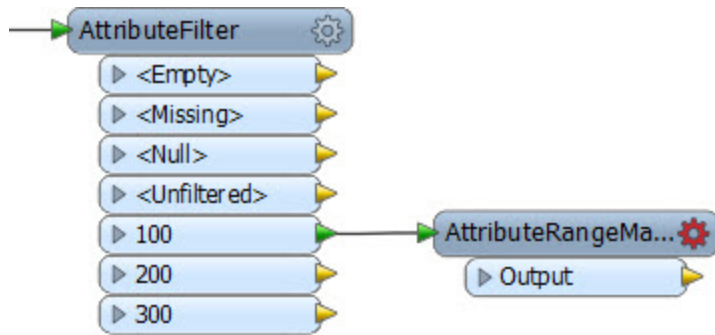
Open the parameters dialog. As you'll see this is a lookup table that involves ranges. We should be able to map the elevation range to a final flood risk using the information in the original table.

So, select Elevation as the Source Attribute.

Enter FloodRisk as the Output Attribute
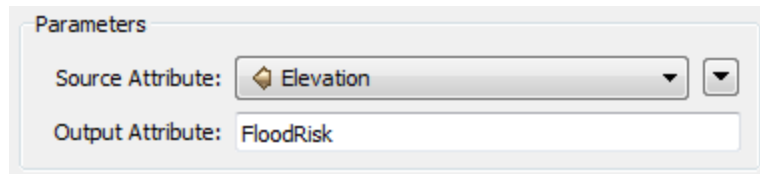
In the Range Lookup Table, enter the From-To values as follows:

0	10
10	25
25	60
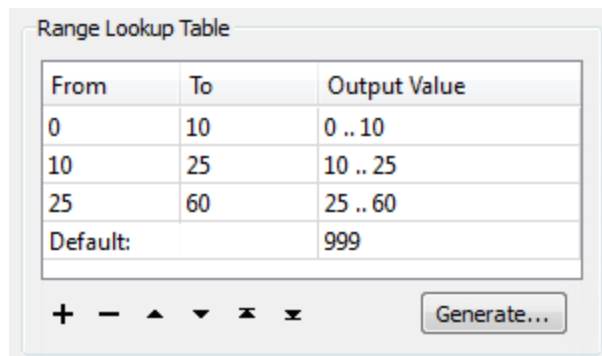
If an elevation falls exactly on one value (for example 25) it will be counted in the lower band (i.e. 10-25).

In the Output Value fields, enter the values from the table for where Zone=100. These should be 1, 2, 3.

Enter 999 as the Default, so that any features whose elevation does not match, for whatever reason, is flagged appropriately.

From	To	Output Value
0	10	1
10	25	2
25	60	3
Default:		999

Click OK to close the dialog.

6) 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:
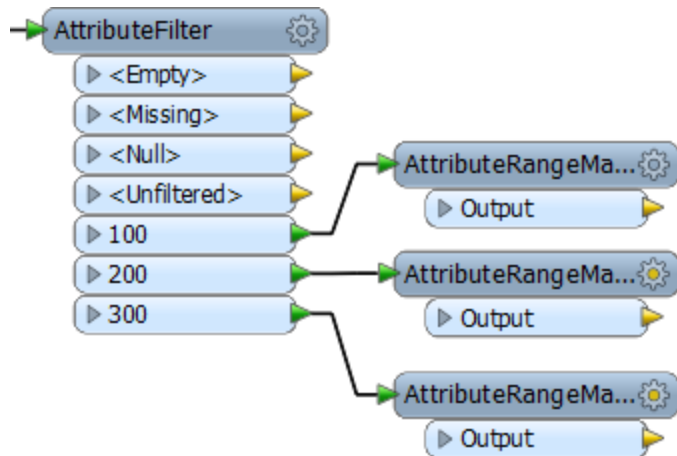
There are what we can call the 100m zone AttributeRangeMapper, the 200m zone AttributeRangeMapper, and the 300m zone AttributeRangeMapper.

However, what still needs to be done is that the Output Values need to be changed in each of these in accordance with the original table of calculations.

So, open the parameters dialog for each AttributeRangeMapper transformer in turn and change the output value fields.

The values will be:

- 100m Zone: 1, 2, 3
- 200m Zone: 2, 3, 4
- 300m Zone: 3, 4, 5

7) Add Inspector

Place a single Inspector transformer and connect each AttributeRangeMapper output to it.
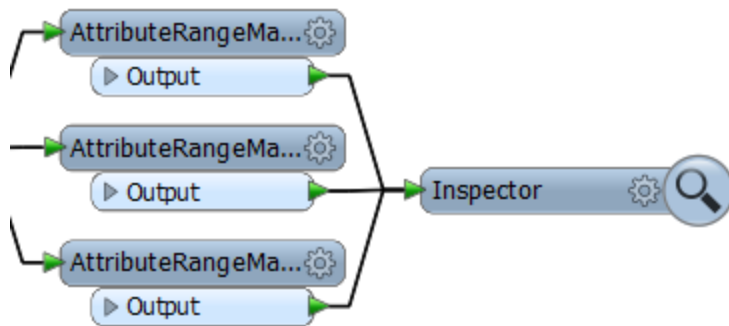
Open the Inspector parameters dialog and under Group-By select the newly created attribute called FloodRisk.

8) 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 AttributeValueMappers."

The Complex Filtering Method

This also is a filtering process, but the filtering is all done in a single step - for both zones and elevation - with a TestFilter.

9) Place TestFilter

Place a TestFilter connected to the AttributeRenamer.
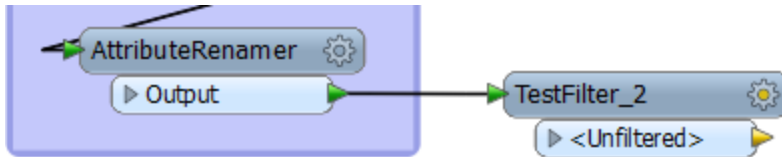
What we want to get here is a separate output port for each flood risk value. So we'll need to incorporate all of the tests into this one transformer.

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.

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 Tester 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.
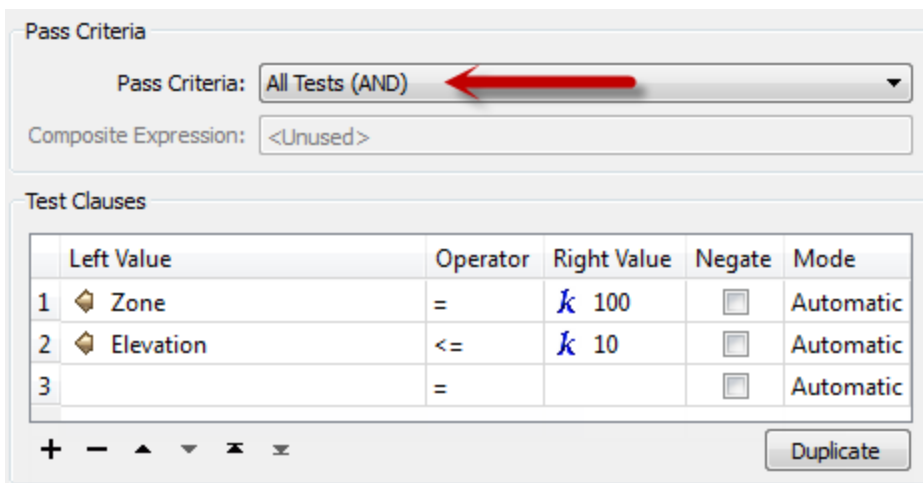
Enter 1 into the Output Port parameter at the foot of the dialog:

Output Port

Output Port: 1

Now click OK to close this part of the dialog.

The main TestFilter dialog now looks like this:

	Test Condition	Output Port
If	@Value(Zone) = 100 AND @Value(Elevation) <= 10	1

OK, 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 3) OR (2 AND 4)

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.

Pass Criteria

Pass Criteria: Composite Test

Composite Expression: (1 AND 3) OR (2 AND 4)

Test Clauses

	Left Value	Operator	Right Value	Negate	Mode
1	Zone	=	k 200	☐	Automatic
2	Zone	=	k 100	☐	Automatic
3	Elevation	<=	k 10	☐	Automatic
4	Elevation	<=	k 25	☐	Automatic

Duplicate

Enter 2 into the Output Port parameter and click OK to close this dialog. The main TestFilter dialog now looks like this:

	Test Condition	Output Port
If	@Value(Zone) = 100 AND @Value(Elevation) <= 10	1
Else If	@Value(Zone) = 200 @Value(Zone) = 100 @Value(Elevation) <= 10 @Value(Elevation) <= 25 Composite Test: (1 AND 3) OR (2 AND 4)	2

Now repeat this step for each of the other flood risk values. There will be three clauses for zone 3, two clauses for zone 4, and back to one clause for zone 5.

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.

	Test Condition	Output Port
If	@Value(Zone) = 100 AND @Value(Elevation) <= 10	1
Else If	@Value(Zone) = 200 @Value(Zone) = 100 @Value(Elevation) <= 10 @Value(Elevation) <= 25 Composite Test: (1 AND 3) OR (2 AND 4)	2
Else If	@Value(Zone) = 300 @Value(Zone) = 200 @Value(Zone) = 100 @Value(Elevation) <= 10 @Value(Elevation) <= 25 @Value(Elevation) <= 60 Composite Test: (1 AND 4) OR (2 AND 5) OR (3 AND 6)	3
Else If	@Value(Zone) = 300 @Value(Zone) = 200 @Value(Elevation) <= 25 @Value(Elevation) <= 60 Composite Test: (1 AND 3) OR (2 AND 4)	4
Else If	@Value(Zone) = 300 AND @Value(Elevation) <= 60	5
Else If		
Else	<All Other Conditions>	<UNFILTER...

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 TestFilter transformer will now look like this:
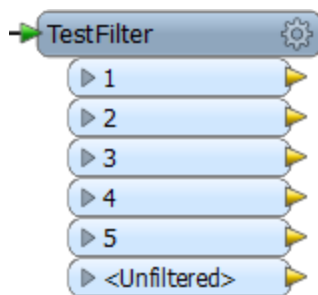
10) Add AttributeCreator

Add an AttributeCreator 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.

11) 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.
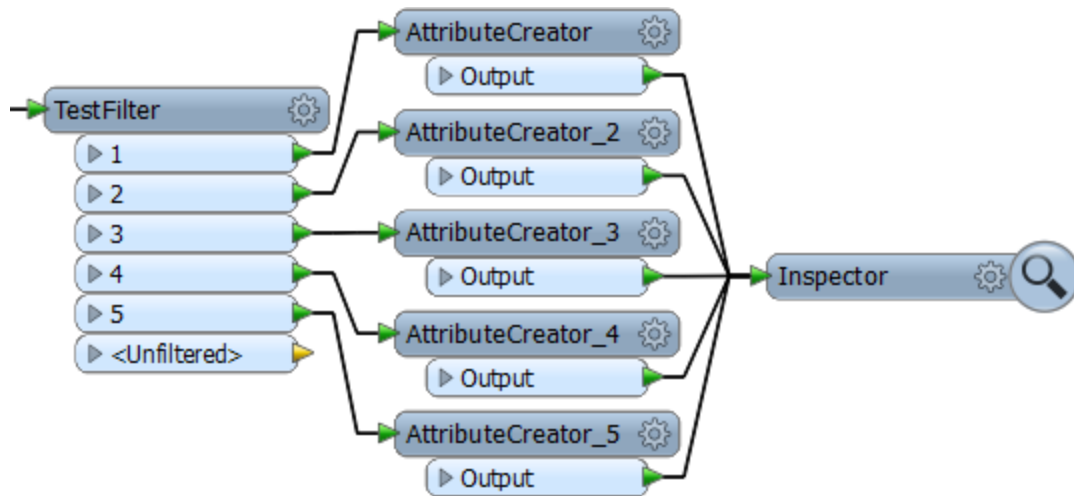
7) Save and Run Workspace

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.

The Conditional Values Method

This is a one-step process involving an AttributeCreator transformer.

12) Place AttributeCreator

Place an AttributeCreator transformer and connect it to the AttributeRenamer.

Open the parameters dialog.

Under AttributeName enter FloodRisk.

Under Value click the drop-down arrow and choose Conditional Value.

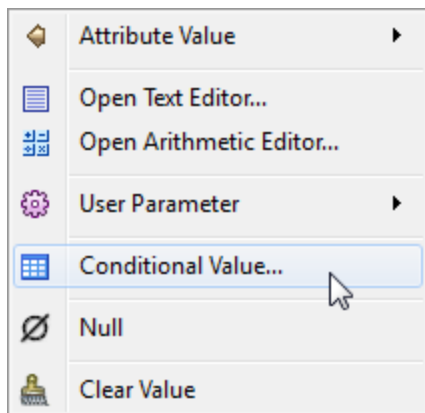See that there are fields for Test Condition and Output Value.

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 Tester 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.
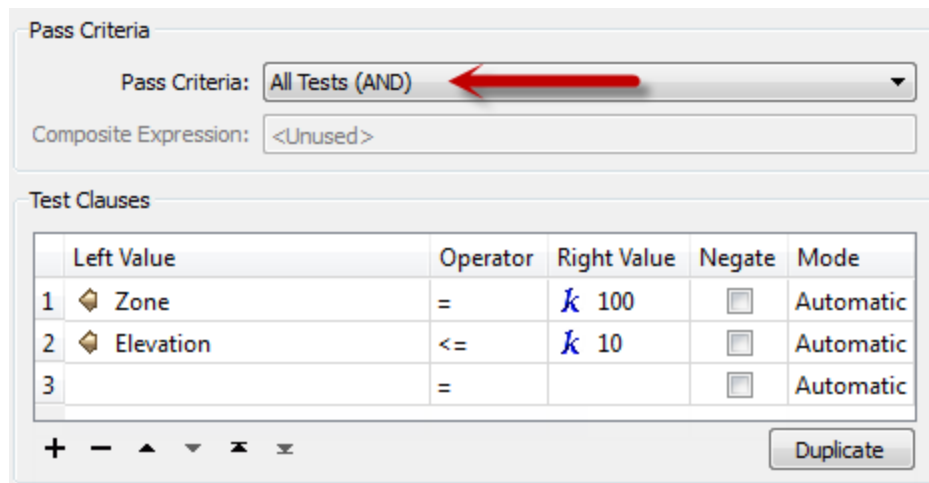
Enter 1 into the Output Value parameter at the foot of the dialog:

Now click OK to close this part of the dialog.

The main Conditional Definition dialog now looks like this:

	Test Condition	Output Value
If	@Value(Zone) = 100 AND @Value(Elevation) <= 10	1

OK, 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 3) OR (2 AND 4)

Of course this will depend on the order you entered the clauses in.

Enter 2 into the Output Value parameter and click OK to close this dialog. The main Conditional Definition dialog now looks like this:

	Test Condition	Output Value
If	@Value(Zone) = 100 AND @Value(Elevation) <= 10	1
Else If	@Value(Zone) = 200 @Value(Zone) = 100 @Value(Elevation) <= 10 @Value(Elevation) <= 25 Composite Test: (1 AND 3) OR (2 AND 4)	2

Now repeat this step for each of the other flood risk values. There will be three clauses for zone 3, two clauses for zone 4, and back to one clause for zone 5.

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.

	Test Condition	Output Value
If	@Value(Zone) = 100 AND @Value(Elevation) <= 10	1
Else If	@Value(Zone) = 200 @Value(Zone) = 100 @Value(Elevation) <= 10 @Value(Elevation) <= 25 Composite Test: (1 AND 3) OR (2 AND 4)	2
Else If	@Value(Zone) = 300 @Value(Zone) = 200 @Value(Zone) = 100 @Value(Elevation) <= 10 @Value(Elevation) <= 25 @Value(Elevation) <= 60 Composite Test: (1 AND 4) OR (2 AND 5) OR (3 AND 6)	3
Else If	@Value(Zone) = 300 @Value(Zone) = 200 @Value(Elevation) <= 25 @Value(Elevation) <= 60 Composite Test: (1 AND 3) OR (2 AND 4)	4
Else If	@Value(Zone) = 300 AND @Value(Elevation) <= 60	5
Else If		
Else	<All Other Conditions>	<UNFILTER...

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:

Attribute Name	Value
◇ FloodRisk	⊞ 6 Possible Values [...] [▼]
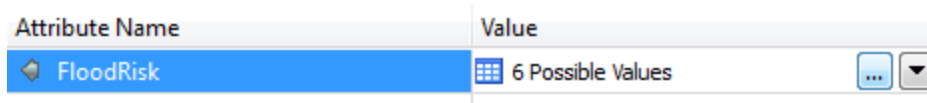
13) Add Inspector

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.

14) 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.

You'll also see unmapped features appear, which doesn't happen with the other methods.

Multiple Feature Attributes

Multiple Feature Attributes

Multiple Feature Scenarios
Multiple Feature Functionality

Multiple Feature Attributes are a way to look into the past features in a translation, and into future ones!

FME Desktop Advanced Training 2015

Multiple Feature Scenarios

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:

XY

+0.0 +3.0

+3.2 +0.0

-3.2 +0.0

+0.0 +3.4

+4.2 +0.0

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 Multiple Feature Attributes in FME.

Multiple Feature Functionality

Multiple feature functionality is activated by checking the box labelled Prior/Subsequent Feature Attribute Retrieval in an AttributeCreator transformer:
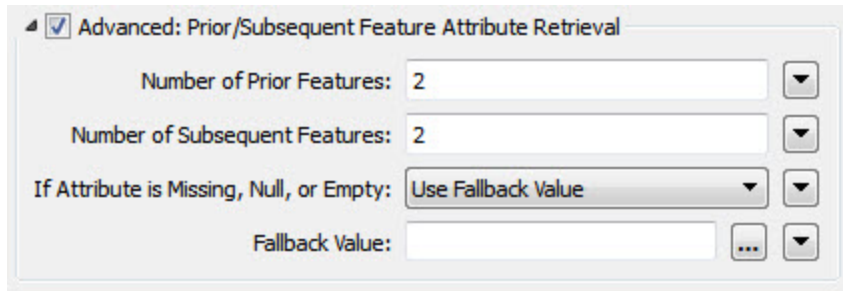
This opens up a section of dialog in which the author can specify how many features preceding the current feature, or how many features that succeed it, should be made available.

There is also an option to specify what should happen if the attributes are missing from a prior/subsequent feature. There is the ability to use a fallback value (which might just be an empty string), to use the value of the closest feature that does contain the attribute, or to use an attribute value instead.

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.

In the editors, the list of feature attributes will have an expandable section for prior and subsequent features:
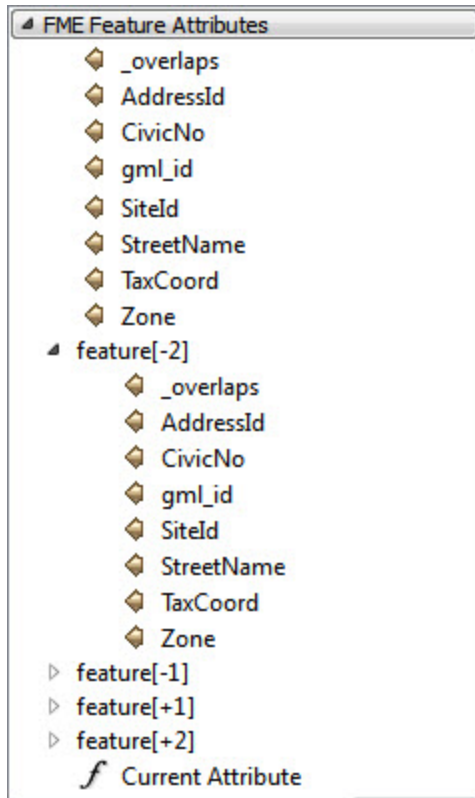
When a feature is expanded then its list of attributes becomes available for use:

Double-clicking an attribute adds it to the expression window:

@Value(feature[-2].AddressId)

So, you can see that prior and subsequent attribute values can be accessed simply by using feature[x].<attribute name> 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, cool to the highest degree; just one that you might not use very often.

Also, be aware that because extra resources are used for storage of the surrounding features, performance will take a (very minor) hit when using these capabilities.

Exercise 5c Multiple Feature Attributes	
Scenario	FME author; City of Interopolis
Data	Interopolis Precipitation (CSV)
Overall Goal	Calculate monthly precipitation totals
Demonstrates	Use of Multiple Feature Attributes
Starting Workspace	None
Finished Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise5c-Complete.fmw*

Here we have a dataset of precipitation for the city of Interopolis. Unfortunately, the numbers are all cumulative and the planning department wants them as absolute figures per month. Rather than reaching into your desk drawer for a calculator, you decide to use FME to do the calculations!

1) Inspect Data

Open the source dataset (C:\FMEData2015\Data\ElevationModel\Precipitation.xlsx)

Notice that it is an Excel spreadsheet that records precipitation, but cumulative rather than as numbers for each individual month.

|	A	B
1	Month	Precipitation
2	Jan	168
3	Feb	273
4	Mar	387
5	Apr	476
6	May	541
7	Jun	595
8	Jul	631
9	Aug	668
10	Sep	719
11	Oct	840
12	Nov	1029
13	Dec	1191

We can subtract the previous month's figure to get the precipitation for any given month, but in FME that requires us to use Multiple Feature Attributes.

2) Open Workbench

Open FME Workbench and generate a workspace from Microsoft Excel to Microsoft Excel, using the above file as the source dataset.

When creating the workspace, check the parameters for the Reader to ensure FME is recognizing the headers at the top of each column.
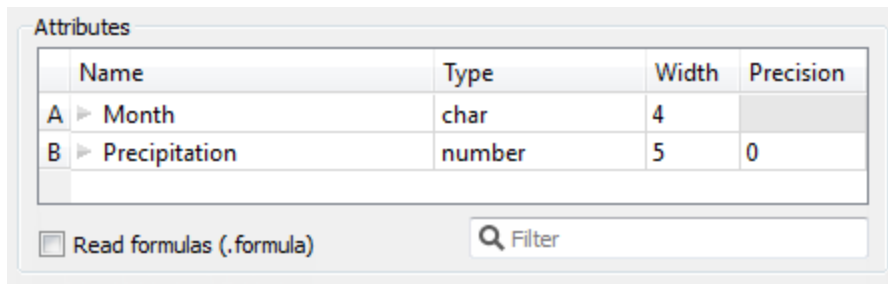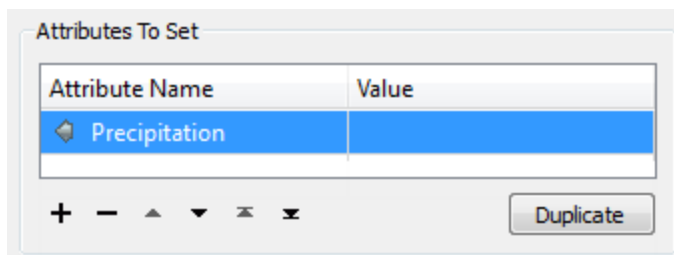

3) Place AttributeCreator

In the newly created workspace, place an AttributeCreator transformer between the Reader and Writer feature types.
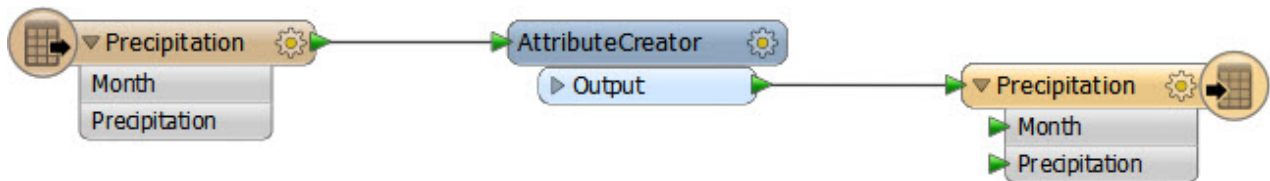
4) Set AttributeCreator Parameters – Part 1

Open the AttributeCreator parameters dialog. In the Attribute Name field select the existing attribute called Precipitation. This will overwrite it with a new value.

Then check the box marked Prior/Subsequent Feature Attribute Retrieval. In the fields provided enter 1 for the Number of Prior Features to be kept.

Next set the 'If Attribute is Missing' field to Use Fallback Value and enter 0 into the Fallback Value field.

Professor Lynn Guistic says…

"The "Attribute is Missing" 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."

5) Set AttributeCreator Parameters – Part 2

Now let's deal with the Value field. Double-click in the field and open the Arithmetic Editor.

Using the menu on the left double-click:

- 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 "Attribute is Missing" 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 AttributeCreator dialog.

6) Save and Run Workspace

Save the workspace and then run it. Inspect the output.

	A	B
1	Month	Precipitation
2	Jan	168
3	Feb	105
4	Mar	282
5	Apr	194
6	May	347
7	Jun	248
8	Jul	383
9	Aug	285
10	Sep	434
11	Oct	406
12	Nov	623
13	Dec	568

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, not the value we've just overwritten it with!
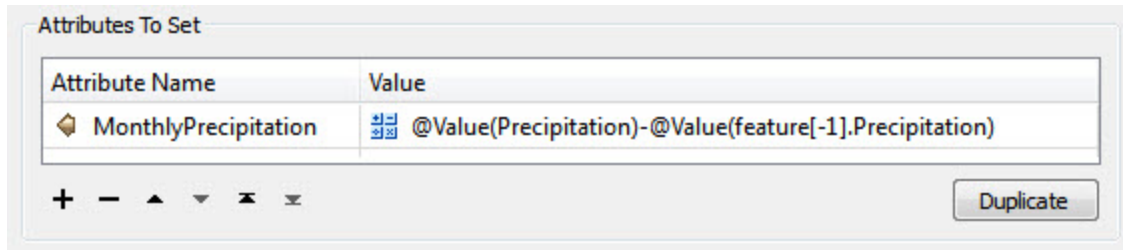
7) Adjust Workspace

OK. Return to the workspace. Edit the Writer schema by renaming the destination attribute Precipitation to MonthlyPrecipitation.

Now return to the AttributeCreator and change the attribute it is creating to MonthlyPrecipitation:

8) 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:

	A	B
1	Month	MonthlyPrecipitation
2	Jan	168
3	Feb	105
4	Mar	114
5	Apr	89
6	May	65
7	Jun	54
8	Jul	36
9	Aug	37
10	Sep	51
11	Oct	121
12	Nov	189
13	Dec	162

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 when we talk about "null". In reality there are many ways to represent nothing:

- An attribute is null

- An attribute exists but is empty

- An attribute doesn't exist (i.e. is missing)

- An 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.

So when we talk about null in FME, 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.

How does FME Represent Null Values?

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 a number of attributes:

```
Feature Type: `Parks_LOGGED'
Attribute(encoded: fme-system)   : `DogPark' has value `N'
Attribute(string)                : `EWStreet' is <null>
Attribute(string)                : `NSStreet' is <null>
Attribute(encoded: fme-system)   : `NeighborhoodName' has value `Kitsilano'
Attribute(string)                : `ParkId' has value `15'
Attribute(string)                : `ParkName' is <null>
Attribute(string)                : `RefParkId' has value `-9999'
Attribute(string)                : `SpecialFeatures' is <null>
```

Similarly, the FME Data Inspector will depict nulls as <null>:

	ParkId	ParkName	NeighborhoodName	EWStreet	NSStreet	DogPark	Washrooms
1	1	<null>	Kitsilano	<null>	<null>	N	<null>

It can also differentiate between states by displaying <missing> or <empty> as well.

Recognizing Null Values

When FME reads data, if the source attributes contain nulls – and the Reader format has been updated to support them – then FME will emit that attribute with a null value.

To check for incoming nulls the Tester transformer has a specific operator to test for null, empty, and missing values:
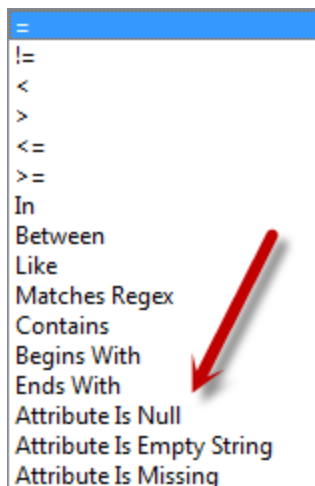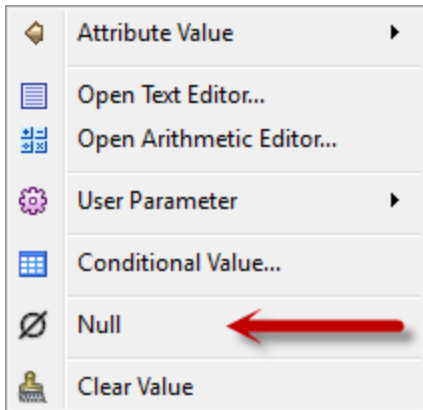

Because the Tester interface is incorporated into many facets of FME (such as the TestFilter transformer) you can test for nulls wherever you find that interface.

Other transformers, such as the Matcher, also allow testing for nulls. In the case of the Matcher there is even a parameter that can be used to decide whether null, empty, and missing values should be treated equally, or as their own unique representation:
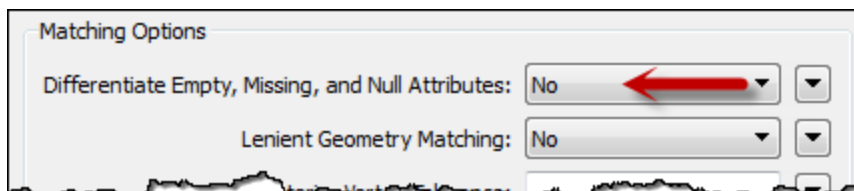
Setting a Null Value

The usual way to set an attribute value is with the AttributeCreator, and this has an option in its drop-down menu to set a value to null:

|		
◈	Attribute Value	▶
▤	Open Text Editor...	
볆	Open Arithmetic Editor...	
⚙	User Parameter	▶
⊞	Conditional Value...	
Ø	Null	←
⚖	Clear Value	

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 – and the Writer has been updated to support them too – 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.

```
─▶ NullAttributeMapp...⚙
     ▷ Output          ▶
```
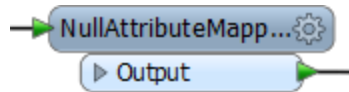
The NullAttributeMapper transformer allows the author to check values for any or all attributes on a feature, and convert them in bulk to or from null.

For example, here the author is checking for all 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."

Exercise 5d Null Attribute Handling	
Scenario	FME author; City of Interopolis
Data	Parks (MapInfo Tab)
Overall Goal	Sort output by park name
Demonstrates	Null attribute handling
Starting Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise5d-Begin.fmw*
Finished Workspace	*C:\FMEData2015\Workspaces\DesktopAdvanced\Exercise5d-Complete.fmw*

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:\FMEData2015\Workspaces\DesktopAdvanced\Exercise5d-Begin.fmw

Inspect the source dataset by right-clicking the source feature type and choosing Inspect.

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.

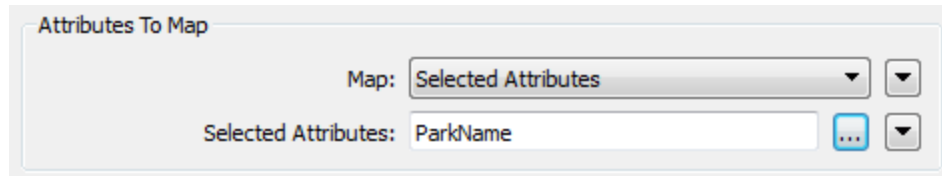|	ParkId	ParkName	NeighborhoodName
1	1	<missing>	Kitsilano
2	2	Rosemary Brow...	Kitsilano
3	3	Tea Swamp Park	Mount Pleasant
4	4	<missing>	Strathcona
5	5	Morton Park	West End

2) Add NullAttributeMapper

Add a NullAttributeMapper transformer prior to the Sorter transformer.

Open the parameters dialog.

Ensure "Map" is set to Selected Attributes, and choose the attribute ParkName.

Underneath that is a section of what to map to.

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 ZZZZ as the new value.
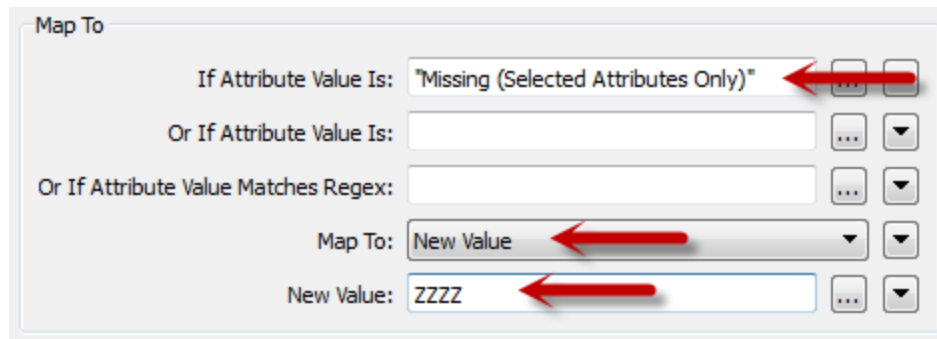
Click OK to close the dialog.

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. Then turn the ZZZZ 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:

Module Review

Module Review

What you should have learned from this module

FME Desktop Advanced Training 2015

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.

Chapter 6 - 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

Our 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 and services philosophy built on the principle of knowledge transfer.

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 Downloads section.

Community Information and Resources

Safe Software actively promotes users of FME to become part of the FME Community.

FMEpedia

FMEpedia is our community web site - a one-stop shop for all community resources, plus tools for browsing documentation and downloads.

FME 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.

FME Community Answers

FME community members post FME related messages and questions and share in answering other users' questions. Top users are known as "heroes". Come and see how they can help with your FME projects!

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.

Course Feedback

Course Feedback

Course Feedback Form

CONNECT. TRANSFORM. AUTOMATE.

FME Desktop Advanced Training 2015

The format of this training course undergoes regular changes prompted by comments and feedback from previous courses.

There's one final Q+A to go – and this time you'll be telling us the answers!

Safe Software Inc. greatly values feedback from training course attendees. This is your chance to tell us what you really think about how well we're meeting your training goals.

The current course structure has been determined by attendee comments and we appreciate your feedback more than ever.

You'll be sent a link to a feedback form after your course.

SAFE SOFTWARE

FME Desktop Training

*1. Please rate the following statements

	Strongly Agree	Agree	Undecided	Disagree	Strongly Disagree
I can apply the course content to my job	○	○	○	○	○
The course length was adequate to cover the content	○	○	○	○	○
There was an appropriate balance between lecture and lab	○	○	○	○	○
The exercises were directly applicable to the material covered	○	○	○	○	○
The graphics and instructions were clear and concise	○	○	○	○	○

*2. Please rate the following

	Excellent	Good	Fair	Poor
Overall course rating	○	○	○	○
Overall Instructor rating	○	○	○	○
Overall learning environment	○	○	○	○
Overall usability of FME	○	○	○	○

Congratulations!

Judge GIS says…

"If you've completed all of the exercises – and I mean ALL – then here's something for you to pass the time.

By combining selected words from this grid, how many FME transformers can you piece together? As a starter, I see both Attribute and Creator, giving AttributeCreator. You can only use each word once – so LineOnLineOverlayer is not allowed!"

CREATOR	EXTRACTOR	LENGTH	POINT	SQL
ATTRIBUTE	RASTER	LINE	LIST	OVERLAYER
CALCULATOR	AREA	REPLACER	REMOVER	ON
STRING	BUILDER	TILER	KML	TEST
BAND	UPDATER	XML	PROPERTIES	FILTER

"Also, just one of the words in this grid cannot be combined with any other to form a transformer name. Can you see which one it is?

Double points awarded if you create a workspace to do the work! Triple points if it uses a transformer from the list!"

Certificates

Certificates

Congratulations!
Thank You

FME Desktop Advanced Training 2015

All FME training course attendees receive a certificate.

Congratulations!

With the presentation of your certificate of achievement, you have now officially completed the FME Desktop 2015 Advanced Training Course.

This certifies that

Mark Ireland

has successfully completed FME Desktop Advanced Training

June 2014	Manitoba, Canada	
Date	Location	Presented by

Thank You

Thank you for attending this FME training course.

All of us piratical types at Safe Software Inc. wish you good luck and fair winds with your FME voyage of discovery.