

# FME® Desktop Basic Training Course



## Table of Contents

Introduction	1.1
About This Document	1.2
Course Overview	1.2.1
Course Resources	1.2.2
Data Translation Basics	1.3
What is FME?	1.3.1
Exercise: Exploring FME	1.3.1.1
FME Desktop Components	1.3.1.2
Introduction to FME Workbench	1.3.2
Window Control in Workbench	1.3.2.1
Creating a Translation	1.3.2.2
The New Workspace	1.3.2.2.1
Running a Workspace	1.3.2.2.2
Exercise: Basic Workspace Creation	1.3.2.2.3
Data Inspection	1.3.3
Data Inspector	1.3.3.1
Inspecting Data	1.3.3.2
Exercise: Basic Data Inspection	1.3.3.3
Display Control	1.3.3.4
Background Maps	1.3.3.5
Feature Caching	1.3.3.6
Visual Preview	1.3.3.7
Exercise: The FME Data Inspector	1.3.3.7.1
Module Review	1.3.4
Quiz	1.3.4.1
Exercise: Tourist Bureau Project	1.3.4.2
Data Transformation	1.4
What is Data Transformation	1.4.1
Structural Transformation	1.4.1.1
Schema Editing	1.4.1.2
Schema Mapping	1.4.1.3
Exercise: Grounds Maintenance Project - Schema Editing	1.4.1.4
Transformation with Transformers	1.4.2
Exercise: Grounds Maintenance Project - Structural Transformation	1.4.2.1
Content Transformation	1.4.3
Transformers in Series	1.4.3.1
Exercise: Grounds Maintenance Project - Calculating Statistics	1.4.3.2
Feature Count Display	1.4.3.3
Transformers in Parallel	1.4.3.4
Exercise: Grounds Maintenance Project - Labelling Features	1.4.3.5
Group-By Processing	1.4.4
Exercise: Grounds Maintenance Project - Neighborhood Averages	1.4.4.1
Coordinate System Transformation	1.4.5
Exercise: Grounds Maintenance Project - Data Reprojection	1.4.5.1
Module Review	1.4.6
Quiz	1.4.6.1

Exercise: Voting Analysis Project	1.4.6.2
Workspace Design	1.5
Workspace Prototyping	1.5.1
Incremental Development	1.5.1.1
Rejected Ports	1.5.1.2
Exercise: Residential Garbage Collection Zones	1.5.1.3
Reading and Writing Workflows	1.5.2
Key Components	1.5.2.1
Multiple Readers and Writers	1.5.2.2
Reader Parameters	1.5.2.3
Writer Parameters	1.5.2.4
Exercise: Residential Garbage Collection Zones	1.5.2.5
Workspace Testing Techniques	1.5.3
Partial Runs	1.5.3.1
Exercise: Residential Garbage Collection Zones	1.5.3.2
Module Review	1.5.4
Quiz	1.5.4.1
Practical Transformer Use	1.6
Locating Transformers	1.6.1
The Transformer Gallery	1.6.1.1
Transformer Searching	1.6.1.2
Most Valuable Transformers	1.6.1.3
Exercise: Transformer Selection Practice	1.6.1.4
Workflow Transformers	1.6.2
FeatureReader and FeatureWriter	1.6.2.1
Read/Write with Integration Transformers	1.6.2.2
Managing Attributes	1.6.3
Creating and Setting Attributes	1.6.3.1
Constructing Attributes	1.6.3.2
Constructing Transformer Parameters	1.6.3.3
Renaming and Copying Attributes	1.6.3.4
Bulk Attribute Renaming	1.6.3.5
Removing Attributes	1.6.3.6
Exercise: Address Open Data Project	1.6.3.7
Conditional Filtering	1.6.4
Tester and TestFilter	1.6.4.1
Other Key Filter Transformers	1.6.4.2
Exercise: Noise Control Laws Project	1.6.4.3
Data Joins	1.6.5
Key-Based Join Transformers	1.6.5.1
Spatially Based Join Transformers	1.6.5.2
Exercise: Crime Mapping Data Request	1.6.5.3
Module Review	1.6.6
Quiz	1.6.6.1
Best Practice	1.7
Debugging	1.7.1
Logging	1.7.1.1
Inspecting Output	1.7.1.2
Feature Counts	1.7.1.3

Checking Key Stages	1.7.1.4
Exercise: Debugging a Workspace	1.7.1.5
Feature Debugging	1.7.1.6
Methodology	1.7.2
Maintenance Methodology	1.7.2.1
Performance Methodology	1.7.2.2
Exercise: Methodology	1.7.2.3
Style	1.7.3
Annotating Workspaces	1.7.3.1
Bookmarks	1.7.3.2
Bookmarks for Design	1.7.3.2.1
Bookmarks for Access	1.7.3.2.2
Bookmarks for Editing	1.7.3.2.3
Bookmarks for Performance	1.7.3.2.4
Object Layout	1.7.3.3
Connection Styles	1.7.3.4
Exercise: The FME Style Guide	1.7.3.5
Module Review	1.7.4
Quiz	1.7.4.1
Exercise: FME Hackathon	1.7.4.2
Course Wrap-Up	1.8
Product Information and Resources	1.8.1
Community Information and Resources	1.8.2
Feedback and Certificates	1.8.3
More Courses	1.8.4
Thank You	1.8.5

# FME Desktop Basic Training Manual

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



The training will introduce basic concepts and terminology, help students become efficient users of FME, and direct you to resources to help apply the product to your own needs.

## Course Structure

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

- Data Translation Basics
- Data Transformation
- Workspace Design
- Practical Transformer Use
- Best Practice

## Current Status

The current status of this manual is: **COMPLETE**: this manual **CAN** be used for training.

This manual applies to **FME2019.0**

The status of each chapter is:

- Chapter 0: Complete content. No exercises
- Chapter 1: Complete content and exercises
- Chapter 2: Complete content and exercises
- Chapter 3: Complete content and exercises
- Chapter 4: Complete content and exercises
- Chapter 5: Complete content and exercises
- Chapter 6: Complete content. No exercises
- Slides: Complete
- FMEData: Complete

**Note:** Even for completed content, Safe Software Inc. assumes no responsibility for any errors in this document or their consequences, and reserves the right to make improvements and changes to this document without notice. See the full licensing agreement for further details.



## About This Document

This manual is the introductory-level training course for FME Desktop.



Look out for the FME Lizard from the City of Interopolis, who will appear from time-to-time to give you advice and dispense FME-related wisdom. In fact, here's the FME lizard now:

### FME Lizard says...

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

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

## Licensing and Warranty

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

1. The origin of the Tutorials and any associated FME® software must not be misrepresented.
2. Redistributions in original or modified form must include Safe Software's copyright notice and any applicable Data Source(s) notices.
3. You may not suggest that any modified version of the Tutorials is endorsed or approved by Safe Software Inc.
4. Redistributions in original or modified form must include a disclaimer similar to that below which: (a) states that the Tutorials are provided "as-is"; (b) disclaims any warranties; and (c) waives any liability claims.

Safe Software Inc. makes no warranty either expressed or implied, including, but not limited to, any implied warranties of merchantability, non-infringement, or fitness for a particular purpose regarding these Tutorials, and makes such Tutorials available solely on an "as-is" basis. In no event shall Safe Software Inc. be liable to anyone for direct, indirect, special, collateral, incidental, or consequential damages in connection with or arising out of the use, modification or distribution of these Tutorials.

This manual describes the functionality and use of the software at the time of publication. The software described herein, and the descriptions themselves, are subject to change without notice.

## Data Sources

### City of Vancouver

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

### Others

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

Digital Elevation Model: GeoBase®

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

Stanley Park GPS Trail: Used with kind permission of [VancouverTrails.com](#).

OpenStreetMap Datasets: © [OpenStreetMap contributors](#).

Contains information licensed under the [Open Government Licence – British Columbia](#)

## Copyright

© 2005–2019 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

**Email:** [train@safe.com](mailto:train@safe.com)

**Web:** [safe.com](http://safe.com)

**GitHub:** [github.com/safesoftware/FMETraining](https://github.com/safesoftware/FMETraining)

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

## Trademarks

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

## Document Information

Document Name: FME Desktop Basic Training Manual 2019.0

## What's New?

A list of changes to this manual and its accompanying datasets can be found on [GitHub](#). The file includes a list of general revisions compared to the previous year's materials. It is designed to help trainers become up-to-speed with new content, and for students to identify which FME functionality is new for the current release.



## Course Overview

This training course provides a framework for a basic understanding of FME. We find users come to master one function but go home with many new FME uses.

The training will introduce basic concepts and terminology, help students become efficient users of FME, and direct you to resources to help apply the product to your own needs.

## Course Structure

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

- Data Translation Basics
- Data Transformation
- Workspace Design
- Practical Transformer Use
- Best Practice

## Course Length

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

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

**Safe Software** offers training in various lengths, but usually either two days or five days. Both courses cover the same content. The two-day course consists of fewer, but longer, days and is more intense. The five-day course consists of more, but shorter, days and allows a little more time to cover the content from a beginner's perspective.

## About the Manual

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

All screenshots in these materials were taken using FME on Windows Server 2016. The fonts used (especially in screenshots of the log window) may be resized or otherwise changed for improved legibility.

## Course Resources

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

### On Your Training Computer

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

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

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

Location	Resource
C:\FMEData2019\Data	Datasets used by the City of Interopolis
C:\FMEData2019\Resources\DesktopBasic	Other resources used in the training
C:\FMEData2019\Workspaces\DesktopBasic	Workspaces used in the student exercises
C:\FMEData2019\Output	The location in which to write exercise output
< documents>\FME\Workspaces	The default location to save FME workspaces

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

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

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

### Course Etiquette

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

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

## Data Translation Basics

At its heart, FME is a data translation tool, and this is usually the first aspect users wish to learn about.

The screenshot shows the FME Integrations Gallery interface. At the top, it says "FME Integrations Gallery" and "Use FME to easily move data between hundreds of apps, web services, databases and file formats.". Below this is a search bar with a magnifying glass icon and the placeholder text "Find your format or a new app...". To the right of the search bar is a dropdown menu labeled "Featured". On the left, there is a sidebar titled "Filter By" with a "Category" section containing checkboxes for various data types: 3D, BIM, Big Data, Business, CAD, Data Warehouse, Database, GIS & Mapping, Imagery & Raster, and LiDAR & Point Clouds. To the right of the sidebar are six cards, each representing a different data format or application:

- Autodesk AutoCAD (DWG) with a red A logo
- Esri ArcGIS Shapefile (SHP) with an Esri logo
- Hadoop Distributed File System (HDFS) with a yellow elephant logo
- Oracle with an Oracle logo
- Salesforce with a blue cloud logo
- Trimble SketchUp with a red cube logo

**Data translation** is the term we at Safe Software use to refer to the conversion of data from one format to another. Have you ever had trouble opening data in an unusual format in your go-to application? Have you ever needed to load spreadsheet data into a database in a systematic way? Have you ever wanted to extract and convert web data in formats like HTML, JSON, and XML with no coding? Then you have come to the right place! FME makes data conversion easy, allowing you to translate between over 400 different formats.

We refer to **translation** rather than **conversion** to emphasize the goal of seamlessly letting data speak in the language of a different format. FME is designed to let you not just convert data from one format to another, but to create output data to your exact specifications.

In this unit, you will learn how to conduct basic data translations yourself using FME.

## What is FME?

Safe Software delivers **FME (Feature Manipulation Engine)**, the data integration platform with the best support for spatial data worldwide.

[fme.ly/whatisfme](http://fme.ly/whatisfme)

## History

Safe Software began in 1993, helping forestry companies exchange maps with the provincial government. It was technically possible to share the maps back then, but only after hours of manual work. Often, an incredible amount of information was lost in the process.

Nobody was happy. Safe Software created FME to address this problem and has been solving data challenges ever since. Today FME is the data integration platform with the best spatial support in the world, and it continues to expand what is possible by adding support for new data formats and workflows with each release.

## Data Integration

"The world's most valuable resource is no longer oil, but data."

-- [The Economist](#), 2017

The creation, manipulation, and analysis of data represent a significant challenge for contemporary organizations. Never before has so much machine-readable data existed, but organizations still struggle to find ways to use this mass of information to aid in decision-making.

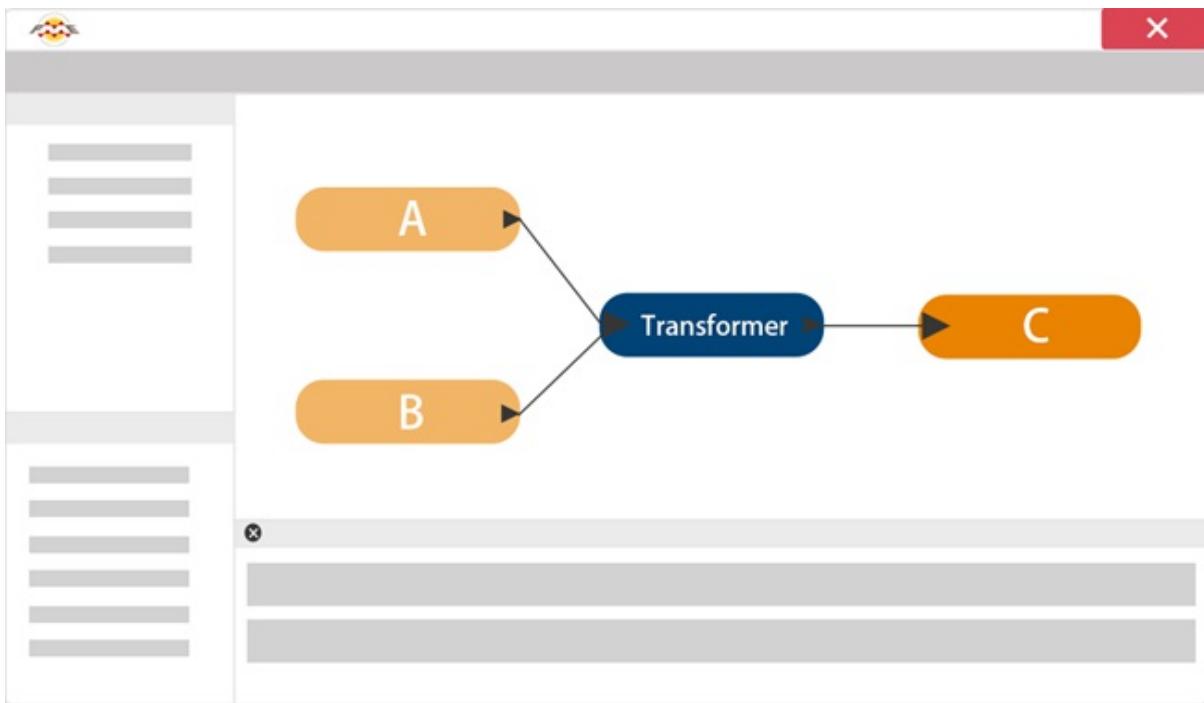
### What is Data Integration?

We define data integration as bringing "together data from disparate sources in a unified view to create a dataset with both valuable and usable information."

Data integration enables the combination and analysis of data across isolated "data silos" where it would normally be difficult to collaborate. Data silos refer to heterogeneous data sources that store data in specific locations. They have long been an issue due to legacy systems and disjointed departments. In the past, it would make sense for departments to select software and methods for data storage with only their needs in mind. Now it is essential to consider cross-functionality. Consolidating data with FME can help bring proprietary, legacy data into new systems that can easily be accessed by any team member.

### Data Integration with FME

FME accomplishes data integration by reading data from multiple sources (in the graphic below, this is shown as sources A and B), using transformer tools to change or restructure the data to fit the users' needs, and writing it to an output location (C):

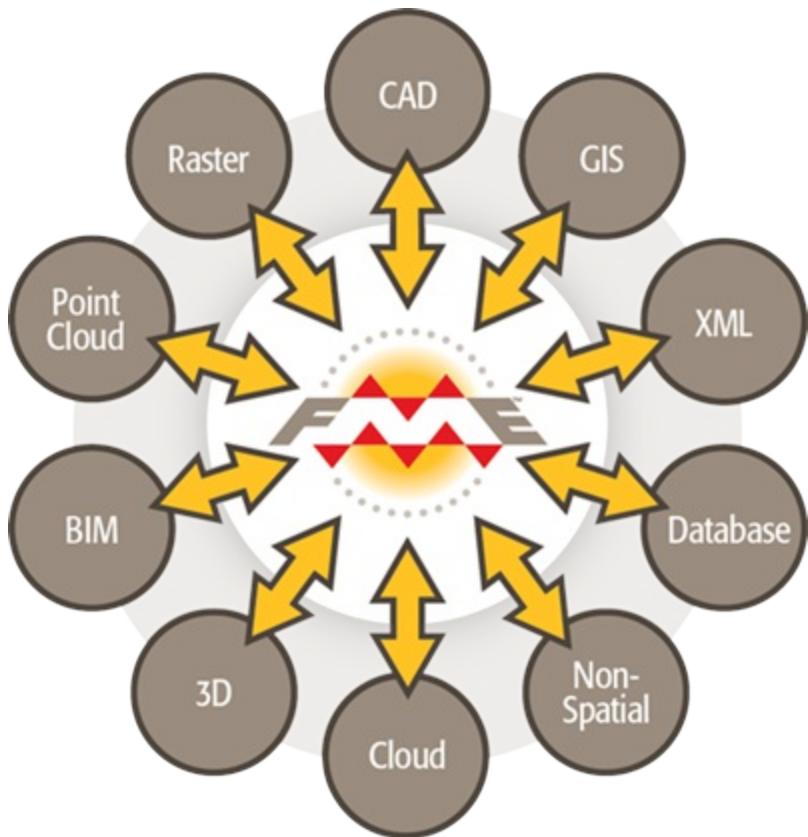


FME's data integration capabilities can be used to convert file formats, transform data, or do both simultaneously. FME can also integrate complex **spatial** data, a category of data most integration tools on the market are unable to utilize. It uses a graphical interface, so no coding is required.

## How FME Works

At the heart of FME is an engine that supports an array of data types, formats, and applications: Excel, CSV, XML, and databases, as well as various types of mapping formats including GIS, CAD, BIM, [and many more](#).

The capability to support so many data types is made possible by a rich data model that handles all possible geometry and attribute types.



## Who Uses FME?

FME has helped thousands of customers worldwide leverage their data so it can be used exactly where, when, and how it's needed. Many of our customers are in the following industries:

- Architecture, Engineering, & Construction
- Energy
- Federal Government
- Local Government
- Telecommunications
- Utilities

Below are a few examples of how people use FME, with links to more details.

### Vancouver International Airport

The Vancouver International Airport (YVR) wanted to help passengers navigate inside the airport. They chose to share indoor mapping data in their own mobile app, on their web site, and via Apple Maps. Their indoor mapping source data was stored as CAD drawings. YVR became one of the first airports to provide indoor wayfinding by using FME to convert their CAD drawings into the [Indoor Mapping Data Format](#).

[Watch a video](#) about this example. [Watch a presentation](#) about this example.



### The Weather Network or Pelmorex Corp.

The Weather Network or Pelmorex Corp. wanted to provide government agencies, private companies, and the public, with real-time and archived lightning strike data. They used FME to convert data from sensors all over Canada to create the [Pelmorex Lightning Detection Network \(PLDN\)](#), which delivers maps, email alerts, and other valuable information.

[Read a blog post](#) about this example.



### Tetrad Sitewise

Tetrad's [Sitewise](#) provides market analysis solutions to help their clients select sites for new business locations using FME. Sitewise can analyze competition, parking, transit accessibility, combining multiple datasets to make let clients make data-driven decisions about where to locate new facilities.

[Watch a webinar](#) about this example.

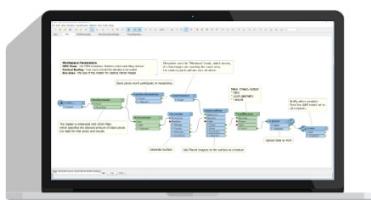


Visit our website for more [customer stories](#).

## Data Integration Platform

This module covers using FME Desktop to create data translations. FME Desktop is one piece of software in the FME data integration platform:

- **FME Desktop** lets you connect and transform data.
  - For example, taking an Excel spreadsheet of business information and addresses and adding it to a MySQL database that is the backend to a citizen data access portal that allows searching for business license information.
- **FME Server** lets you automate workflows on-premises.
  - For example, automatically retrieving data from a GIS layer to match incoming municipal business license applications, as well as sending an email to alert relevant stakeholders.
- **FME Cloud** lets you automate workflows in the cloud.
  - For example, the city government above could access FME Server technology in a fully hosted AWS cloud environment.



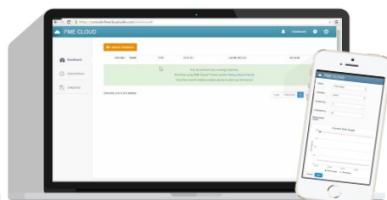
**FME Desktop**

Build & Run Workflows



**FME Server**

Automate Workflows (on-premises)



**FME Cloud**

Automate Workflows (cloud)



## Exercise 1

## Opening and Running a Workspace

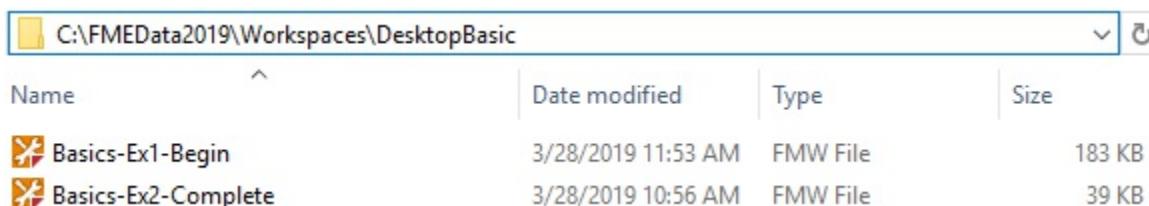
<b>Data</b>	Libraries (Esri Geodatabase) Roads (AutoCAD DWG)
<b>Overall Goal</b>	To open and run an FME workspace to explore what it can do with data
<b>Demonstrates</b>	Opening and running a workspace
<b>Start Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\Basics-Ex1-Begin.fmw
<b>End Workspace</b>	N/A

Rather than trying to explain what FME is and does, let's try it for ourselves! In this exercise you will explore a workflow to integrate and transform data to create an HTML report summarizing information about libraries in Vancouver, BC, Canada. This workflow transforms spatial and attribute data.

### 1) Locate Workspace File

When translations and transformations are defined in FME, they can be saved in a .fmw file.

Using a file explorer, browse to C:\FMEData2019\Workspaces\DesktopBasic\Basics-Ex1-Begin.fmw:

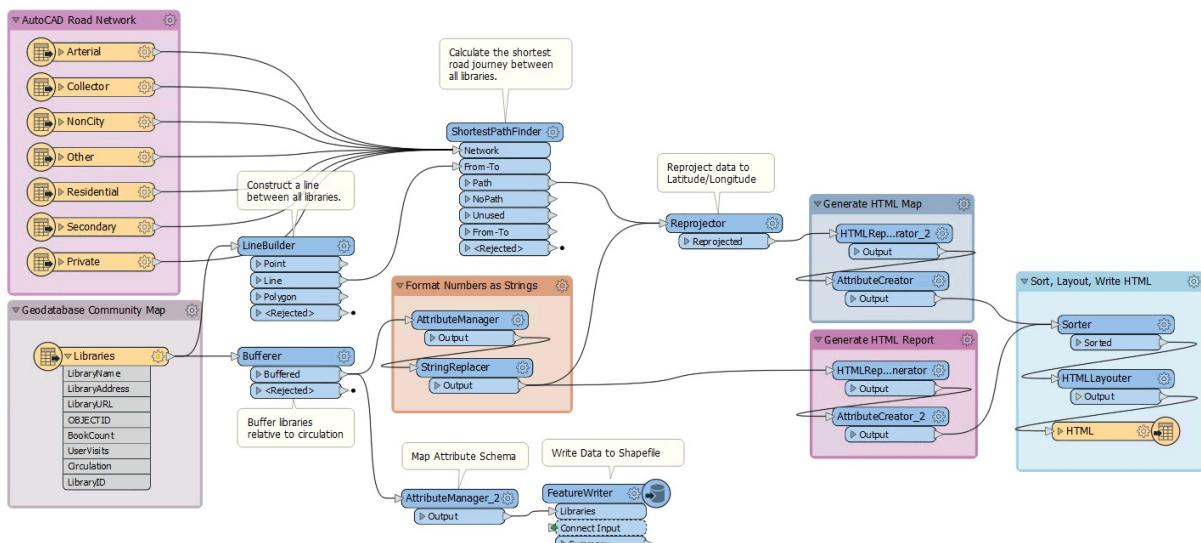


Double-click on the file. It will open an application called FME Workbench.

### 2) Explore FME Workspace

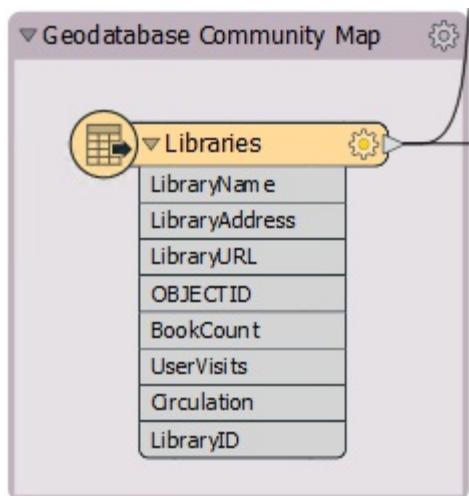
When FME Workbench opens you will see the option of viewing the Workbench Essentials tutorial. You can complete this now if you wish, or view it later under Help > Workbench Essentials. For now, click the **x** to close the window.

The main part of the application will look like this (click to expand):



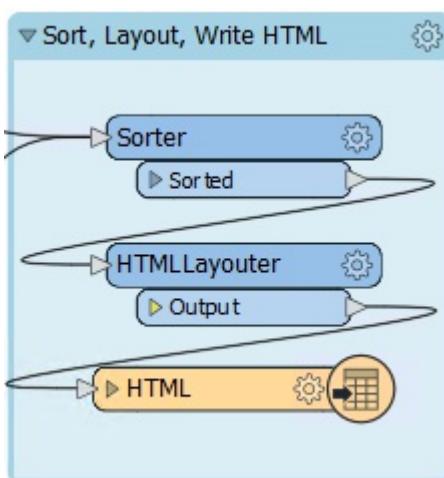
This part we call the canvas. It is where the translation and transformation of data is defined graphically. Although it might look complicated, it does not take much practice with FME to create workflows of this type.

Examine the left-hand side of the canvas:



This area is where we read data, in this case, a table of libraries from an Esri Geodatabase.

Now look at the right-hand side:



This area is where we write data, in this case, a report of the libraries in HTML format.

In between the reader and writer are objects that transform data.

Labels and other annotations show us what the workspace does. It:

- Reads both roads (AutoCAD DWG) and libraries (Esri Geodatabase)
- Calculates the shortest road route taking in all libraries
- Creates circles with diameters relative to each library's book circulation
- Creates an HTML report and an HTML map of the libraries
- Writes the data to HTML and also to Esri Shapefile

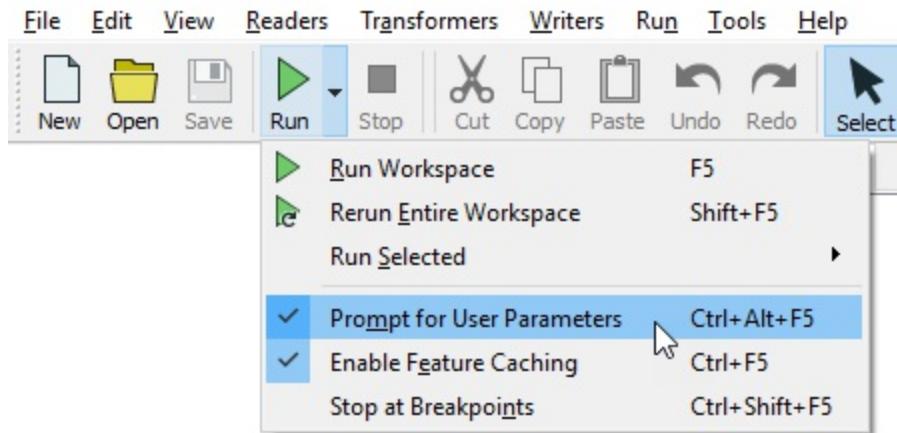
### FME Lizard says...

*Let's make sure we are clear on terminology. The application itself is called FME "Workbench," but the process defined in the canvas window is called a "Workspace." The terms are easily confused.*

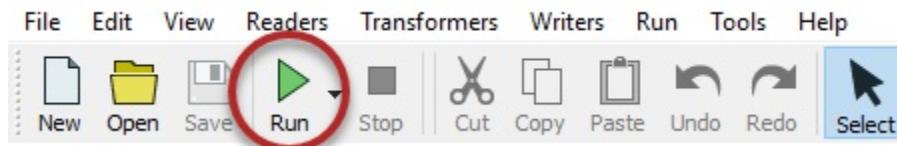
### 3) Run FME Workspace

Let's run this workspace.

Before doing so, we want to control *how* to run the workspace. By default, a feature called **Prompt for User Parameters** is turned on. We don't need this on for this course, so let's turn it off by clicking the dropdown arrow next to the Run button on the toolbar, and clicking Prompt for User Parameters:



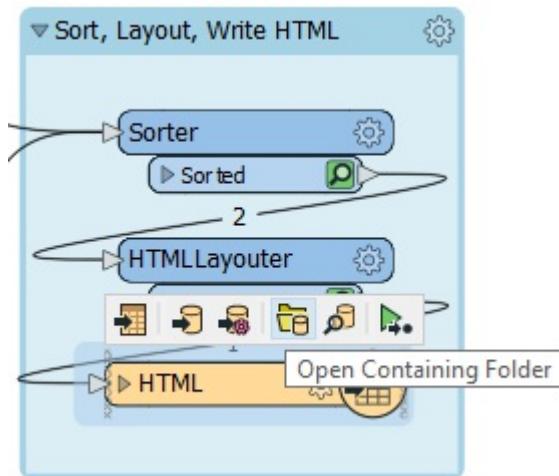
Now we are ready to run the workspace. Click on the green Run button on the Workbench toolbar:



The workspace will now run. As it does, you will see messages pass by in a log window. You may also see numbers appear on the canvas connections and green annotated icons on each object. We'll get to what these are for later!

#### 4) Locate and Examine Output

Once the translation is complete, click on the HTML writer object on the canvas. It is located on the right side of the workspace and is labelled "HTML." Choose the option to Open Containing Folder:

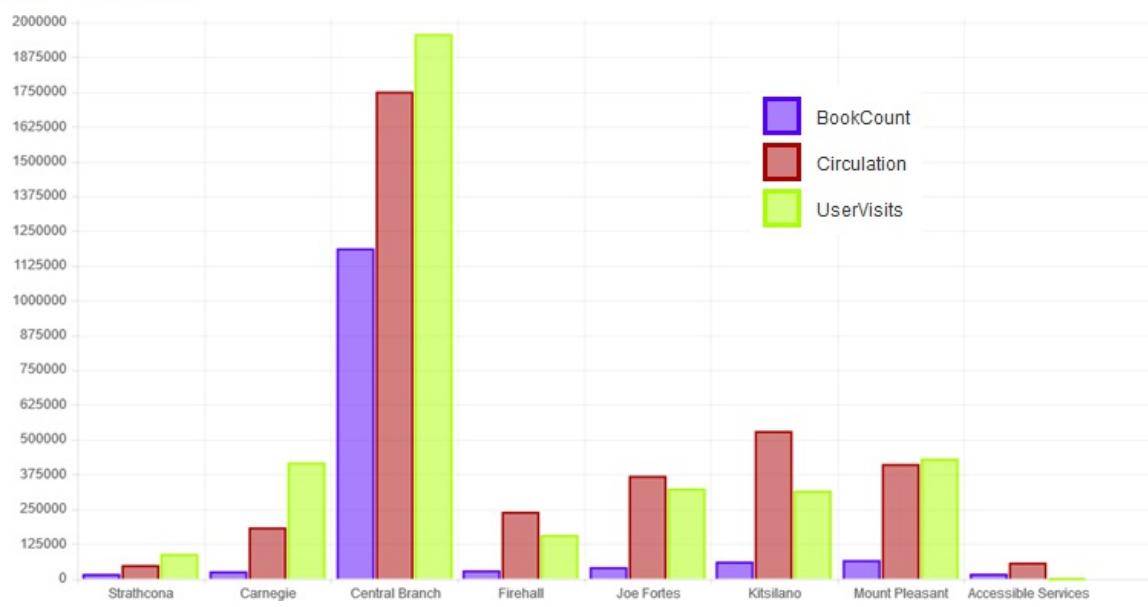


In the Explorer dialog that opens you will find both the HTML output and the Shapefile dataset:

This PC > Local Disk (C:) > FMEData2019 > Output > Training			
Name	Date modified	Type	Size
Libraries	3/28/2019 3:01 PM	DBF File	2 KB
Libraries.prj	3/28/2019 3:01 PM	PRJ File	1 KB
Libraries	3/28/2019 3:01 PM	SHP File	10 KB
Libraries.shx	3/28/2019 3:01 PM	SHX File	1 KB
LibraryReport	3/28/2019 3:01 PM	Chrome HTML Do...	136 KB

Open the output file created by FME with a web browser such as Firefox or Chrome (double-clicking it should open it in your default browser). You will see a table of libraries, a graph of library statistics, and an interactive map showing where the libraries are located. All this has been generated by FME from the incoming Geodatabase points and attributes:

### Library Statistics



### FME Lizard says...

*This small demonstration illustrates the power of FME. This workspace read data from multiple spatial datasets and wrote it out to datasets in both spatial and "tabular" formats. In between it carried out a series of transformations and spatial analyses, buffering and reprojecting the data, and creating added value and information.*

### CONGRATULATIONS

By completing this exercise you have learned how to:

- Open an FME workspace
- Run an FME workspace
- Locate the output from an FME workspace



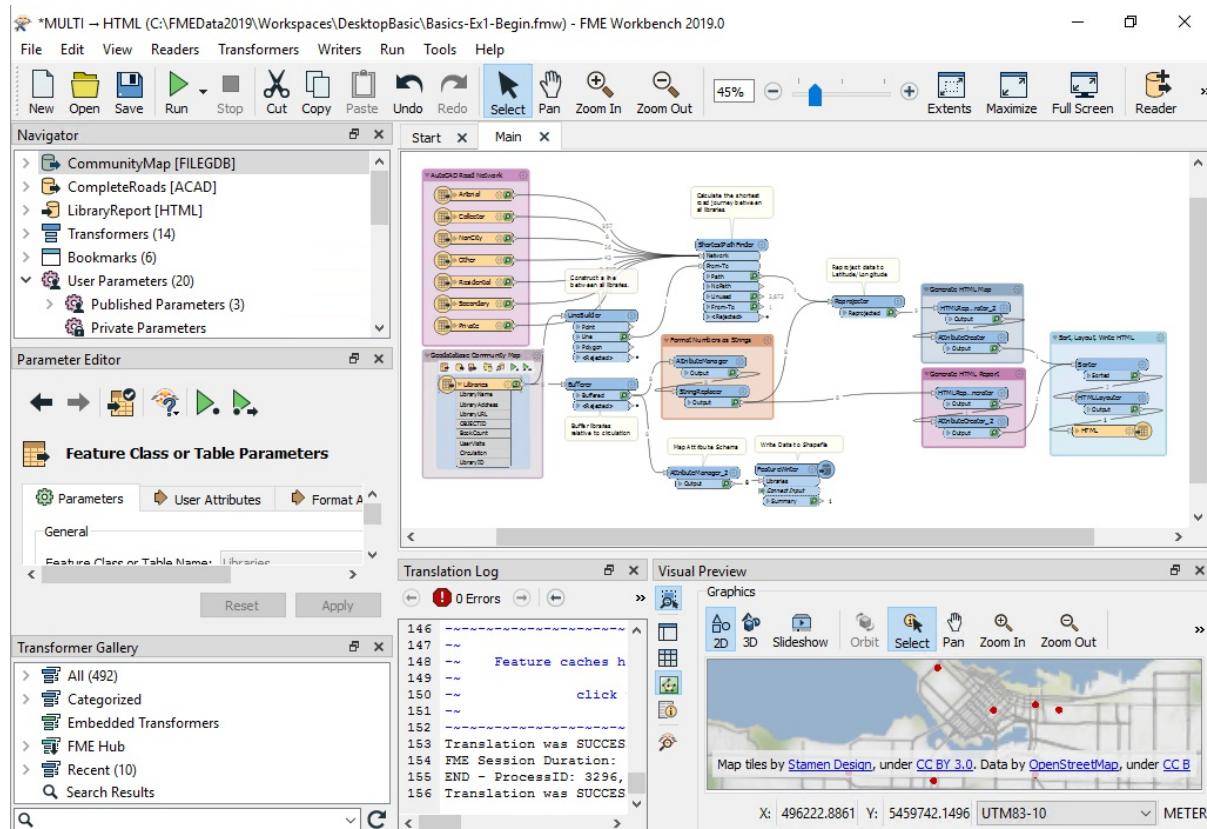
## FME Desktop

This course is about FME Desktop. FME Desktop is for data translations and transformations at the desktop level (as opposed to FME Server, which is an enterprise-level, web-based product).

FME Desktop consists of a number of different tools and applications. The two key applications are **FME Workbench** and the **FME Data Inspector**.

## FME Workbench

FME Workbench is the primary tool for defining data translations and data transformations. It has an intuitive point-and-click graphic interface to enable translations to be graphically described as a flow of data.

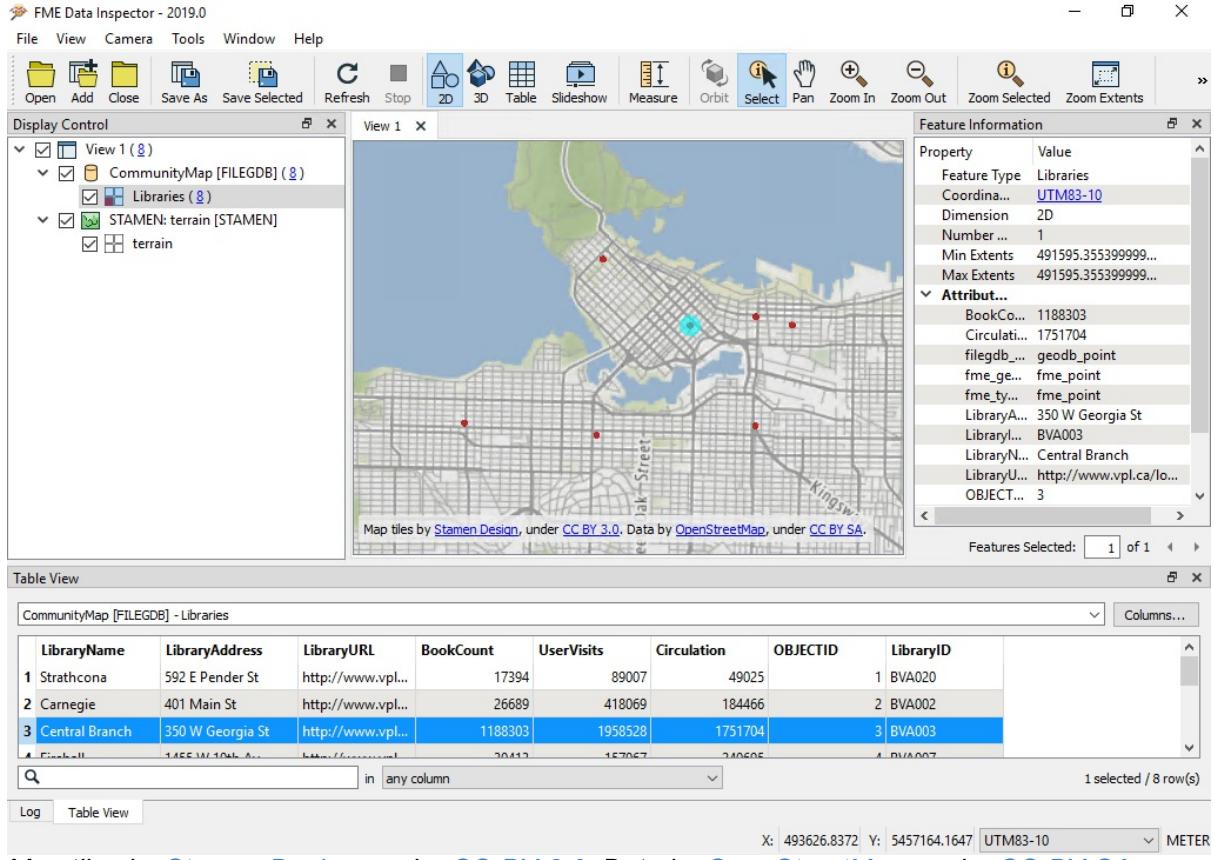


Map tiles by [Stamen Design](#), under CC-BY-3.0. Data by [OpenStreetMap](#), under CC-BY-SA.

Workbench is not a standalone tool. It is fully integrated to interact with other FME Desktop applications such as the FME Data Inspector.

## FME Data Inspector

The FME Data Inspector is a tool for viewing data in any of the FME supported formats. It is used primarily for previewing data before translation or reviewing it after translation.



Map tiles by [Stamen Design](#), under [CC-BY-3.0](#). Data by [OpenStreetMap](#), under [CC-BY-SA](#).

## FME Utilities

Besides Workbench and the Data Inspector, there are several other FME utilities.

- **FME Help**
  - A tool for browsing through the various help documents for FME.
- **FME Quick Translator**
  - A precursor to FME Workbench that is used only for quick translations requiring no data transformation, or for running a workspace without opening it for editing.
- **FME Integration Console**
  - A tool for embedding FME functionality into other applications, particularly spatial tools such as ArcGIS, AutoCAD, Geomedia, and MapInfo.
- **FME Licensing Assistant**
  - An application for managing FME licensing.

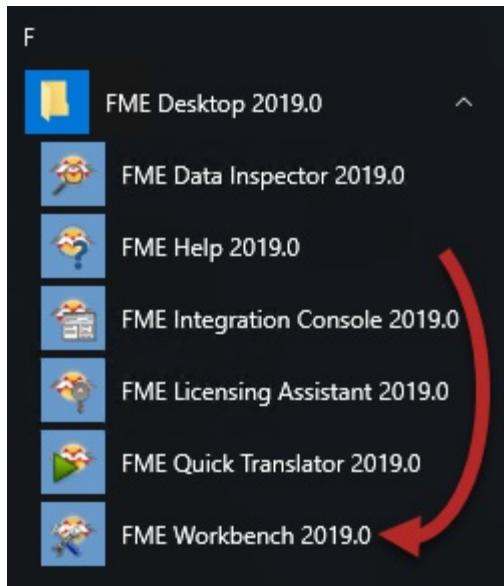
## Other FME Desktop Components

Additional components are also included as part of FME Desktop.

- **FME Command Line Engine**
  - The FME Command Line Engine enables translations to be initiated at the command line level.
- **FME Plug-In SDK**
  - The FME Plug-In SDK allows developers to add formats and functionality to the FME core.

## Introduction to FME Workbench

Let's take a closer look at FME Workbench. If you are opening a workspace directly, you can start Workbench through the Windows start menu:



### FME Lizard says...

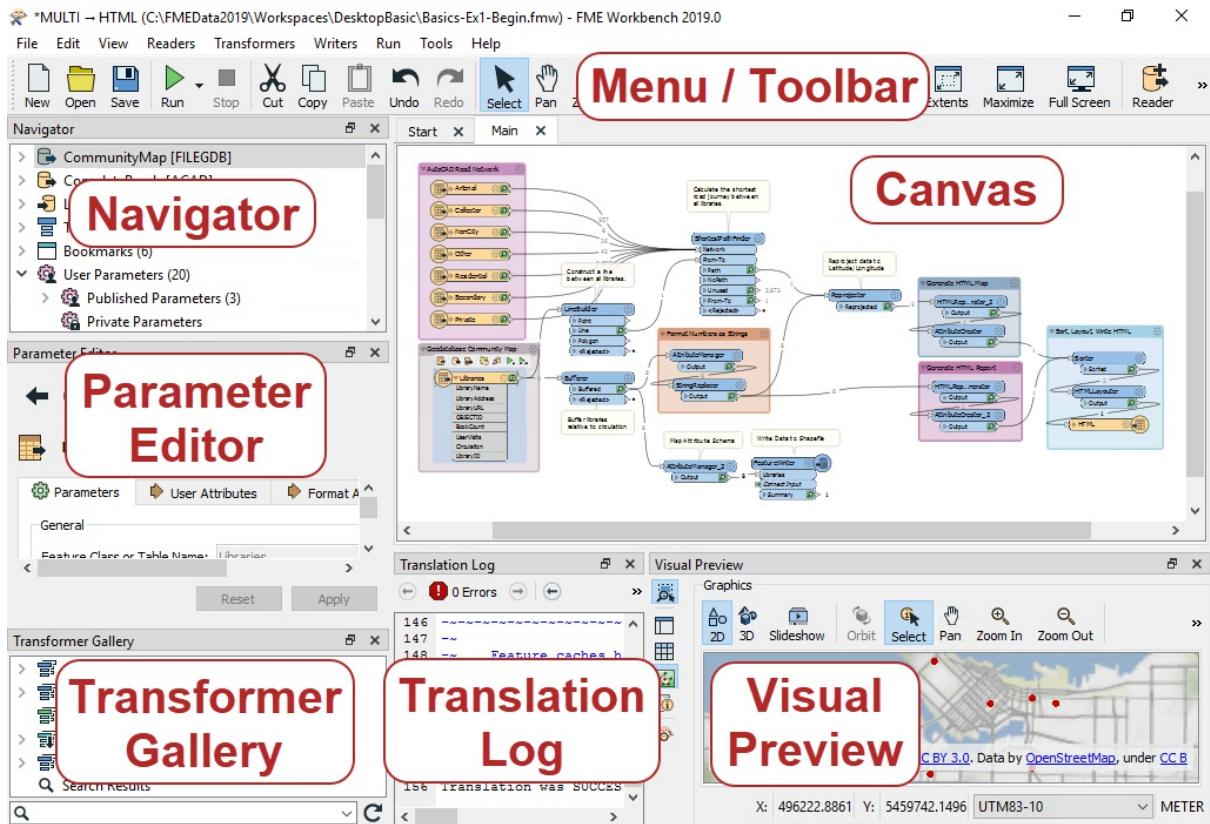
Hello, I'm the [FME Lizard](#)! I'm an FME expert and will pop in occasionally to provide some helpful tips for using FME. Here's my first one!

Feel free to follow along in Workbench during these informational sections if you wish. Alternatively, you can just complete the hands-on exercises in each unit, which will illustrate the concepts shown between exercises.



## Major Components of FME Workbench

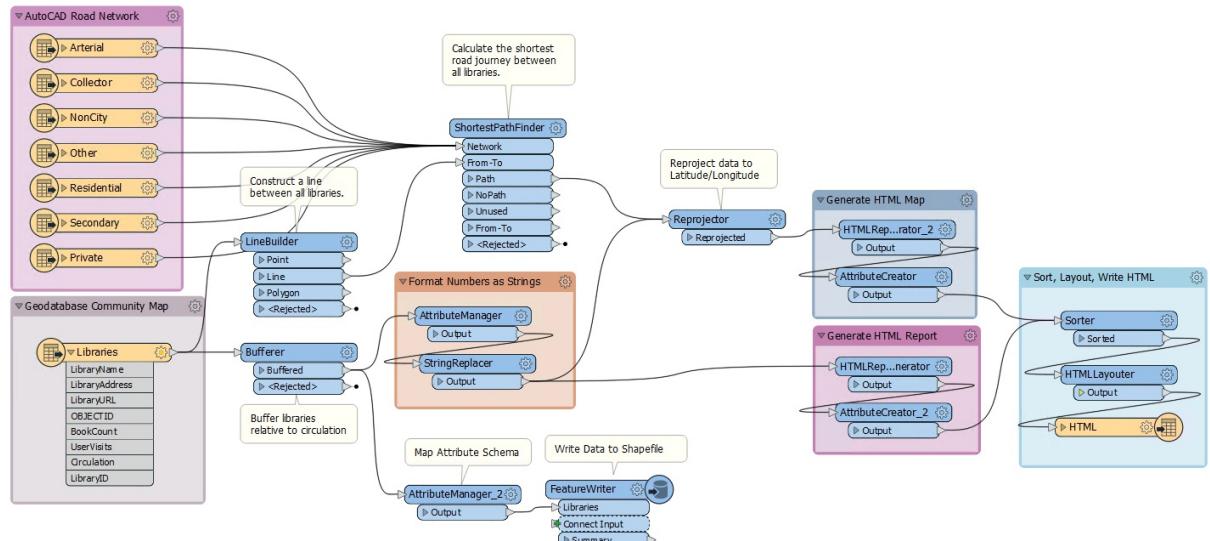
The FME Workbench user interface has multiple components:



Map tiles by [Stamen Design](#), under [CC-BY-3.0](#). Data by [OpenStreetMap](#), under [CC-BY-SA](#).

## Canvas

The FME Workbench canvas is where you define the translation. It is the primary window within Workbench:



By default the workspace reads from left to right; data source on the left, transformation tools in the center, and data output on the right. Connections between each item represent the flow of data and may branch in different directions, merge together, or both.

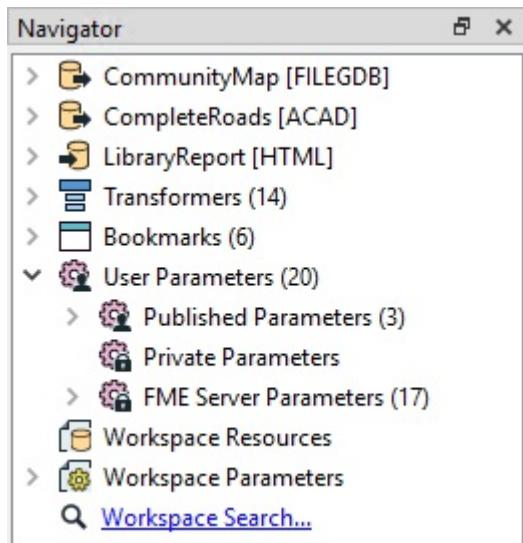
## Menu/Toolbar

The menu bar and toolbar contain many tools: for example, tools for navigating around the Workbench canvas, controlling administrative tasks, and adding or removing readers/writers:



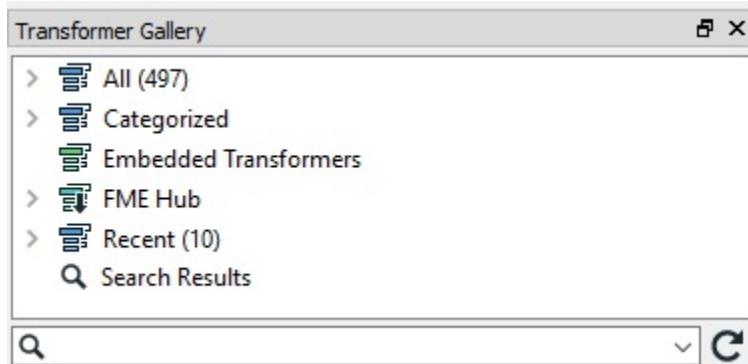
## **Navigator**

The Navigator window is a structured list of parameters that represent and control all of the components of a translation:



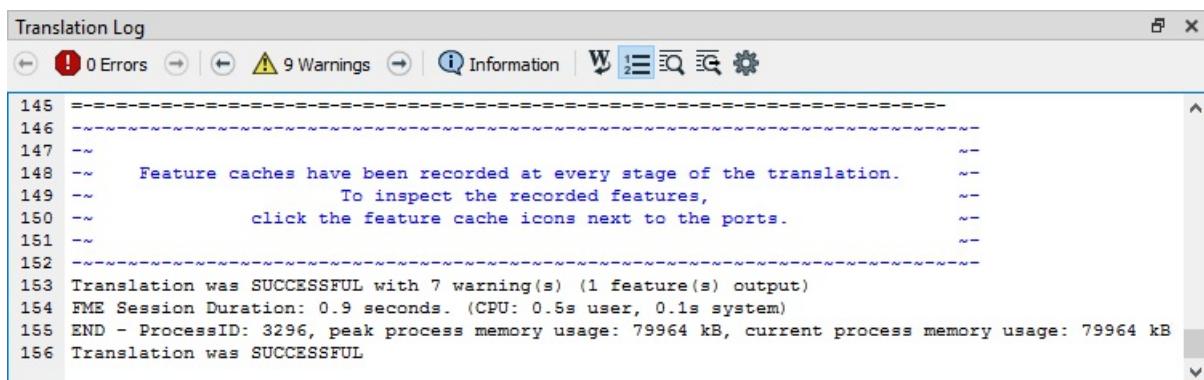
## **Transformer Gallery**

The Transformer Gallery is a tool for the location and selection of FME transformation tools. The number of transformers varies depending on the version of FME and any optional custom transformers installed:



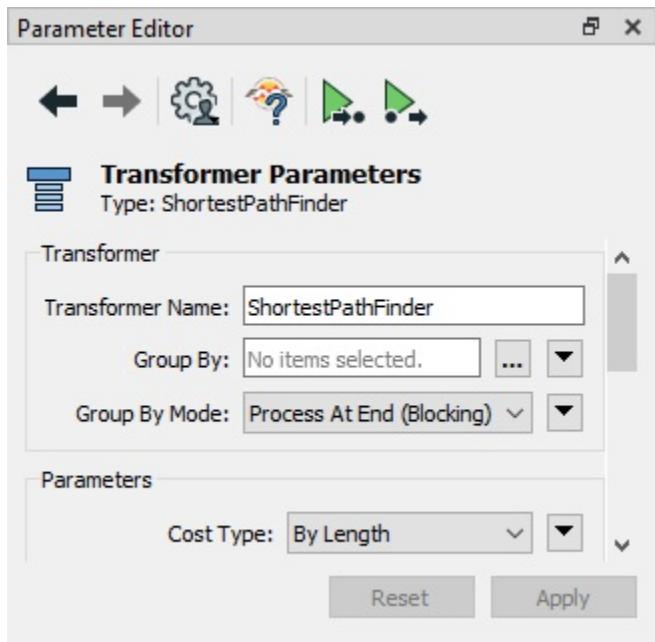
## **Translation Log**

The Translation Log reports on translations and other actions. Information includes any warning or error messages, translation status, length of translation, and the number of features processed:



## **Parameter Editor Window**

The Parameter Editor window is for editing parameters for objects on the canvas window:



...although each canvas object also has its own parameter window as well.

## Visual Preview

Visual Preview is an embedded version of FME Data Inspector that displays features in a table or on a map. This window lets you track how your data is changing as you build your translation. Many of the features available in the stand-alone Data Inspector application are available in Visual Preview. We'll discuss it more in a later section.

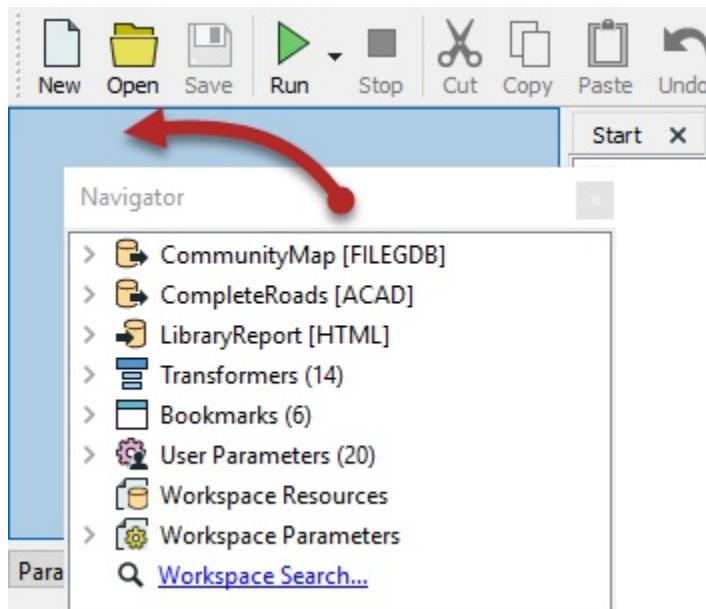
The Visual Preview window contains two main panes: 'Table' and 'Graphics'. The 'Table' pane shows a list of local areas with names like Arbutus-Ridge, Shaughnessy, Riley Park, etc. The 'Graphics' pane shows a map of Vancouver with a red boundary around a specific area, identified as 'B' on the map. A status bar at the bottom provides coordinates (X: -123.1103, Y: 49.2698) and a coordinate system (LL84 DEGREE).

	Name	Description
1	Arbutus-Ridge	<missing>
2	Shaughnessy	<missing>
3	Riley Park	<missing>
4	South Cambie	<missing>
5	Kensington-Ce...	<missing>
6	Renfrew-Collin...	<missing>
7	Oakridge	<missing>

Map tiles by [Stamen Design](#), under [CC-BY-3.0](#). Data by [OpenStreetMap](#), under [CC-BY-SA](#).

## Window Control

All windows in Workbench can be detached from their default position and deposited in a custom location. To do this simply click on the frame of the window and drag it into a new position.



If a window is dropped on top of an existing window, then the two will become **tabbed**.

If a window is dropped beside an existing window (or between two existing windows), then they will become **stacked**.

In the above screenshot, the user is dropping the Navigator into position on the left-hand side of Workbench, stacked on top of the Parameter Editor.

### FME Lizard says...

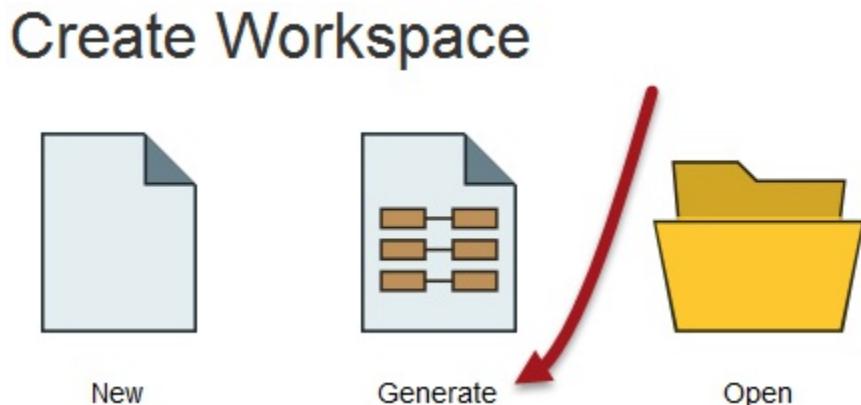
*No need to scale back on canvas space. Press F11 to move lesser-used windows to one side and expand the canvas window to full size! The windows can also be undocked and moved to another monitor for even more space!*

## Creating a Translation

Workbench's intuitive interface makes it easy to set up and run a simple format-to-format ('quick') translation.

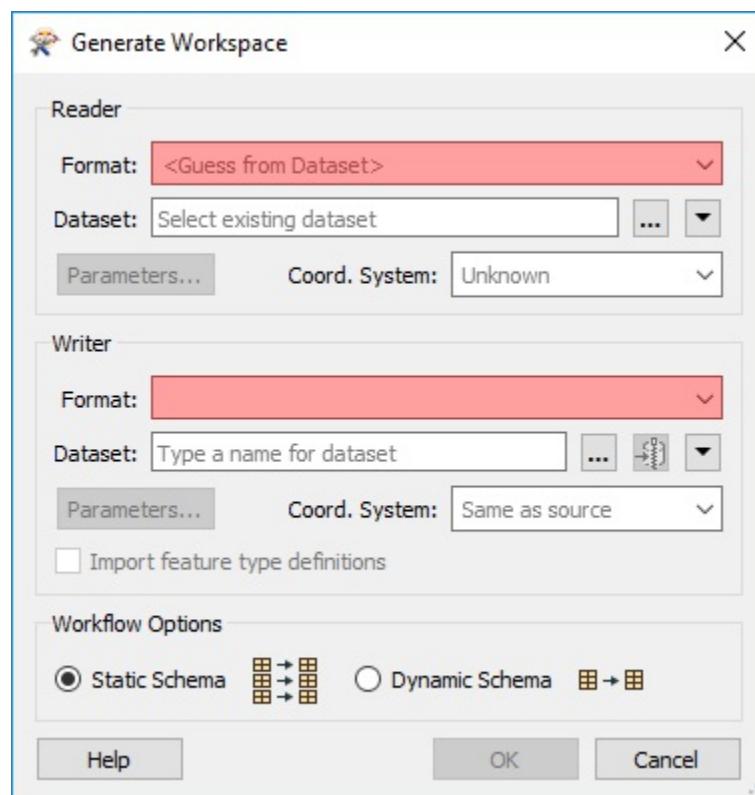
### The Start Tab

The Start Tab in FME Workbench includes different ways to create or open a workspace. The simplest method is Generate Workspace:



### Generate Workspace Dialog

The Generate Workspace dialog condenses all the choices to be made into a single dialog box. It has fields for defining the format and location of both the data to be read, and the data to be written.



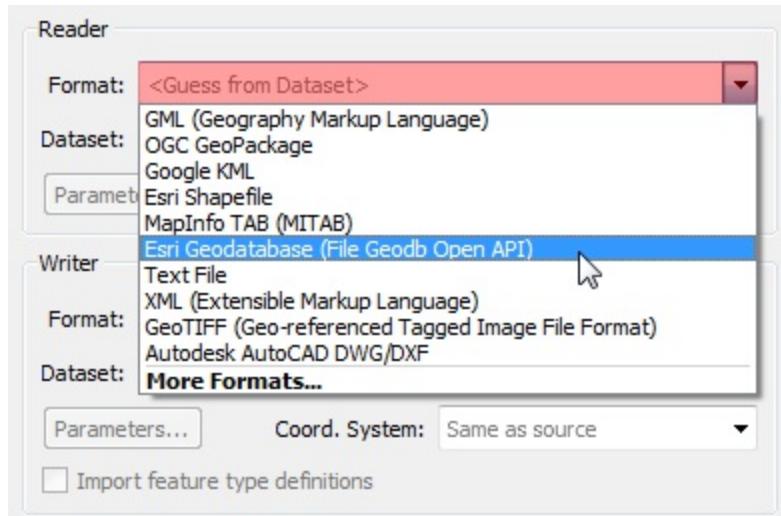
Red coloring in an FME dialog indicates mandatory fields. Users must enter data in these fields to continue. In most dialogs, the OK button is deactivated until the mandatory fields are complete.

### Format and Dataset Selection

A key requirement is the format of the source data. All format selection fields in FME are both a pull-down menu and a text entry field.

The text entry field allows you to type a format name directly. It has an 'intelli-complete' function that selects close matches as you type.

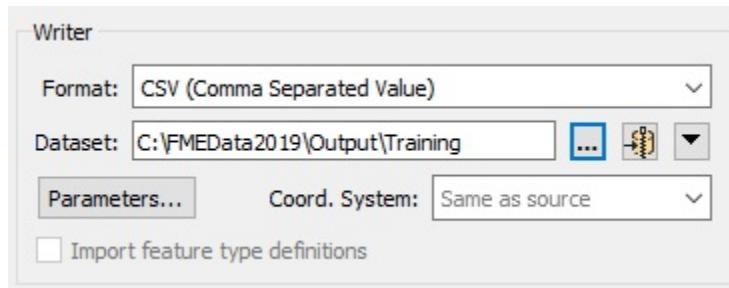
The drop-down list shows some of the most commonly used formats, so many favourite formats are instantly available:



Click 'More Formats' and a table opens showing all of the formats supported by FME in the **Reader and Writer Gallery**.

The source dataset is another key requirement. Dataset selection fields are a text entry field, but with a browse button to open an explorer-like dialog for file selection.

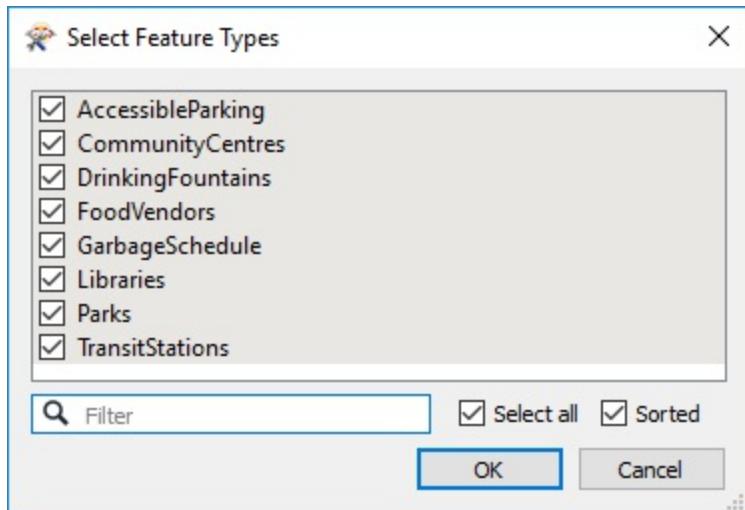
Similarly, the Writer format and dataset are defined in this dialog:



## Feature Types Dialog

Clicking OK on the Generate Workspace dialog causes FME to generate the defined workspace. However, whenever a source dataset contains multiple **feature types**, the user is first prompted to select which are to be translated.

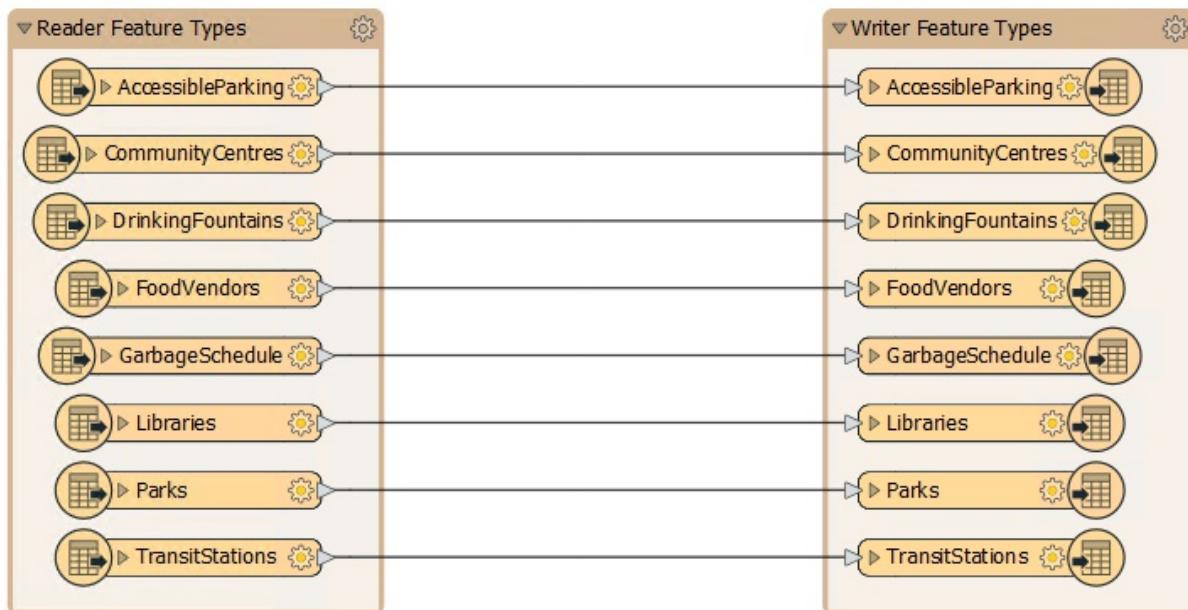
This is achieved through the Select Feature Types dialog. Feature type is the FME term that describes a subset of records. Common alternatives for this term are *layer*, *table*, *sheet*, *feature class*, and *object class*. For example, each sheet in an Excel workbook, table in a database, or layer in a spatial data file is defined by a feature type in FME. Only feature types selected in the Select Feature Types dialog will be added to the workspace:



Here, for example, is a Select Feature Types dialog where the user has chosen to include all available layers in the workspace.

## The New Workspace

A new workspace reads from left to right, from the source (Reader) feature types, through to the destination (Writer) feature types. Arrows denote the direction of data flow:



In the above screenshot, eight feature types are being read from one format and written to another.

### FME Lizard says...

*In most cases FME uses the terms 'Reader' and 'Writer' instead of 'Source' and 'Destination.' So a Reader reads datasets and a Writer writes datasets, in terms analogous to source/destination and input/output.*

## Saving the Workspace

Workspaces can be saved to a file so that they can be reused at a later date. The save button on the toolbar is one way to do this:



There are also menu options to do the same thing, in this case, File > Save (shortcut = **Ctrl+S**) or File > Save As. The default file extension is .fmw.

### FME Lizard says...

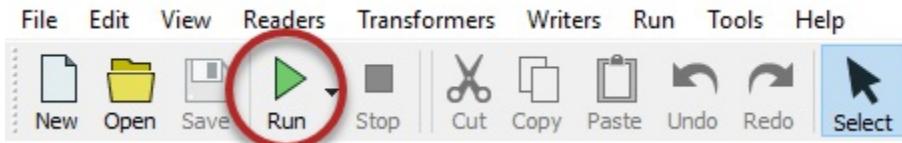
*There are several intuitive, but less obvious, features of the FME Workbench interface.*

*For example, select File > Open Recent on the menu bar to reveal a list of previously used*

*workspaces. This list can show up to a huge 15 entries.*

## Running a Workspace

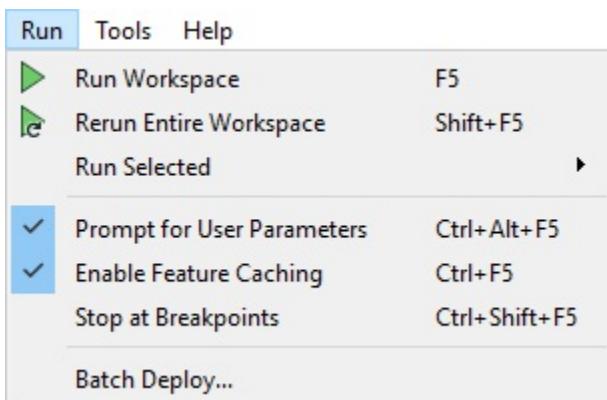
The green arrow (or 'play' button) on the Workbench toolbar starts a translation:



Alternatively, look under Run on the menu bar:



The same toolbar options appear on both the menu bar and toolbar. Notice the shortcut option F5 can be used instead:



### FME Lizard says...

*The action of the Run button can be modified by the dropdown on the button, including the ability to only run sections of your workspace (Run Selected), prompt the user for input (Prompt for User Parameters), the ability to cache intermediate data (Enable Feature Caching) and the ability to run with breakpoints for debugging (Stop at Breakpoints). We'll cover some of them later in the training.*

## Workspace Results

After running a workspace, related information and statistics are found in the Translation Log, which is displayed in the Workbench log window.

The translation log reveals whether the translation succeeded or failed, how many features were read from the source and written to the destination, and how long it took to perform the translation.

```

=====
          Features Read Summary
=====
AccessibleParking                                42
CommunityCentres                                 10
DrinkingFountains                               113
FoodVendors                                     91
GarbageSchedule                                  6
Libraries                                       8
Parks                                            69
TransitStations                                 11
=====
Total Features Read                            350
=====

=====
          Features Written Summary
=====
AccessibleParking                                42
CommunityCentres                                 10
DrinkingFountains                               113
FoodVendors                                     91
GarbageSchedule                                  6
Libraries                                       8
Parks                                            69
TransitStations                                 11
=====
Total Features Written                         350
=====

Translation was SUCCESSFUL with 0 warning(s) (8 feature(s) output)
FME Session Duration: 2.1 seconds. (CPU: 1.1s user, 0.5s system)
END - ProcessID: 4084, peak process memory usage: 131024 kB
Translation was SUCCESSFUL

```

In this example, the log file reveals that 350 features were read (from a spatial data format called Esri Geodatabase) and written out (to a spatial form of XML, called GML).

The overall process was a success, with zero warnings. The elapsed time for the translation was 2.1 seconds.

---

### FME Lizard says...

*In FME a "feature" is an individual piece of data, similar to a record in a database or a row in a spreadsheet.*

## Exercise 2

## Basic Workspace Creation

<b>Data</b>	Zoning Data (MapInfo TAB)
<b>Overall Goal</b>	Create a workspace to translate zoning data in MapInfo TAB format to GeoJSON (Geographic JavaScript Object Notation)
<b>Demonstrates</b>	Basic workspace creation with FME Workbench
<b>Start Workspace</b>	None
<b>End Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\Basics-Ex2-Complete.fmw

Congratulations! You have just landed a job as a technical analyst in the GIS department of your local city. Your old friend, the FME Lizard, gave you a reference, so don't let them down!

On your first day, you've been asked to do a simple file format translation.

We've outlined all of the actions you need to take, though FME's interface is so intuitive you should be able to carry out the exercise without the need for these step-by-step instructions.

### 1) Start FME Workbench

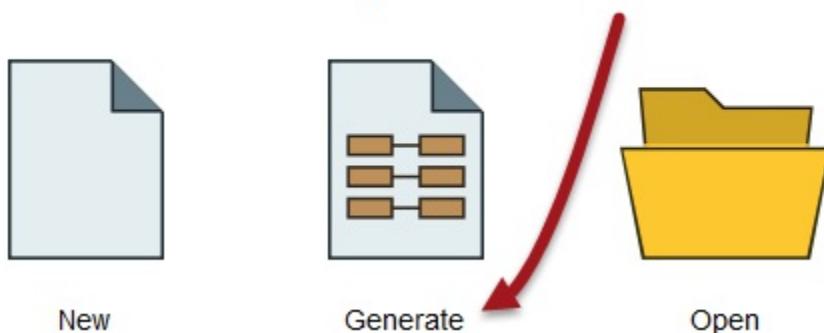
If it isn't open already, start FME Workbench by selecting it from the Windows start menu. You'll find it under Start > FME Desktop 2019.0 > FME Workbench 2019.0.

If Workbench is already open, click on the Start tab above the main canvas.

### 2) Select Generate Workspace

In the Create Workspace part of the Start page select the option to Generate (Workspace). Alternatively, you can use the shortcut Ctrl+G:

## Create Workspace

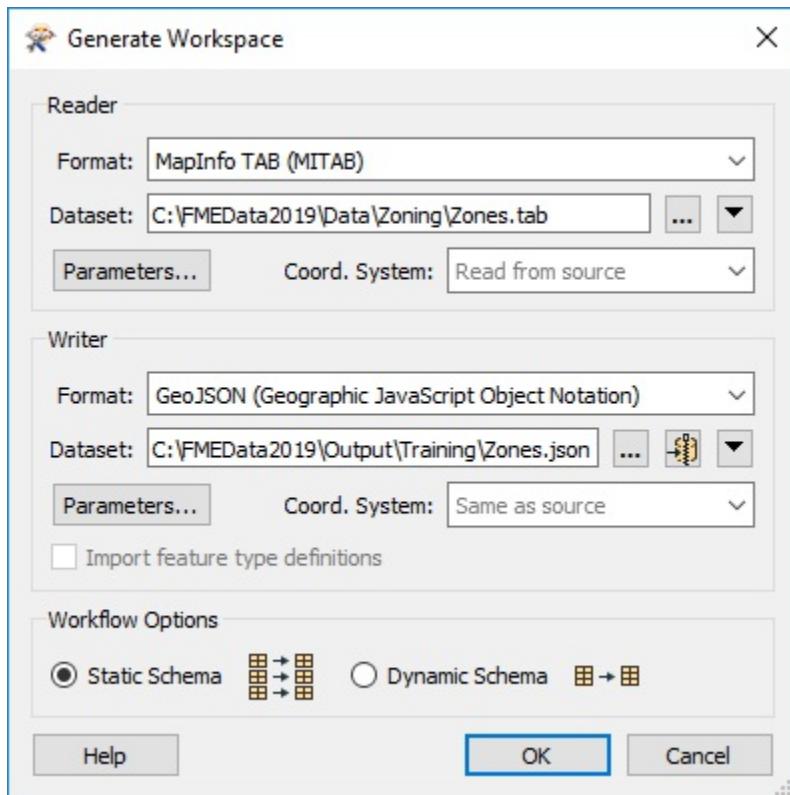


### 3) Define Translation

The Generate Workspace tool opens up a dialog in which to define the translation to be carried out. Fill in the fields in this dialog as follows:

<b>Reader Format</b>	MapInfo TAB (MITAB)
<b>Reader Dataset</b>	C:\FMEData2019\Data\Zoning\Zones.tab
<b>Writer Format</b>	GeoJSON (Geographic JavaScript Object Notation)
<b>Writer Dataset</b>	C:\FMEData2019\Output\Training\Zones.json

The dialog will look like this:



Remember, you can set a format by typing its name, by selecting it from the drop-down list, or by choosing "More Formats" and selecting the format from the full table of formats. For now, ignore the Workflow Options and leave the default of 'Static Schema.'

#### 4) Generate and Examine Workspace

Click OK to close the Generate Workspace dialog. A new workspace will be generated into the FME Workbench canvas. The list of attributes is exposed by clicking the arrow icon on each Zones object:



Then, you can expand the rectangular object labelled Reader Feature Types to contain the exposed attribute list. This object is called a bookmark; we will discuss it more later in the course:



#### 5) Run Workspace

Run the workspace by clicking the run button on the toolbar, or by using Run > Run Workspace on the menu bar. The workspace runs and the log file reports a successful translation:

```

99 -----
100 Features Read Summary
101 -----
102 Zones 416
103 -----
104 Total Features Read 416
105 -----
106 -----
107 Features Written Summary
108 -----
109 Zones 416
110 -----
111 Total Features Written 416
112 -----
113 Translation was SUCCESSFUL with 0 warning(s) (416 feature(s) output)
114 FME Session Duration: 1.7 seconds. (CPU: 0.8s user, 0.2s system)
115 END - ProcessID: 2920, peak process memory usage: 121288 kB, current process memory usage: 86868 kB
116 Translation was SUCCESSFUL

```

## 6) Locate Output

Locate the destination data in Windows Explorer to prove that it's been written as expected (don't forget the Open Containing Folder button from Exercise 1). In the next section, we'll cover how to inspect the dataset visually to ensure that it is correct:

This PC > Local Disk (C:) > FMEData2019 > Output > Training			
Name	Date modified	Type	Size
Zones	4/4/2019 5:35 PM	JSON File	692 KB

## 7) Save Workspace

Save the workspace. We'll be using it in a later exercise. Remember there is a toolbar save button, and on the menu, there is File > Save.

### TIP

*When a translation is run immediately without adjustment it's known as a "Quick Translation". Because FME is a 'semantic' translator, with an enhanced data model, the output from a quick translation is as close to the source data in structure and meaning as possible, given the capabilities of the destination format.*

### CONGRATULATIONS

By completing this exercise you have learned how to:

- Create an FME workspace
- Run an FME workspace

## Data Inspection

To ensure that you're dealing with the right information you need a clear view of your data at every stage of the transformation process.

Data inspection meets this need: it is the act of viewing data for verification and debugging purposes, before, during, or after a translation.

In FME Desktop, you can use two data inspection tools: in Workbench via the **Visual Preview** window, or in a complementary application called the **FME Data Inspector**. In this course we will discuss both, but will primarily use Visual Preview.

FME data inspection tools let you view data in any supported format. They are used primarily to preview data before translation or to verify it after translation. You can use them to set up and debug workspaces by inspecting data *during* the translation.

FME data inspection tools are not designed to be full-featured spreadsheet, database, or mapping applications. They have no analysis or editing functionality, and the tools for symbology modification or printing are intended for data validation rather than producing output.

## What Can Be Inspected?

A number of different aspects of data may be inspected, including the following:

- **Format:** Is the data in the expected format?
- **Schema:** Is the data subdivided into the correct layers, categories, or classes?
- **Spatial:** Is the data spatial and does it have the correct geometry?
- **Symbolology:** Is the color, size, and style of each spatial feature correct?
- **Attributes:** Are all the required attributes present? Are all integrity rules being followed?
- **Quantity:** Does the data contain the correct number of features?
- **Output:** Has the translation process restructured the data as expected?

### FME Lizard says...

*Inspecting data before using it helps to detect problems before they affect the translation.*

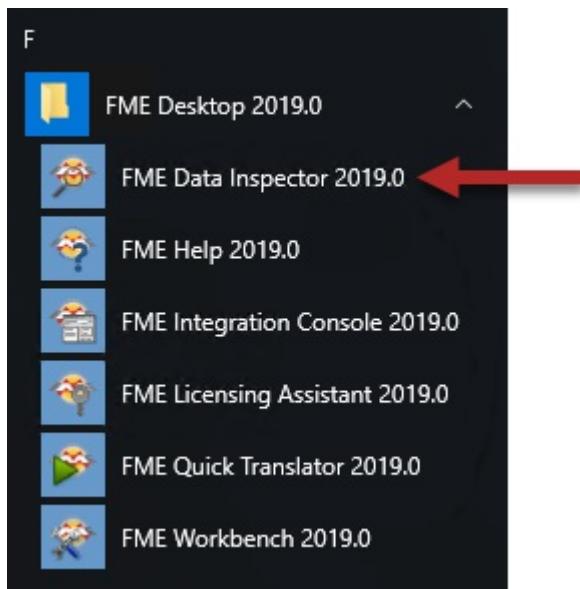
*Inspecting your data before, during, and after building your workspace ensures you are working with the correct data and lets you spot data quality problems early on.*

## FME Data Inspector

FME Data Inspector is a standalone application for inspecting data.

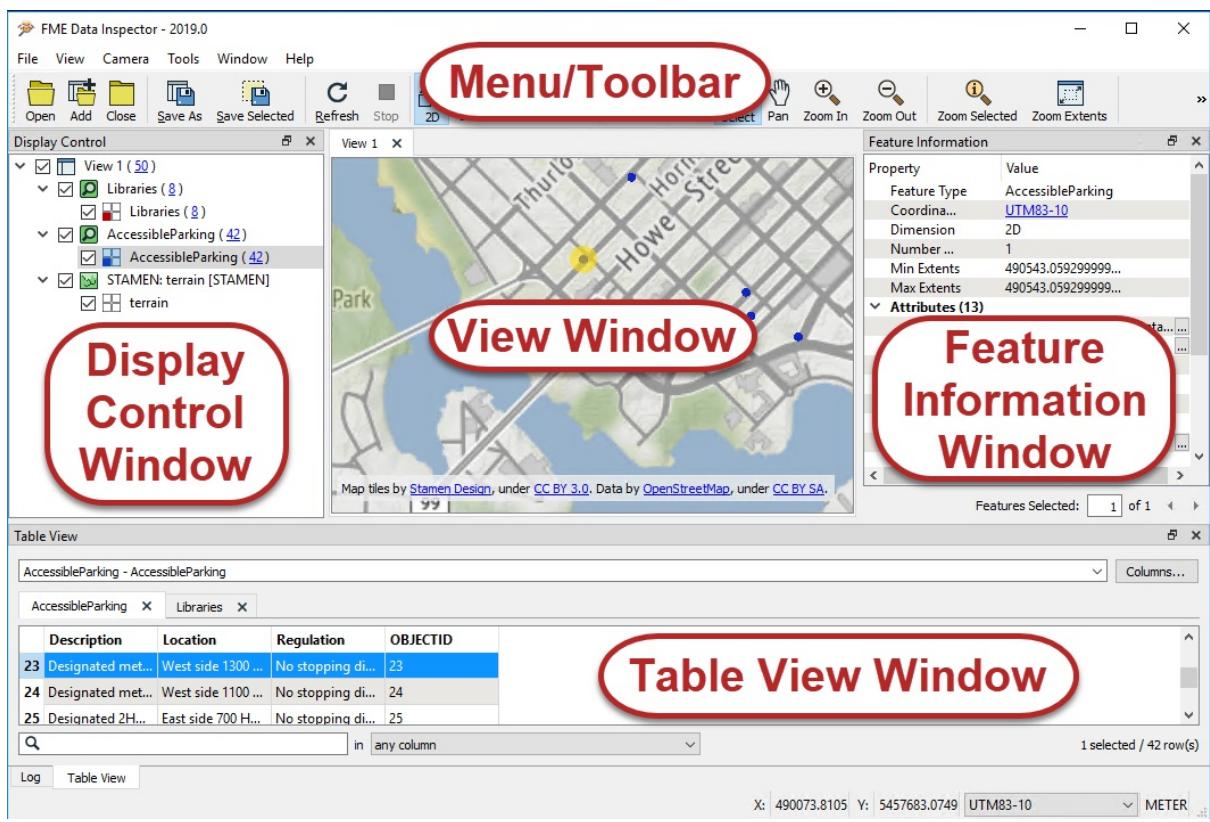
### Starting FME Data Inspector

To start the Data Inspector, locate it in the Windows start menu:



### Major Components of FME Data Inspector

When FME Data Inspector opens a dataset, it looks something like this:



Map tiles by Stamen Design, under CC-BY-3.0. Data by OpenStreetMap, under CC-BY-SA.

These components function the same as Visual Preview, with the following differences:

- There is a menu bar.
- The Graphics window is called the View window and can have multiple tabs called Views, each displaying different datasets.
- You can open data (File > Open Dataset), which opens it in a new View, or you can add data (File > Add Dataset), which adds data to the open View.
- A few additional tools are available, including [Mark Location](#) and [Measure Distances](#).

## NEW

*New for 2019.0: if the data you are inspecting is spatial, it will open in the View window. If it is not, it will open in Table View.*

## Viewing Data

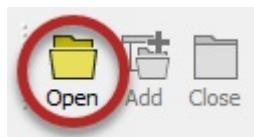
The FME Data Inspector provides two methods for viewing data: opening or adding.

**Opening** a dataset opens a new view window for it to be displayed in. **Adding** a dataset displays the data in the existing view window; this way multiple datasets can be viewed simultaneously.

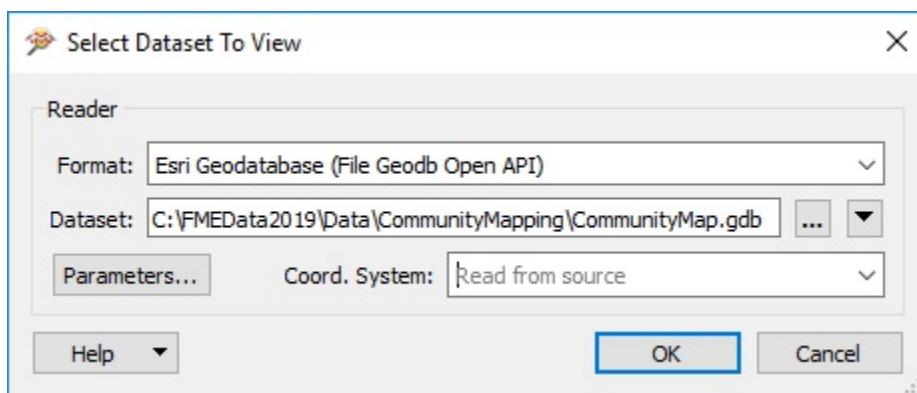
### Opening a Dataset

Datasets can be opened in the FME Data Inspector in several ways.

- Select File > Open Dataset from the menu bar
- Select the toolbar button Open Dataset.
- Drag and drop a file onto any window (except the View window)
- Open from within Workbench



All of these methods cause a dialog to open in the FME Data Inspector in which to define the dataset to view.



### Adding a Dataset

Opening a dataset causes a new View tab to be created and the data displayed. To open a dataset within an existing view tab requires the use of tools to add a dataset.

- Selecting File > Add Dataset from the menu bar
- Selecting the toolbar button Add Dataset
- Dragging and Dropping a file onto the view window



## Windowing Tools

Once data has been opened in the FME Data Inspector, there are many tools available for altering the view.

- Pan
- Zoom In
- Zoom Out
- Zoom to a selected feature
- Zoom to the full extent of the data



### TIP

Press the *Shift* key on the keyboard and it will activate the zoom-in tool in the Inspector. Press the *Ctrl* key, which will activate the zoom-out tool. Release the key to revert to the previous tool.

This functionality allows users to quickly move between select and navigation modes at the press of a key, so there's no need to click between select and navigation tools on the menu bar or toolbar.

## Inspecting Data

The FME Data Inspector includes several inspection tools, but two are particularly important:

- Select individual feature(s)
- Measure a distance within a View window



The select tool button is like a toggle. By default it is active when you start the FME Data Inspector; if you click it again - or select a windowing tool - you turn the select tool off.

The results of a selected feature are shown in the Feature Information window.

### Feature Information Window

The upper part of this window reports on general information about the feature; which feature type (layer/table) it belongs to. If the feature is spatial, it will also report which coordinate system it is in, whether it is two- or three-dimensional, and how many vertices it possesses.

Feature Information	
Property	Value
Feature Type	Libraries
Coordinate System	<a href="#">UTM83-10</a>
Dimension	2D
Number of Vertices	1
Min Extents	491595.35539999977, 5458555.6753
Max Extents	491595.35539999977, 5458555.6753
<b>Attributes (11)</b>	
BookCount (32 bit integer)	1188303
Circulation (32 bit integer)	1751704
filegdb_type (string)	geodb_point
fme_geometry (string)	fme_point
fme_type (string)	fme_point
LibraryAddress (encoded: UTF-16LE)	350 W Georgia St
LibraryID (encoded: UTF-16LE)	BVA003
LibraryName (encoded: UTF-16LE)	Central Branch
LibraryURL (encoded: UTF-16LE)	<a href="http://www.vpl.ca/location/central-library">http://www.vpl.ca/location/central-library</a>
OBJECTID (32 bit unsigned integer)	3
UserVisits (32 bit integer)	1958528
<b>IFMEPoint</b>	
	491595.35539999977, 5458555.6753

The middle part reports the attributes associated with the feature, including user attributes and format attributes (for example *fme\_type*).

The lower part reports the geometry of the feature. It includes the geometry type and a list of the coordinates that go to make up the feature.

### Table View Window

Also available is a window called the Table View.

Table View

CommunityMap [FILEGDB] - Libraries

AccessibleParking   Libraries

	LibraryName	LibraryAddress	LibraryURL	BookCount	UserVisits
1	Strathcona	592 E Pender St	http://www.vpl...	17394	
2	Carnegie	401 Main St	http://www.vpl...	26689	4

< >

in any column 8 row(s)

Log Table View

The table view is a way to inspect data in a tabular, spreadsheet-like, layout. Although it does not have the same depth of information as shown by the Information Window, the Table View is particularly useful for inspecting the attribute values of multiple features simultaneously.

You can switch back and forth between feature types in the Table View by clicking the dropdown menu at the top of Table View.

Table View

CommunityMap [FILEGDB] - AccessibleParking

CommunityMap [FILEGDB] - AccessibleParking

CommunityMap [FILEGDB] - CommunityCentres

CommunityMap [FILEGDB] - DrinkingFountains

CommunityMap [FILEGDB] - FoodVendors

CommunityMap [FILEGDB] - GarbageSchedule

CommunityMap [FILEGDB] - Libraries

CommunityMap [FILEGDB] - Parks

CommunityMap [FILEGDB] - TransitStations

4	Parking meter	South side 100 ...	<null>	4	
---	---------------	--------------------	--------	---	--

in any column 42 row(s)

Log Table View

### TIP

*To improve performance, tables are not all displayed automatically, only when selected from the drop-down list, or when queried in the current view window.*

### Exercise 3

### Basic Data Inspection

<b>Data</b>	Zoning Data (GeoJSON) Neighborhoods (Google KML)
<b>Overall Goal</b>	Inspect the output from a previous translation
<b>Demonstrates</b>	Basic data inspection with the FME Data Inspector
<b>Start Workspace</b>	None
<b>End Workspace</b>	None

In the previous exercise, you were asked to convert some data between formats. Before you send the converted data out, you should inspect it to make sure it is correct. Let's see how the FME Data Inspector interface works by inspecting the output from that quick translation.

#### 1) Start FME Data Inspector

Start the FME Data Inspector by selecting it from the Windows start menu. You'll find it under Start > FME Desktop 2019.0 > FME Data Inspector 2019.0.

#### 2) Open Dataset

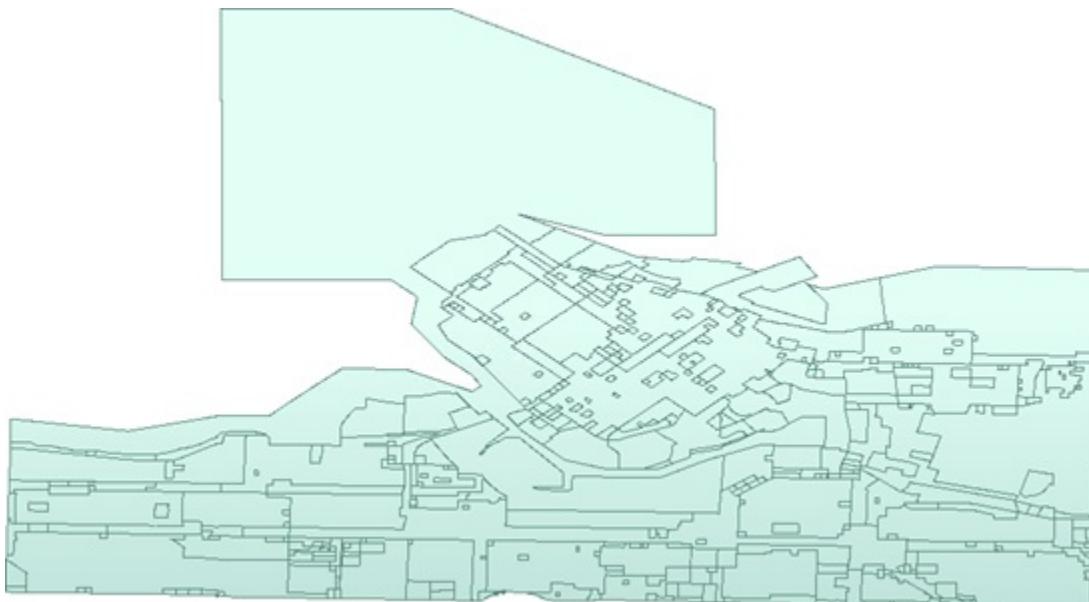
The FME Data Inspector will start up and begin with an empty view display.

To open a dataset, select File > Open Dataset from the menu bar. When prompted, fill in the fields in the Select Dataset dialog as follows:

<b>Reader Format</b>	GeoJSON (Geographic JavaScript Object Notation)
<b>Reader Dataset</b>	C:\FMEData2019\Output\Training\Zones.json

*Note: If you can't find the dataset - maybe you didn't complete the first exercise, or wrote the data to a different location - then you can open the original zoning dataset as described in Exercise 2.*

The GeoJSON dataset looks like this:



#### 3) Browse Data

Use the windowing tools on the toolbar to browse through the dataset, inspecting it closely. Use the Select tool to select individual features and inspect the information in the Feature Information Window.

Try right-clicking in the different Data Inspector windows, to discover functionality that exists on context menus.

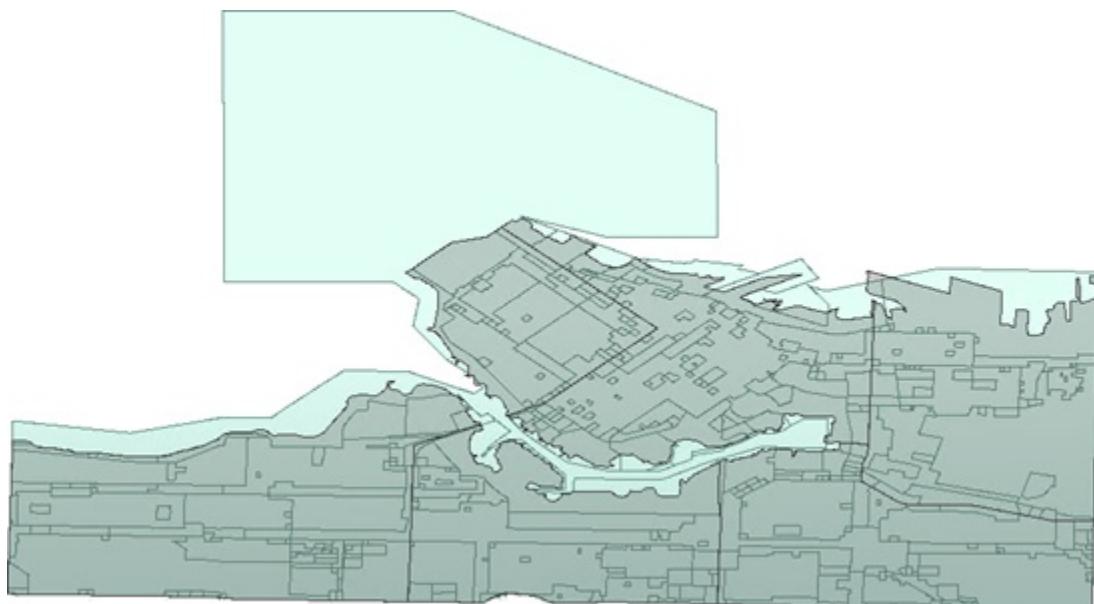
#### 4) Add Dataset

Let's add a second dataset to the display to compare to our zoning data. This dataset will be a KML file of neighborhood boundaries. Then we'll be able to see which neighborhood each zone overlaps.

To add a dataset, select File > Add Dataset from the menu bar. When prompted, fill in the fields in the Select Dataset dialog as follows:

Reader Format	Google KML
Reader Dataset	C:\FMEData2019\Data\Boundaries\VancouverNeighborhoods.kml

The display now looks like this:



Use the Table View to practice inspecting the tabular data for each feature type. Click on the dropdown arrow at the top of Table View and switch back and forth between the Zones.json and Neighborhoods.kml tables:

Zones [GEOJSON] - Zones			Columns...
VancouverNeighborhoods [OGCKML] - Document			
VancouverNeighborhoods [OGCKML] - Folder			
VancouverNeighborhoods [OGCKML] - Neighborhoods			
VancouverNeighborhoods [OGCKML] - Style			
Zones [GEOJSON] - Zones			
2	RT-8	Two Family Dwelling	
3	RT-3	Two Family Dwelling	
4	RT-2	Two Family Dwelling	

in  179 row(s)

Without context, the layers in the View Window look confusing. However, we can improve the display by changing symbology and adding a background map. We'll learn how to do that in the next section.

**CONGRATULATIONS**

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

- *Open datasets in a new view in the FME Data Inspector*
- *Add datasets to an existing view in the FME Data Inspector*
- *Use the windowing and inspection tools in the FME Data Inspector*

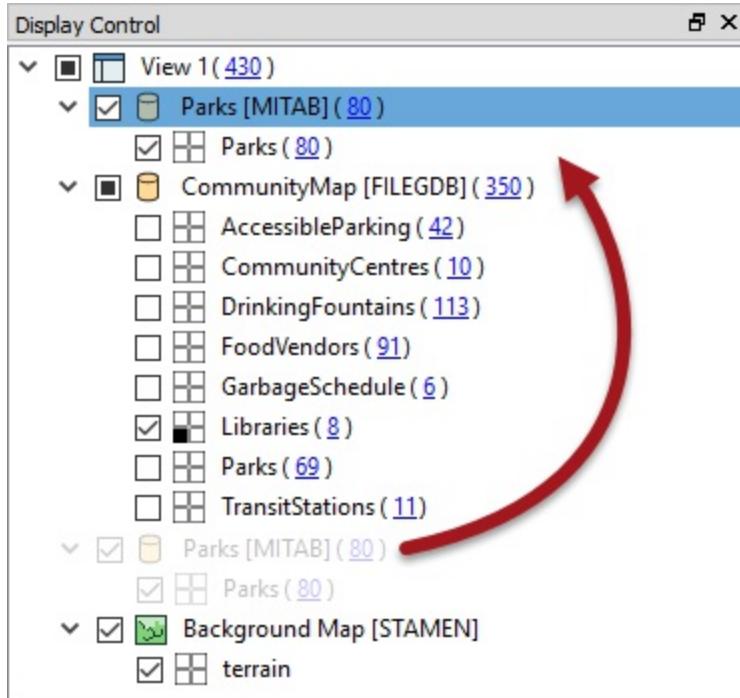
## Data Inspector Display Control

The FME Data Inspector has a number of controls to assist in showing the data in an orderly manner.

### Display Control Window

Management of feature display order is carried out in the Display Control window.

Each dataset and feature type can be dragged above any other to promote its display order in the View window:



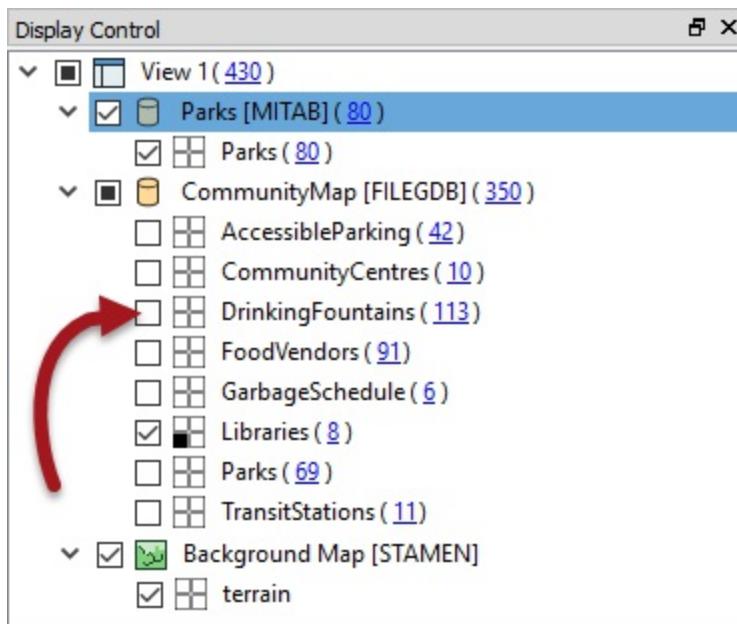
Feature Types can only be ordered within their container dataset.

For example, "Libraries" can be promoted above "GarbageSchedule" in the display, but they cannot by themselves be promoted above the separate Parks dataset.

---

### Display Status

Each level of the Display Control window has a checkbox to turn data on and off at that level:

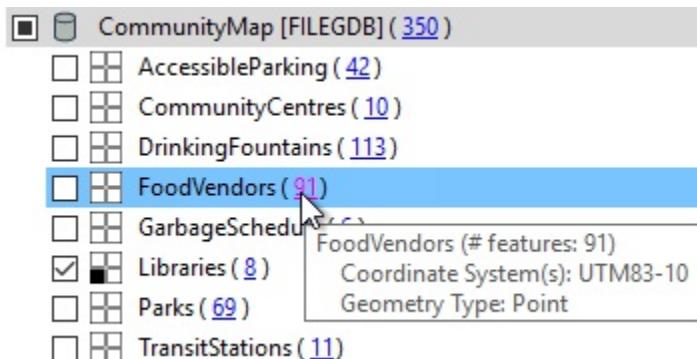


You can choose to turn off individual feature types or an entire dataset at once.

## Feature Counts

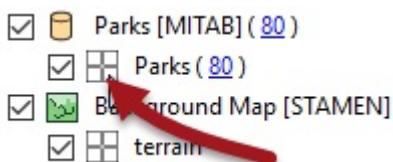
Each Feature Type in the Display Control window is tagged with the number of features it contains, in parentheses after the feature type name. So in the previous image, we can see that there are 113 drinking fountains (or 113 features called DrinkingFountains) in the city of Vancouver.

Clicking a feature count selects all of those features for display in the Feature Information and Table View windows, and also activates the tool File > Save Selected Data As:

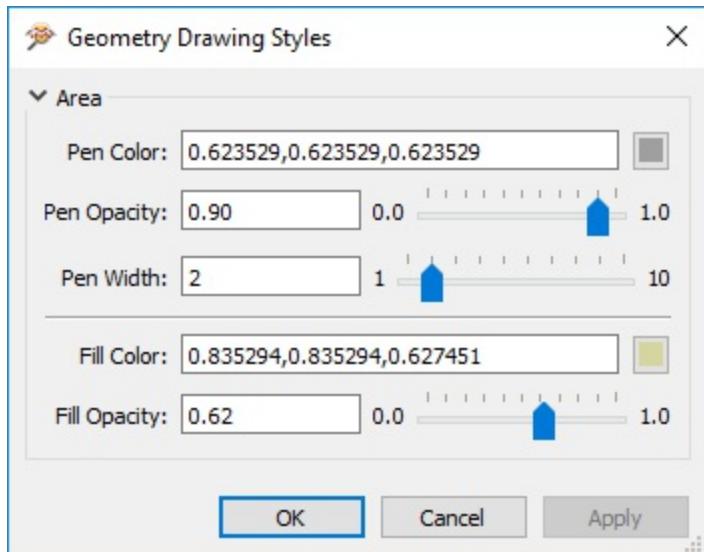


## Style

Each feature type can be assigned a different color or style that applies to its geometries. To access this functionality click on the style icon for that feature type in the Display Control window:



The Drawing Style dialog allows you to set all manner of symbology for a feature. That includes feature color (and degree of transparency), point icon/symbol, point size, line thickness, etc.:



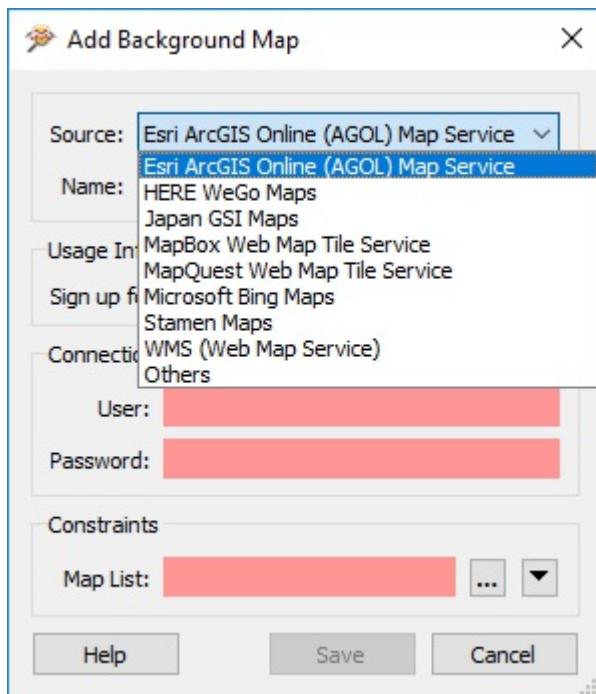
Notice that the dialog only displays symbology options for the type of geometry available. Here it is for a polygon feature. Other geometry types display different options.

## Background Maps

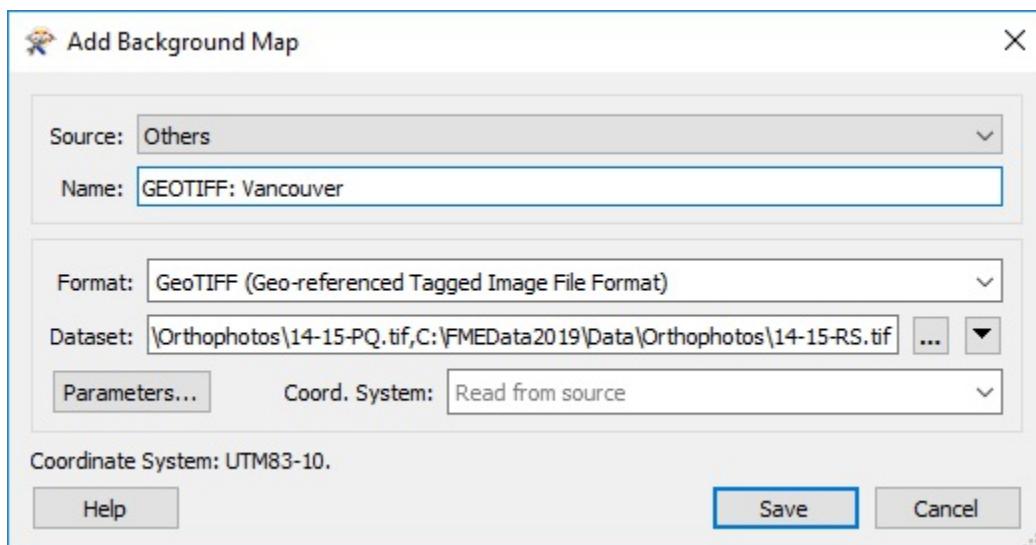
The ability to view maps (or other imagery) as a backdrop to your spatial data is activated by clicking Add a background map on the Data Inspector toolbar:



The background map dialog lets the user add a background map from a web service. Some of these - such as ArcGIS Online - require an existing account:



You can also use an existing dataset (of any FME-supported format) as a backdrop, by selecting Others and filling in the Format and Dataset:

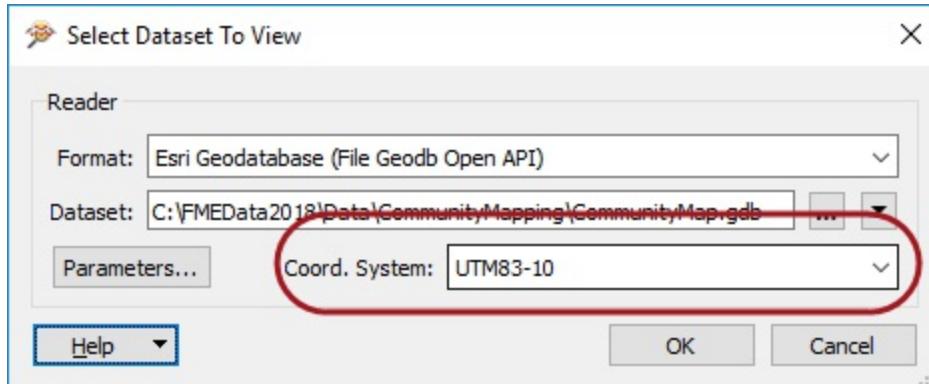


### TIP

*Keep in mind you can select multiple datasets of the same format when adding a background map (or any reader). In the image above, the user is adding many orthophoto raster images.*

## Coordinate Systems

Source data must be referenced with a valid coordinate system to view it with a background map. If the coordinate system is not recorded in the dataset itself, you may enter it into a field when opening the dataset:



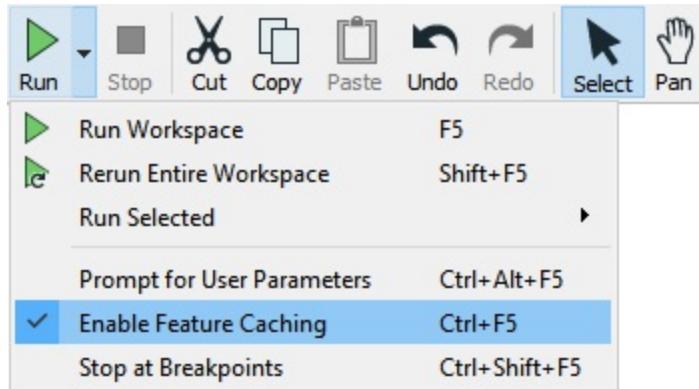
FME can display the source data against a background map, even when there are several source datasets of differing coordinate systems. FME does this by reprojecting the data to the coordinate system used by the background map. Therefore it's recommended that you turn off the background maps when you want to inspect the data in its original form.

## FME Lizard says...

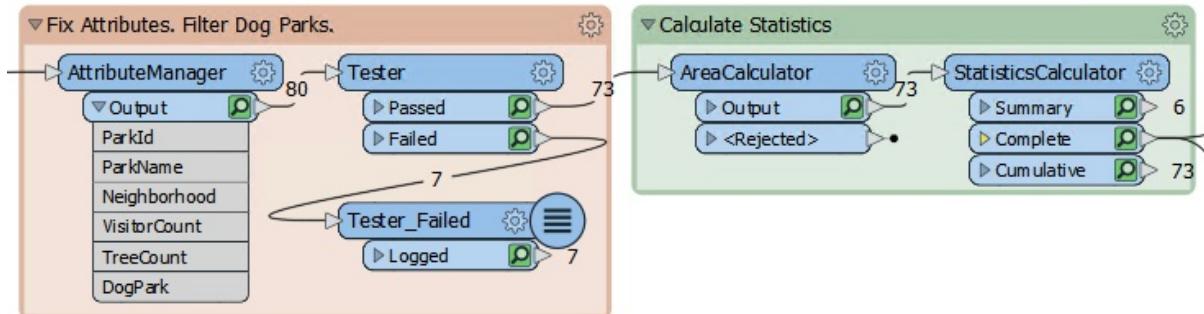
*You can adjust the symbology and display order of the background map in the Display Control window, just as you can for any normal dataset.*

## Feature Caching

Sometimes it's important to be able to inspect data at any step of the translation. This behavior is activated using Run > Run with Feature Caching on the menu bar:



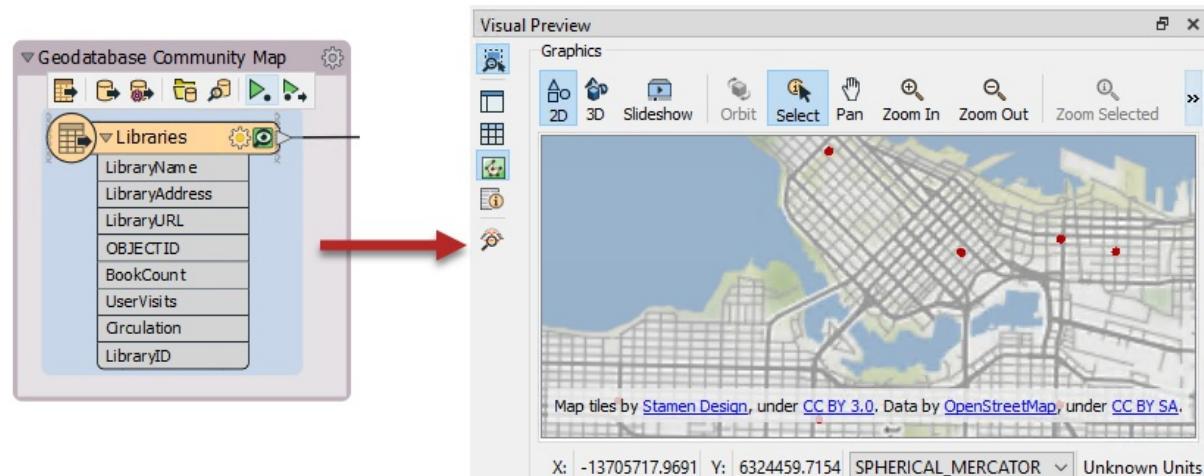
With this option active, FME generates caches at every step of the translation when the workspace is run:



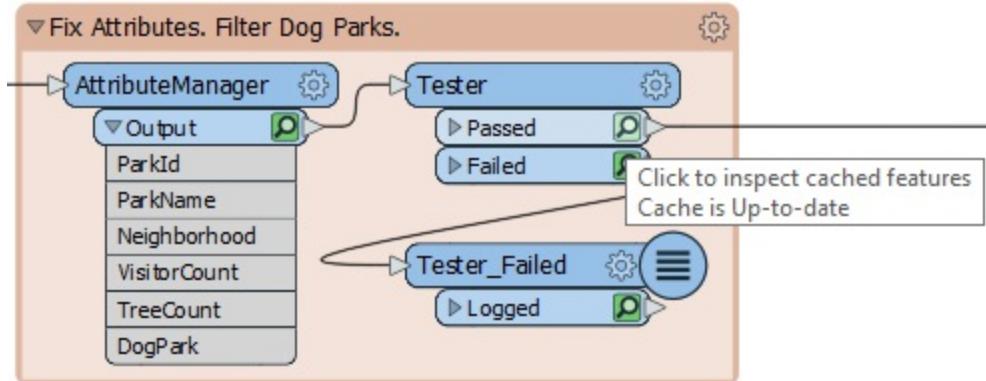
The caches are indicated by the small icons on each object. In the above screenshot, the caches are green, but if any parameters change that could affect the data in the cache, they turn yellow. Yellow caches are considered invalid; you can still inspect them, but their data may no longer accurately represent the results of the translation.

## Inspecting Cached Data

With Toggle Automatic Inspect on Selection enabled, if you click a feature type or transformer with a cache, it will automatically be displayed in Visual Preview (covered in the next section):



Specific data caches (for example a single output port on a transformer) can be inspected by clicking on the feature cache icon:



Inspecting data this way can allow you to quickly debug and develop workspaces, but keep in mind that if you are inspecting a large number of features or features with complex geometry, the caching required for automatic inspection might introduce performance challenges.

## TIP

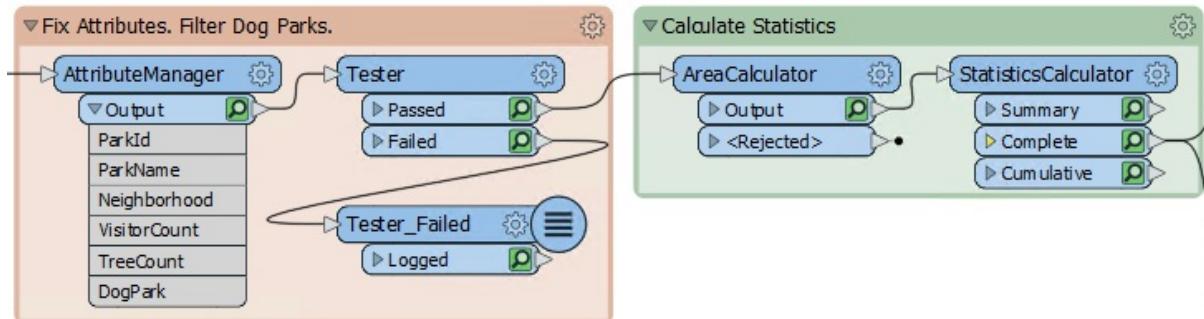
*Be aware that caching data causes the translation to be slower, and to use system resources such as disk space.*

*Data caching is very useful while developing a workspace, but should be turned off before putting a workspace into production.*

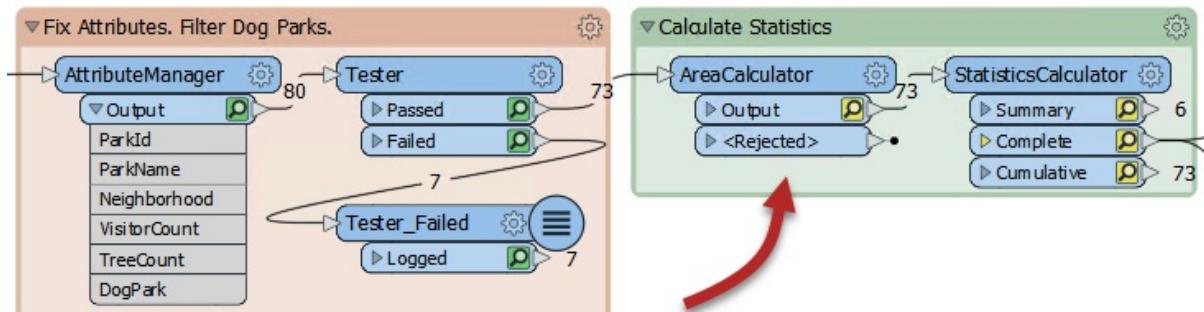
## Partial Runs

When caching is turned on, running a translation causes data to be cached at every part of the workspace. In subsequent runs, those caches can be used instead of having to re-run entire sections of the workspace.

Here, for example, a workspace has been run with caching turned on:

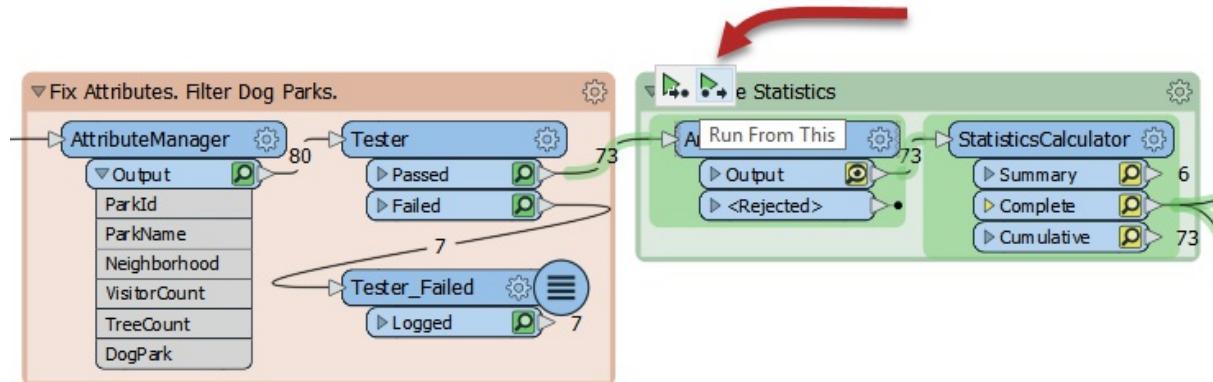


Now the author makes a change to the AreaCalculator parameters:



Notice that the caches change color (to yellow) on the AreaCalculator and subsequent transformers. This color denotes that caches are stale; their data contents no longer match what the workspace would produce.

To get the new results, the author must re-run the workspace. However, they do not have to re-run the entire workspace; they can start the workspace at the point of change - the AreaCalculator:

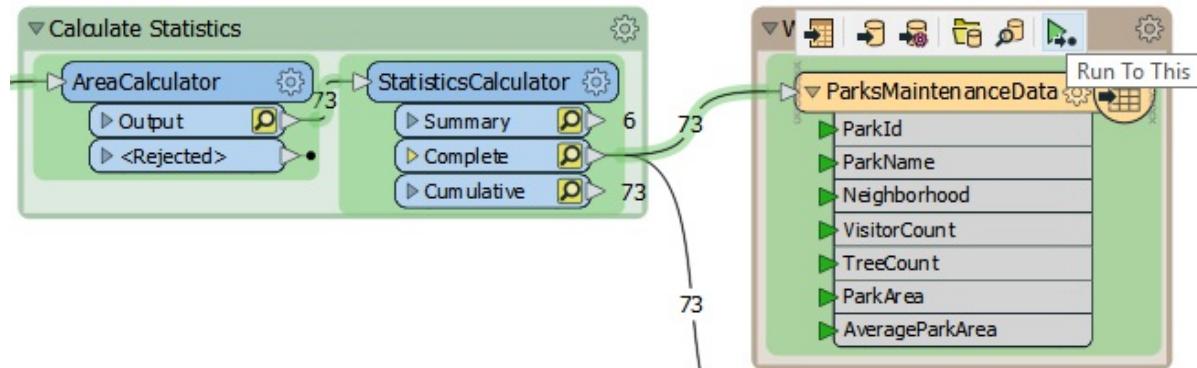


## NEW

*New in FME 2019.0: the keyboard shortcut F5 now runs the workspace from any existing caches, instead of running the entire workspace, which you can do with Shift + F5.*

*Run From This* causes the workspace to run from that point only, using data cached up until that point. Notice how hovering over the option causes all "downstream" transformers to be highlighted. They are the only ones that will be run. That makes the translation quicker.

The other option is *Run To This*. The author could use that option on the writer feature type and get much the same effect:



...but notice how the second branch from the StatisticsCalculator does not get highlighted. It will not be run. That shows how you can avoid running a particular section of workspace, in much the same way as if that connection had been disabled.

## TIP

*A partial run is particularly useful in avoiding re-reading data from its source; especially when the data comes from a slow, remote location such as a web service.*

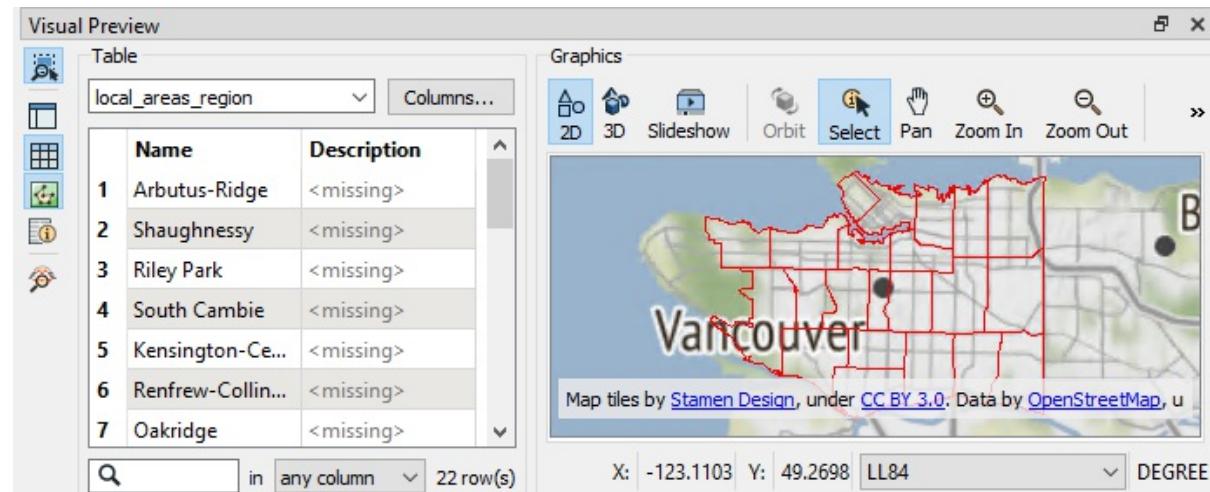
*Also, caches can be saved with the workspace, when it is saved as a template. That means the*

*workspace can be re-run using the caches from a previous session or even from another author!*

## Visual Preview

Visual Preview is an embedded version of FME Data Inspector that displays features in a Workbench window. Many - but not all - of the features available in the stand-alone Data Inspector application are available in Visual Preview. In this course, we will inspect data primarily using Visual Preview and feature caching.

Visual Preview lets you inspect your data directly in Workbench as you build your workspace:



Map tiles by [Stamen Design](#), under [CC-BY-3.0](#). Data by [OpenStreetMap](#), under [CC-BY-SA](#).

### NEW

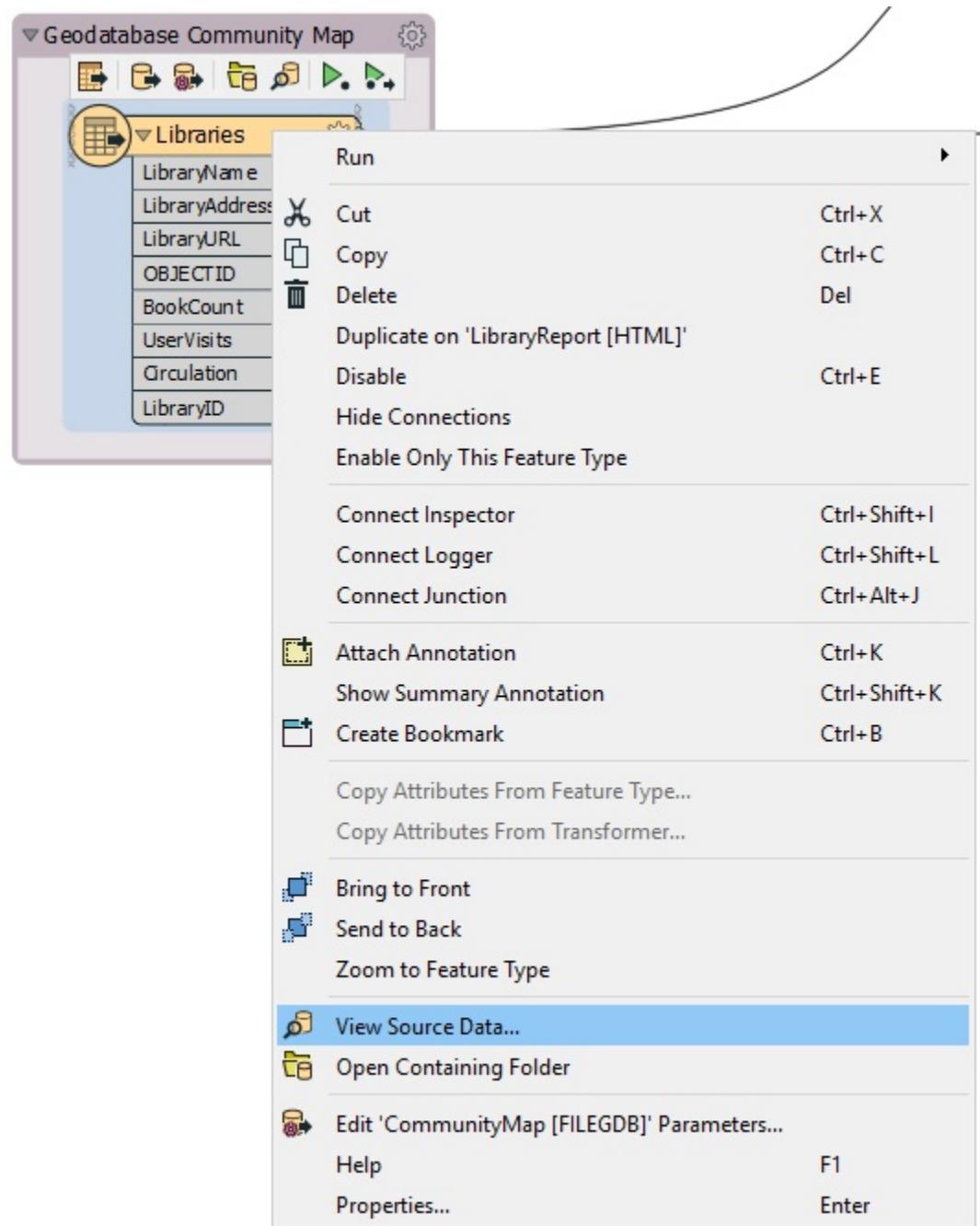
Visual Preview is a new feature for FME 2019.0.

If you would prefer not to use it, you can disable it under FME Options > Workbench > Data Inspection > Inspect with Data Inspector when Visual Preview window closed.

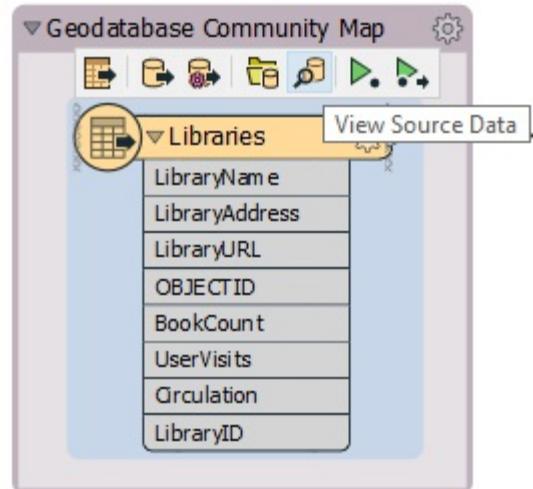
## Viewing Data in Visual Preview

You can view features in Visual Preview in these ways:

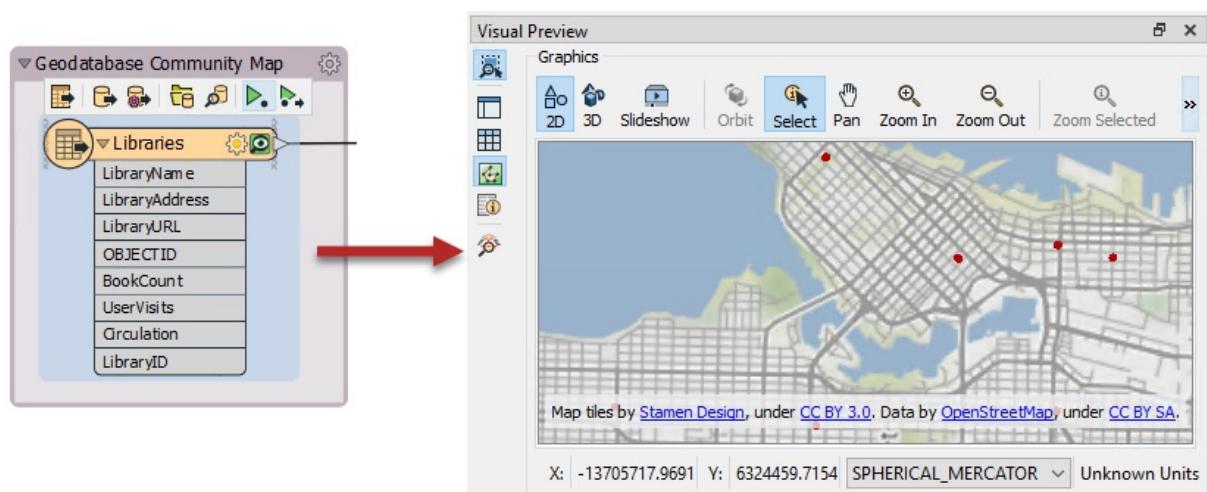
Right-click on a feature type in the Navigator or Workbench Canvas, and select View Source Data:



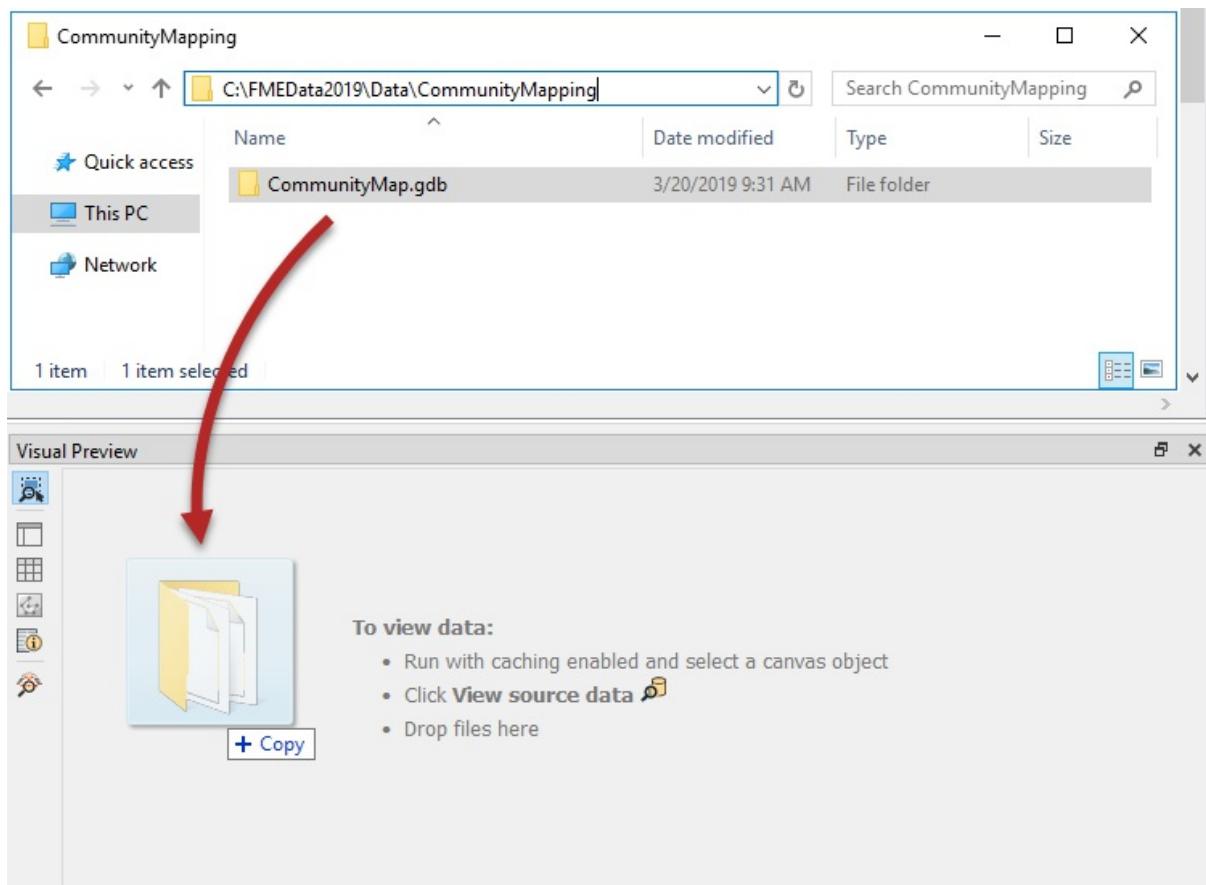
On the canvas, click the View Source Data icon on the mini-toolbar over a feature type or some transformers:



Run a workspace with feature caching enabled and then select cached objects:

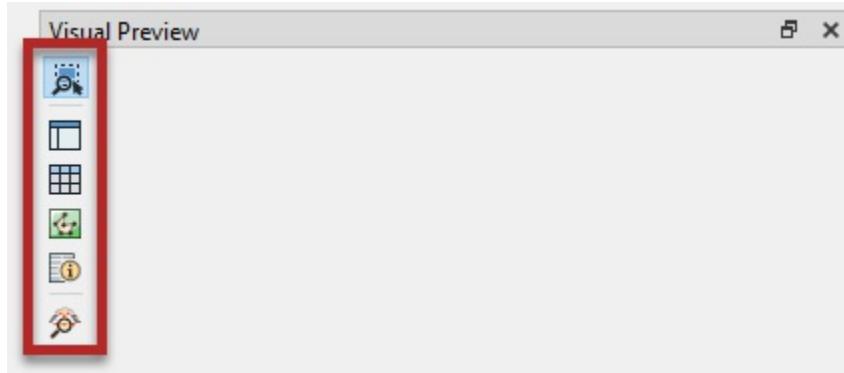


Drag and drop data files onto the pane:



## Major Components of Visual Preview

The Visual Preview pane has options for toggling on and off several other windows to provide different ways of inspecting your data. You can do so using the buttons on the left side of the Visual Preview pane:



## Options

The **Toggle Automatic Inspection on Selection** button lets you decide if data should automatically be displayed in Visual Preview when a cached object is selected. It is on by default.

The **Open in Data Inspector** button opens the displayed data in Data Inspector.

## Windows

The remaining buttons on the Visual Preview toolbar all toggle windows on and off. These windows replicate their function in Data Inspector.

The **Toggle Graphics View** button controls the Graphics window, the area where spatial data is displayed.

The **Toggle Display Control** button  controls the Display Control window.

The **Toggle Table View** button  controls the Table View.

The **Toggle Feature Information window** button  controls the Feature Information window. Just like in Data Inspector, when you select a feature in the Graphics or Table View window, the Feature Information window shows information about that feature.

## Exercise 4

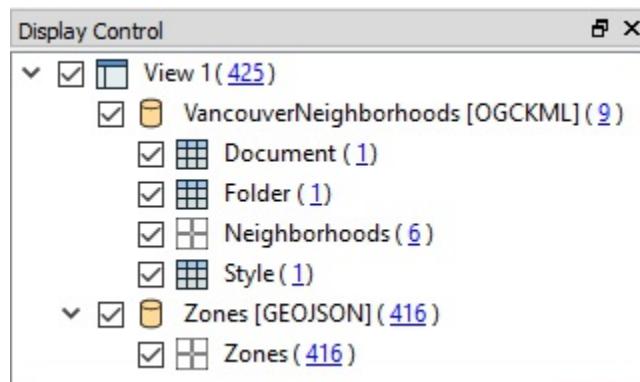
## The FME Data Inspector

<b>Data</b>	Zoning Data (GeoJSON) Neighborhoods (Google KML)
<b>Overall Goal</b>	Set up dataset display
<b>Demonstrates</b>	Use of Display Control and Background Maps in the Data Inspector
<b>Start Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\Basics-Ex4-Begin.fmw
<b>End Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\Basics-Ex4-Complete.fmw

In the previous exercise, we inspected some data from a translation and added a second dataset. Now we can rearrange the data to make the display clearer.

### 1) Start FME Data Inspector

Continue in the FME Data Inspector from the previous exercise. You should have both the converted zones data (as GeoJSON) and a dataset of neighborhood boundaries in KML. The Display Control window looks like this:



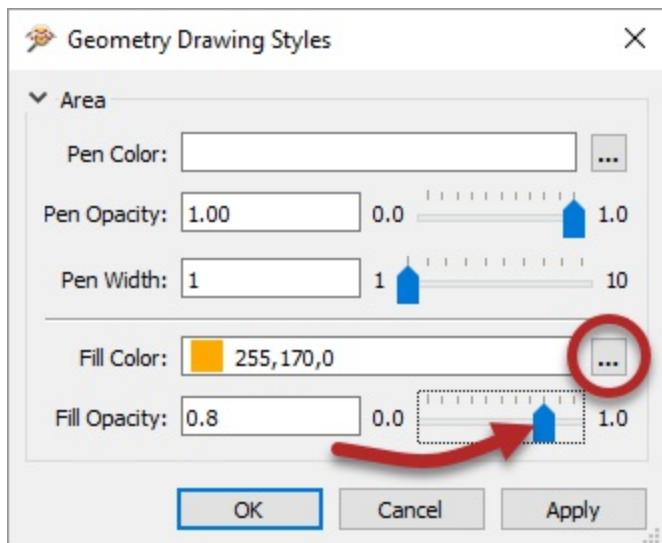
### 2) Set Neighborhoods Symbology

The Display Control window shows a number of different layers in the VancouverNeighborhoods dataset. In reality, most of these are tabular (non-spatial) items. The layer we are interested in is called Neighborhoods.

Click the symbology icon for the Neighborhoods data in the Display Control window:



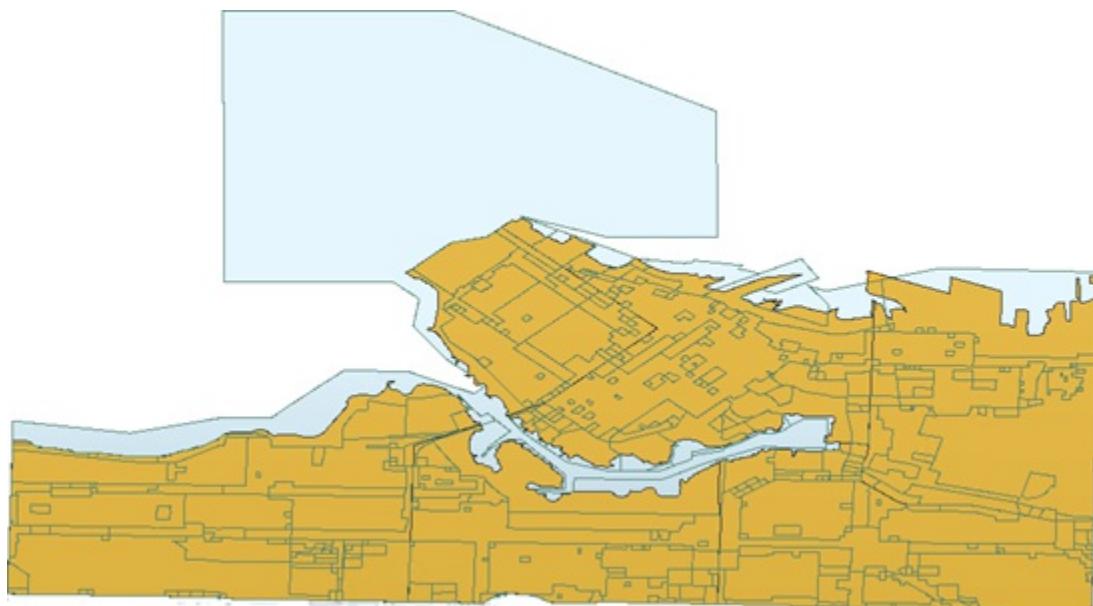
Set the color to be a neutral shade (like orange) and increase the opacity value to 0.8:



### 3) Set Draw Order

The previous change makes it clear that the zone features are below the neighborhoods in the drawing order. To solve this problem drag the Zones dataset above the VancouverNeighborhoods dataset in the Display Control Window.

At the same time set a color for the zones data fill color and reduce the opacity value to 0.1. The main view will now look like this:



If you select a zone feature, you'll see that it has both a ZoneCategory and ZoneName attribute. You might not realize, but there is a relationship between those attributes. Each ZoneName belongs to a specific ZoneCategory, where Category:Name is a 1:Many relationship.

Let's clarify the display by merging all the features into one feature per ZoneCategory. We can do that in FME Workbench.

### 4) Return to Workspace

Return to FME Workbench. Open the workspace saved in Exercise 2, or the workspace listed above.

What we'll do here is use what we call a transformer. This object is something we'll cover in more detail in the chapter on Data Transformation. It is an object to transform data in some way.

Click on the dark line connecting the Reader Feature Type and Writer Feature Type. Start typing the word "Dissolver":



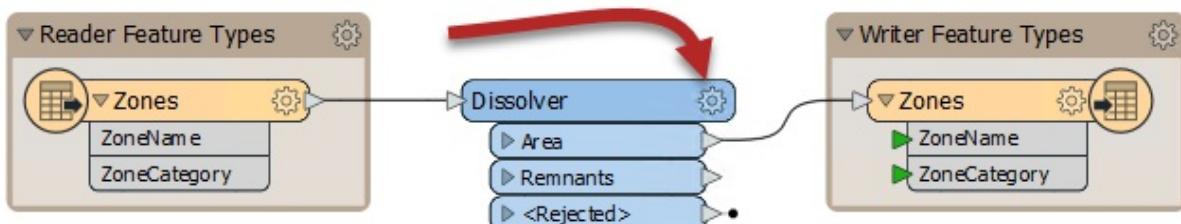
When you see the Dissolver transformer appear in the list, double-click on it to place it into the workspace. The result will look like this:



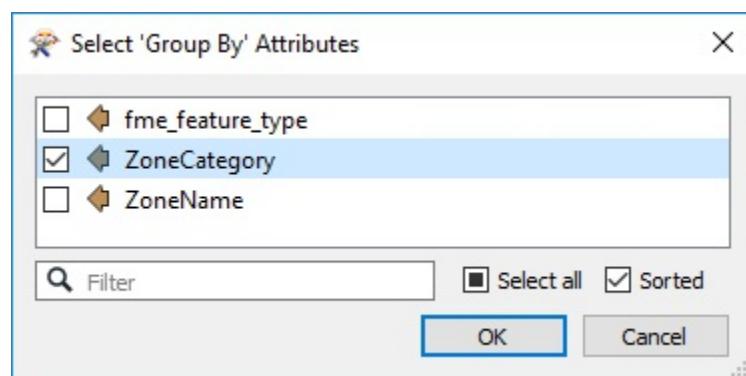
The Dissolver parameter will merge all features with a common attribute value.

## 5) Set Dissolver Parameters

Click on the cogwheel icon on the Dissolver transformer:



Doing so will open a parameters dialog for the transformer. Click the ellipsis (...) button next to the Group By parameter. In the dialog that opens, select the ZoneCategory attribute and click OK:



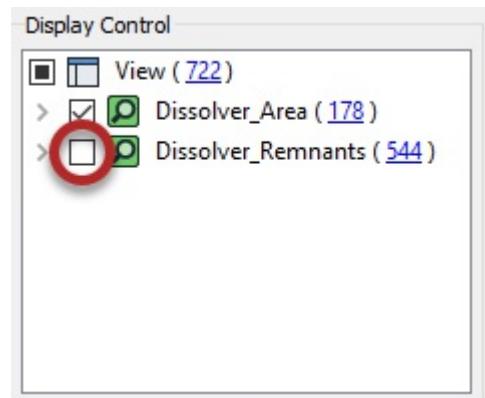
Click OK again to close the parameters dialog.

## 6) Inspect with Feature Caching in Visual Preview

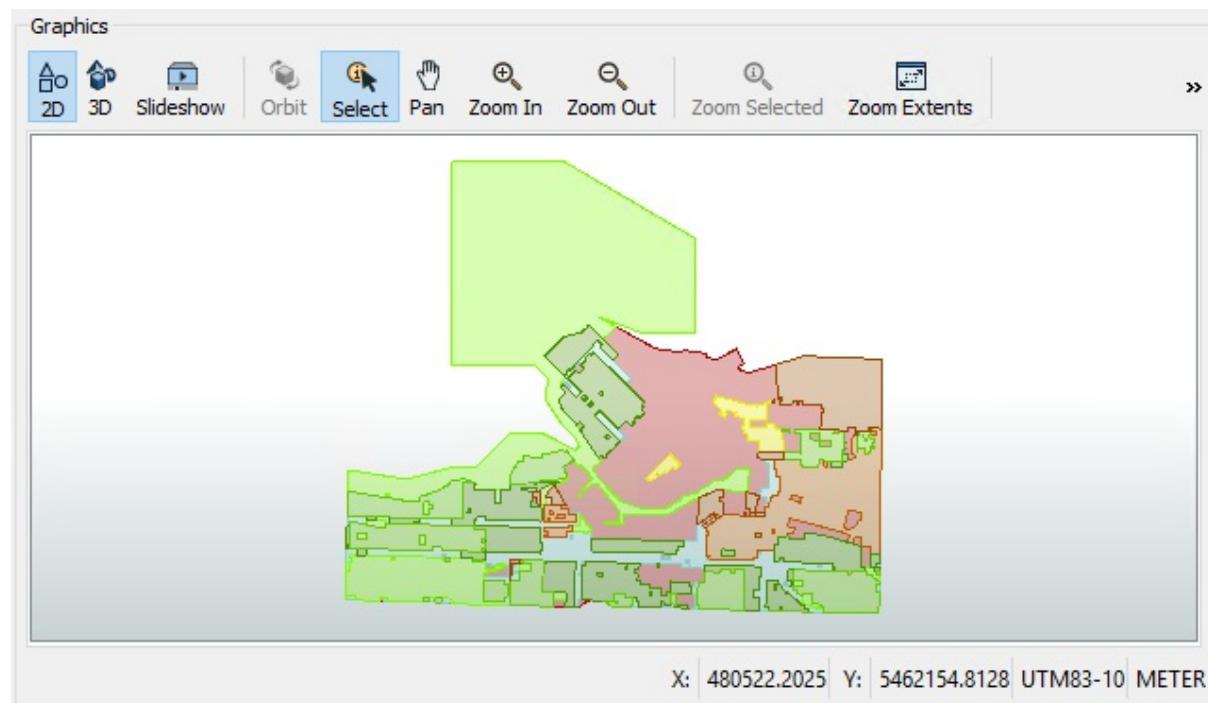
Save the workspace. Let's try using feature caching and Visual Preview now. Click the Dissolver to select it, then click Run to This:



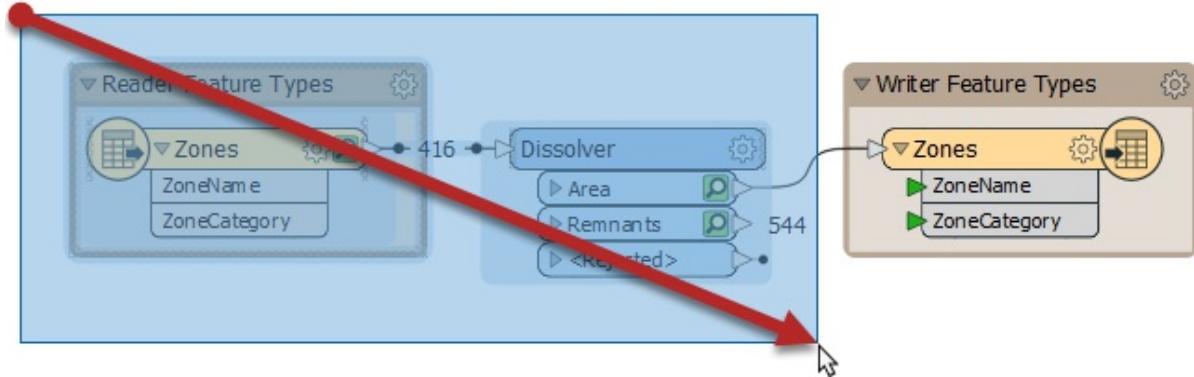
Workbench will cache the Zones feature type and then create two caches on the Dissolver: Area and Remnants. As long as the Dissolver remains selected, Remnants will display automatically in Visual Preview because it is the last output port on the transformer, but Area is the one we are interested in. You can change which port is displayed in Visual Preview by unchecking Dissolver\_Remnants in the Visual Preview Display Control window:



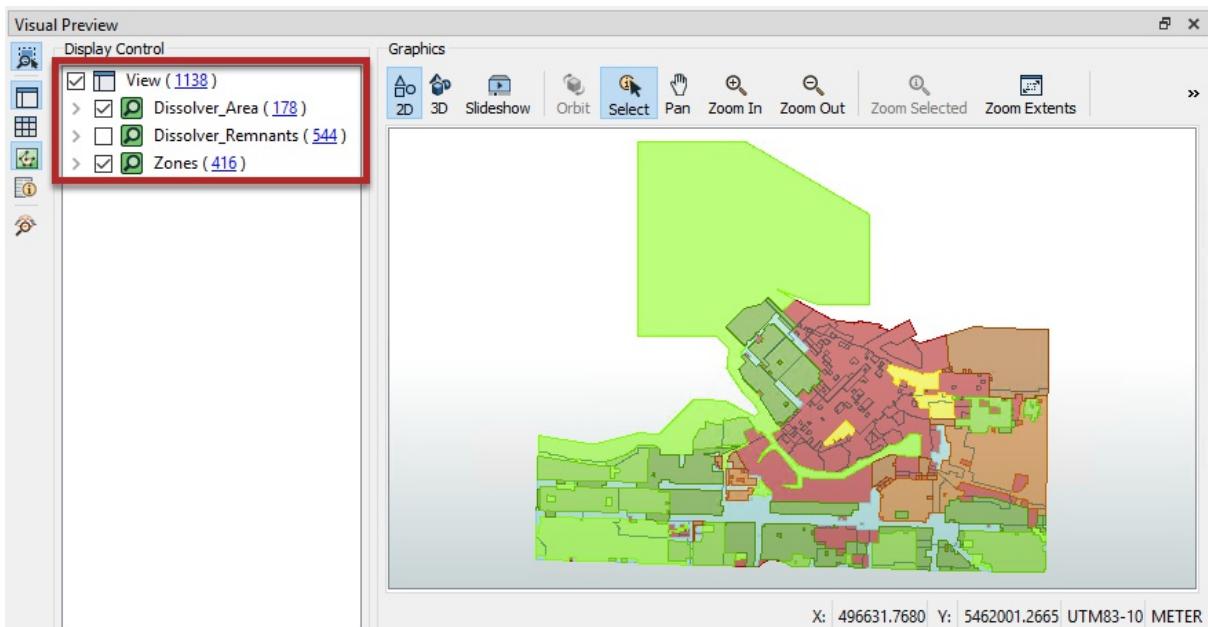
The Graphics window should now show that the Dissolver has simplified the original data by combining adjacent polygons with the same zoning category:



You can compare the before and after results by selecting the Zones feature type as well as the Dissolver transformer. There are two ways to do that: click to select one, then Ctrl click to select the other, or click and drag a rectangle to select both:



Then the original Zones data will be added to your Display Control window and you can toggle the Zones cache on and off to compare (make sure Dissolver\_Remnants stays off):



Now that we are using Visual Preview, you should notice that nearly all of the same tools are available here as in Data Inspector. We will be using Visual Preview for the remainder of the course except where noted.

## 7) Add Background Map

When inspecting data it will help to have a background map to provide a sense of location. You can choose from many mapping services.

Click **Add a background map** in the Visual Preview Graphics window toolbar:



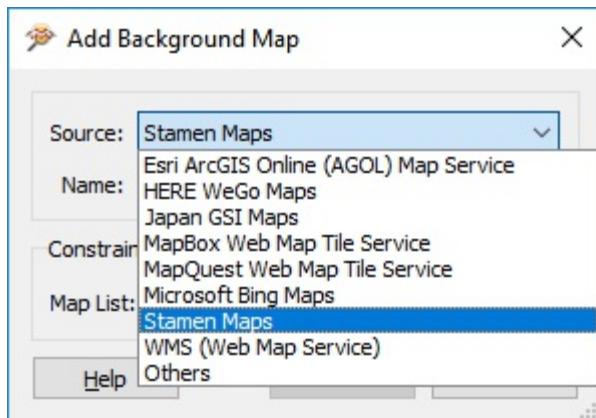
### TIP

*You can right-click the Visual Preview Graphics window and select Background Map > Switch to a new background map, if you prefer:*



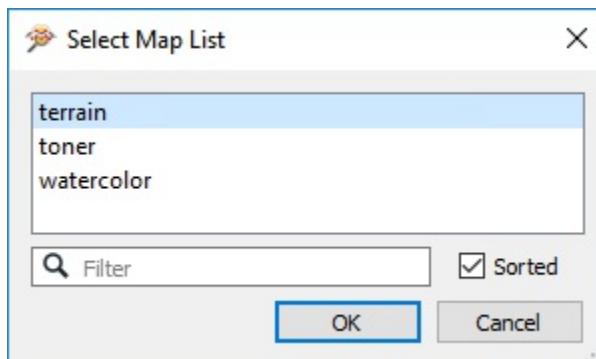
*This method is sometimes easier when working with smaller screens or Visual Preview.*

In the Add Background Map dialog, click the Source dropdown:

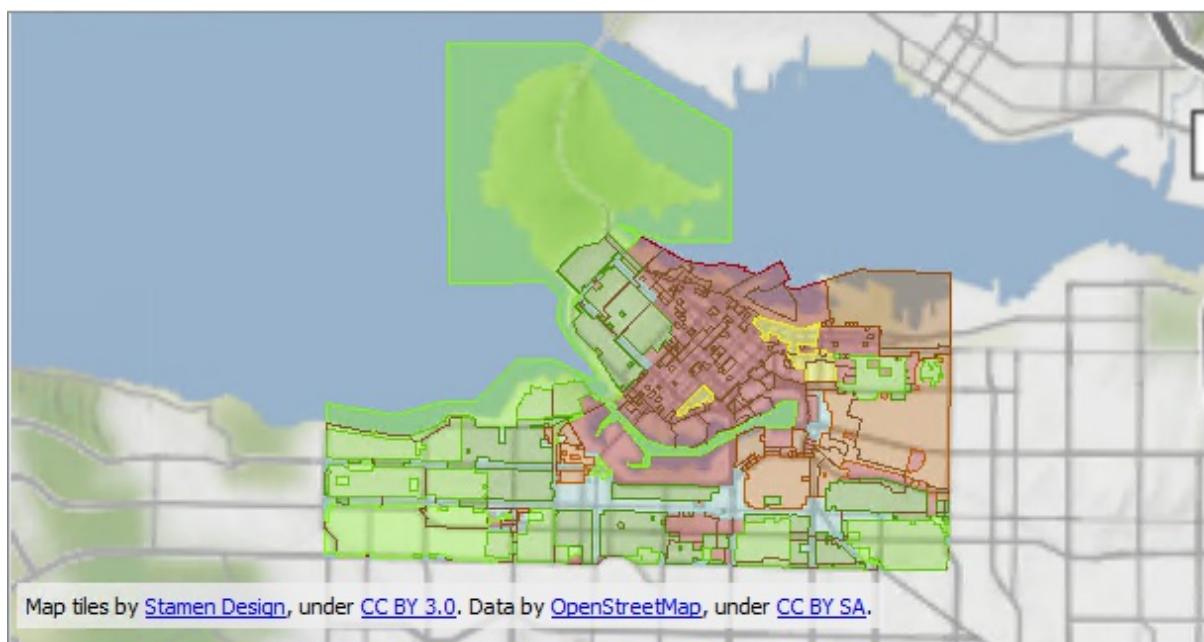


You will notice that FME supports a variety of background map services. Most require an existing account or server, but for this example let's select Stamen Maps, which we can use without an account.

After selecting Stamen Maps, click the ellipsis button to retrieve the list of background map styles available. Select terrain and click OK:



Click OK and then Save to close these dialogs. A background map is added to the display. Notice that the data is reprojected to match the coordinate system of the chosen background. For example, the Stamen Maps background causes the data to reproject to Spherical Mercator, with a clear change of shape:

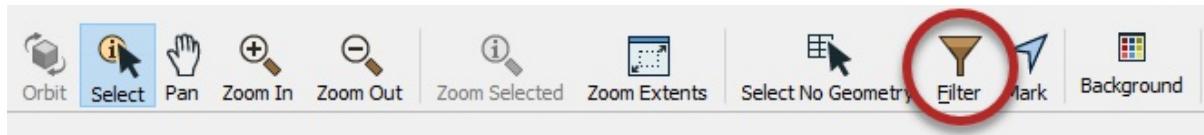


Map tiles by [Stamen Design](#), under [CC-BY-3.0](#). Data by [OpenStreetMap](#), under [CC-BY-SA](#).

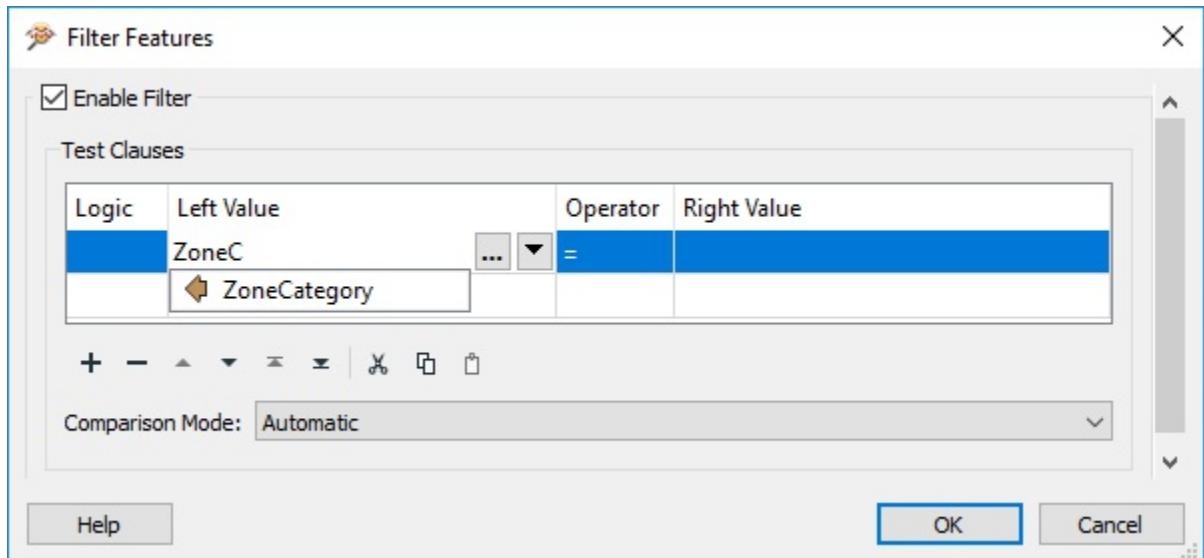
## 8) Filter Data

Let's filter our data to control what is displayed. We'd like to only view industrially-zoned zoning districts.

Click the Visual Preview Graphics window toolbar Filter button:

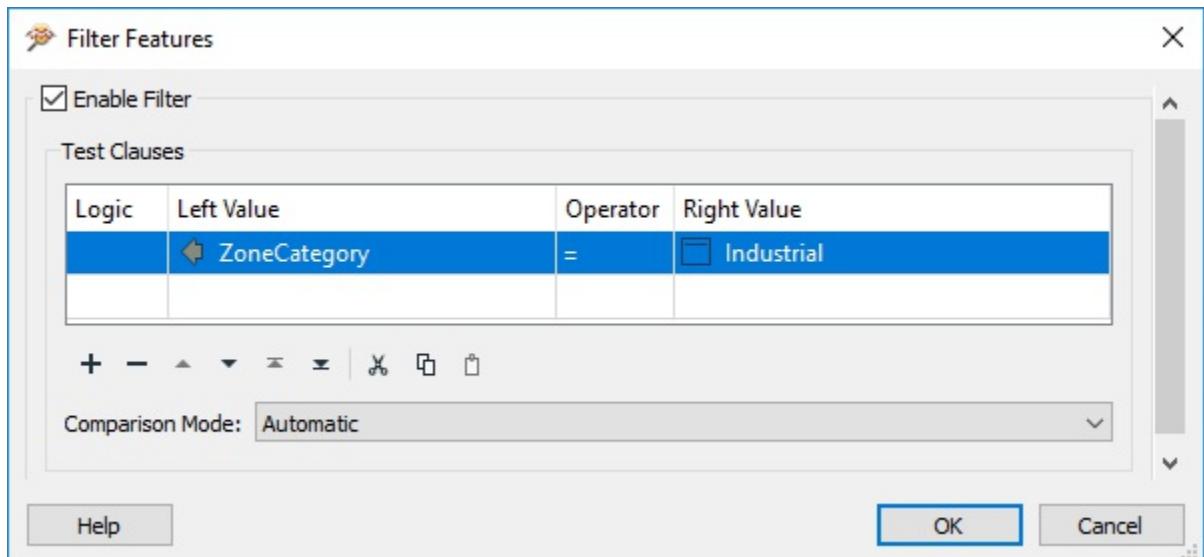


In the Filter Features dialog, check Enable Filter. Then click in the Left Value field and start to type in ZoneCategory. This will search the available attributes:



Once you see ZoneCategory appear in the list, double-click it to select it. Then click in the Operator field and select the = (equals) symbol, if it is not already selected.

For the Right Value field, click in the field and type the word **Industrial** (don't worry, it's not case-sensitive):

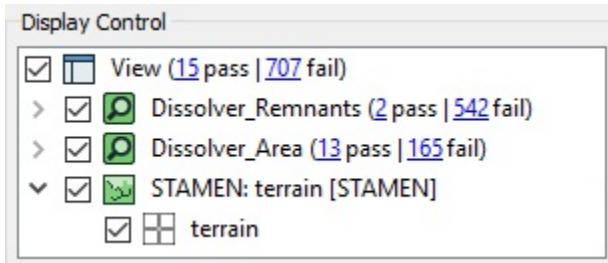


Click OK to close the dialog. The display will be filtered to show only industrial zones:



Map tiles by [Stamen Design](#), under [CC BY 3.0](#). Data by [OpenStreetMap](#), under [CC BY SA](#).

The Display Control Window will also show the effects of the filter:



## CONGRATULATIONS

By completing this exercise you have learned how to:

- Set symbology for inspected features
- Set a background map for inspecting data
- Add a transformer in FME Workbench and set its parameters
- Filter inspected data using test clauses

## Module Review

This module introduced you to FME and investigated the basics of FME data translations.

## What You Should Have Learned from this Module

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

### Theory

- FME is a **data integration tool** to translate and transform data, with an emphasis on spatial data.
- FME Workbench is an application to **graphically define a translation** and the flow of data within it. The definition of a translation is known as a **workspace** and can be saved to a file for later use.
- **Quick Translation** is the technique of carrying out a translation generated by FME, without further editing.
- The **Generate Workspace dialog** is the primary method by which a quick translation can be set up in FME Workbench.
- **Data Inspection** is a technique for checking and verifying data before, during, and after a translation. FME Data Inspector and Visual Preview can both be used for inspecting data.
- **Feature caching** makes it easier to store data for inspection and performance purposes while building workspaces.
- **Partial runs** allow specified sections of a workspace to run in isolation.

### FME Skills

- The ability to open a workspace in FME Workbench and run it
- The ability to start FME Workbench and set up a *quick translation*
- The ability to start the FME Data Inspector, and to open and add datasets
- The ability to navigate a dataset and to select features within it
- The ability to control Data Inspector symbology and display characteristics
- The ability to set background maps in Visual Preview
- The ability to inspect data using Visual Preview
- The ability to run a workspace with feature caching
- The ability to run selected sections of a workspace

## Data Translation Basics Quiz

Each section ends with a quiz to test your new knowledge. Make your selection and click "Check my answers" to check each individual question. If you want an explanation for the answer, click "Explain".

**Note:** your score won't be tallied; this is just for review purposes.

**1)** In [Exercise 1](#), Which library branch has the highest circulation?

- A. Strathcona
- B. Carnegie
- C. Central Branch
- D. Firehall
- E. Joe Fortes
- F. Kitsilano
- G. Mount Pleasant
- H. Accessible Services

**2)** Which of the following statements is true about the Generate Workspace dialog?

- A. You must define both the reader and writer format
- B. You must define the reader format, but writer format is optional
- C. The reader format is optional, but you must define the writer format
- D. Both the reader and writer formats are optional

**3)** Visual Preview and FME Data Inspector are fully-functioned spatial data analysis and cartography tools.

- A. True
- B. False

**4)** Which of the following scenarios would be well-suited to using feature caching? Check all that apply.

- A. Reading from a large database
- B. Reading from a large web dataset
- C. Running a production workspace
- D. Running a simple workspace with a Creator and a single Emailer to send an email
- E. Using partial runs to incrementally develop a workspace with a complicated workflow

**5)** In the Generate Workspace dialog, if you don't set the data format before browsing for the source data, what might happen? Check all that apply.

- A. Generate Workspace will fail
- B. You might not see the file you are looking for, because you are filtering to look for a different file format
- C. Trick question! Workbench will not let you browse for source data before setting a format.
- D. Workbench might guess the wrong format for your data, such as guessing it is a JSON file when it is actually a GeoJSON file.

**6)** Which of these is NOT a way to set the format of a dataset when adding a Reader, Writer, or using Generate Workspace?

- A. Typing the format name
- B. Selecting the format from a drop-down list
- C. Browsing for the format in the Reader and Writer Gallery
- D. By selecting a dataset with a known file extension
- E. None of the above (they are all valid ways to set the format)

## Exercise 5

## Tourist Bureau Project

<b>Data</b>	Community Mapping/Food Vendors (Esri Geodatabase)
<b>Overall Goal</b>	Create a GPS-compatible dataset of food vendors for the local tourist bureau
<b>Demonstrates</b>	Basic Data Translation and Data Inspection
<b>Start Workspace</b>	None
<b>End Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\Basics-Ex5-Complete.fmw

You've barely started in your new job, but requests are coming in fast!

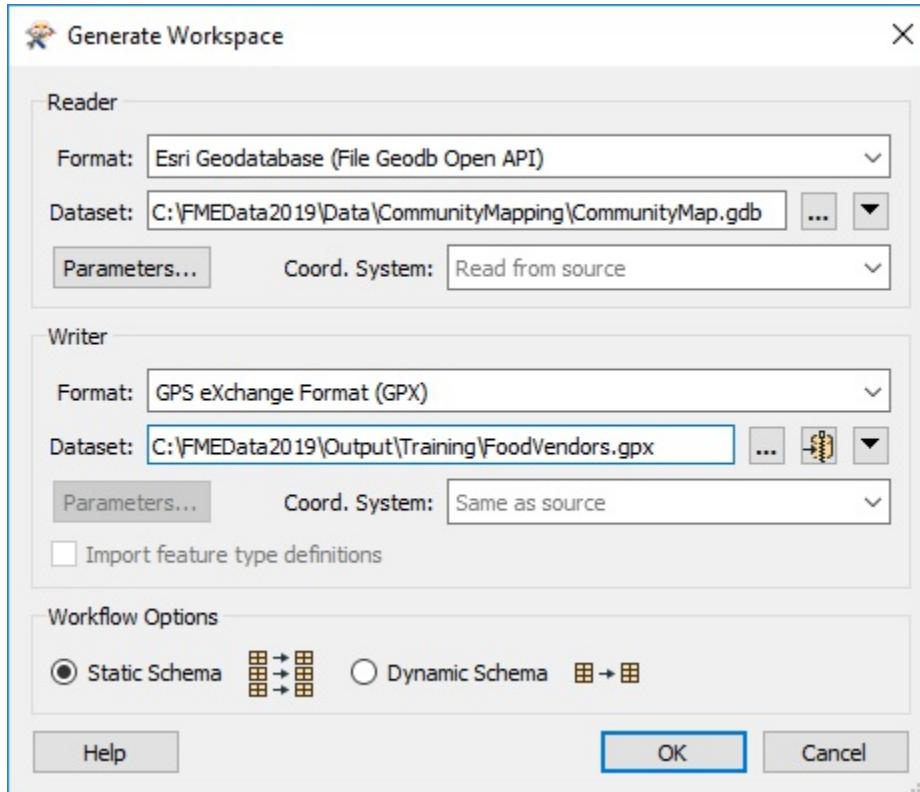
The local tourist bureau is running a promotion where they provide tourists with a GPS device to help them visit street food vendors in the city. Your manager wonders whether you can use FME to produce the data used in this scheme.

Let's get onto that right away shall we?

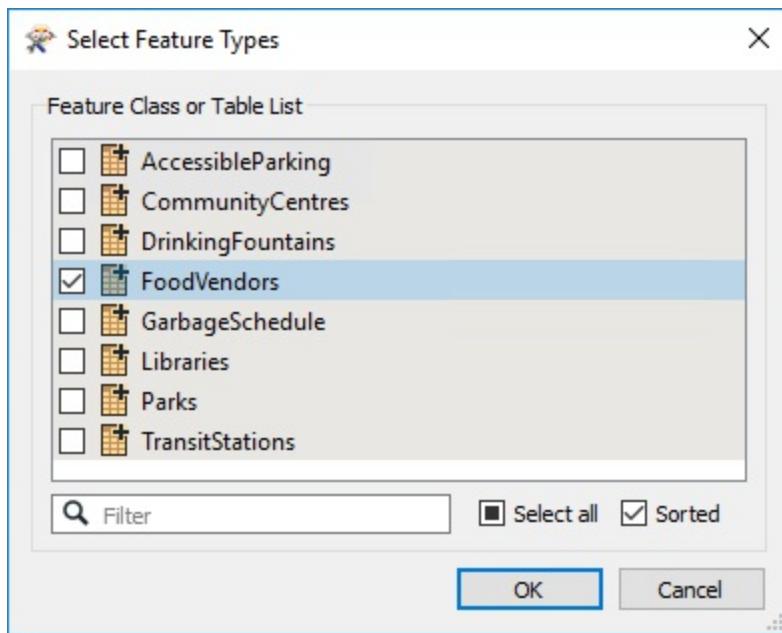
### 1) Start FME Workbench

Start FME Workbench. In the Create Workspace section of the Start window, choose the option to Generate (Workspace). When prompted generate a translation with the following parameters:

<b>Reader Format</b>	Esri Geodatabase (File Geodb Open API)
<b>Reader Dataset</b>	C:\FMEData2019\Data\CommunityMapping\CommunityMap.gdb
<b>Writer Format</b>	GPS eXchange Format (GPX)
<b>Writer Dataset</b>	C:\FMEData2019\Output\Training\FoodVendors.gpx



Click OK to accept the parameters. When prompted which tables to use from the source data (there are several) deselect all tables except for FoodVendors and click OK to create the workspace:



## 2) Connect Reader/Writer

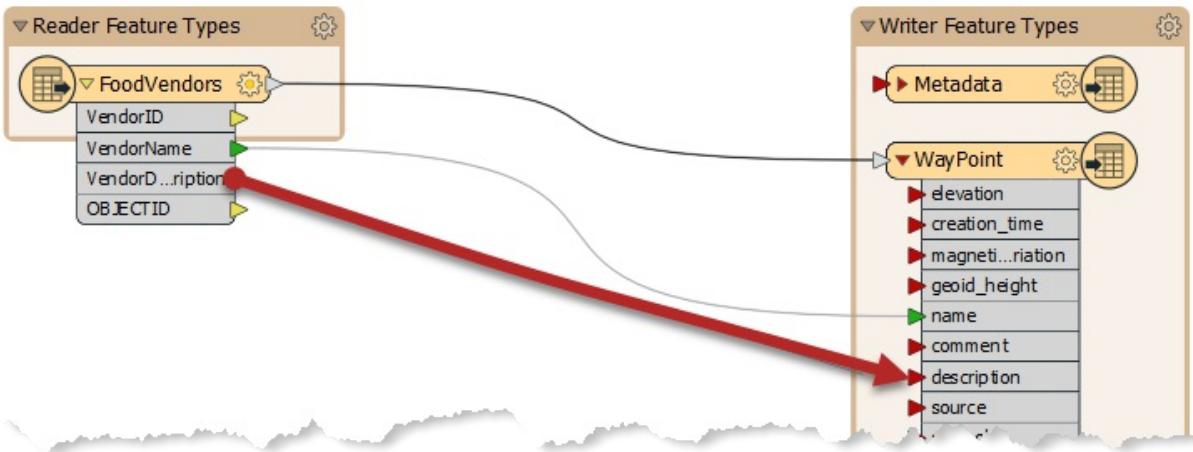
When first created, the reader and writer are not connected in this workspace. Connect them by dragging a connection from the output port of the reader feature type to the input port of the writer feature type labeled WayPoint:



Click the expand buttons on the two objects to expose the list of attributes on each:



Now drag a connection between the Reader attribute *VendorName* and the writer attribute *name*. Repeat the process for *VendorDescription* and *description*:



The technique of connecting objects like this is called schema mapping, and we will learn more about it later.

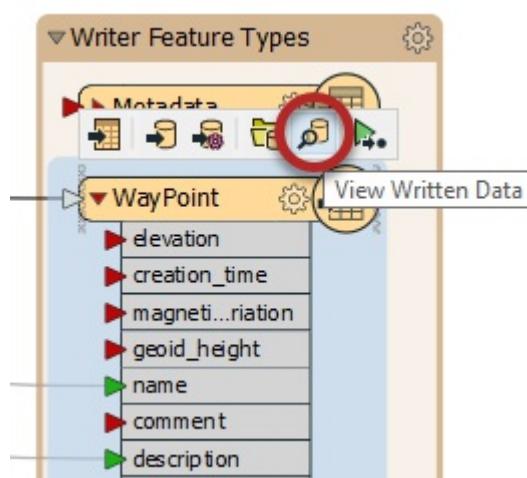
### 3) Run Workspace

Save the workspace, so you have a copy of it, then run the workspace by pressing the green run button. The workspace runs, and the data is written to a Garmin POI dataset:

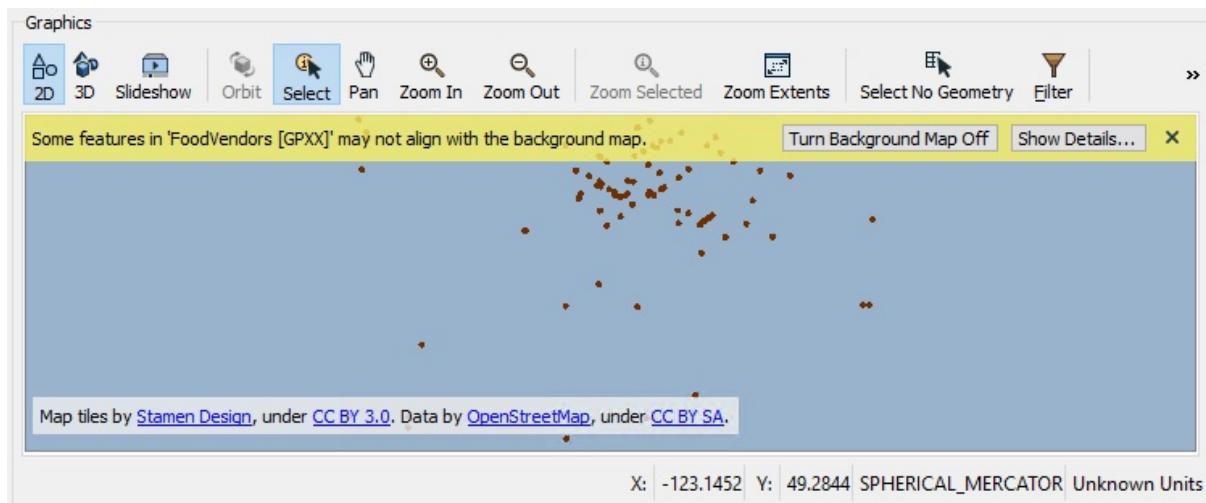
```
=====
Features Read Summary
=====
FoodVendors 91
=====
Total Features Read 91
=====
Features Written Summary
=====
WayPoint 91
=====
Total Features Written 91
=====
Translation was SUCCESSFUL with 0 warning(s) (1 feature(s) output)
FME Session Duration: 1.6 seconds. (CPU: 0.7s user, 0.3s system)
END - ProcessID: 2592, peak process memory usage: 99252 kB, current process memory usage: 89276 kB
Translation was SUCCESSFUL
```

### 4) Inspect Data

Let's try to inspect the data using Visual Preview. Click on the **WayPoint** feature type and then click the View Written Data icon:



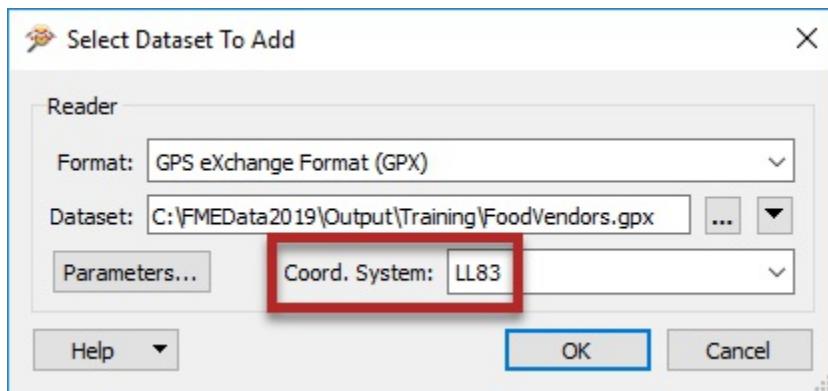
The points will be displayed in Visual Preview, but note that you receive a warning and they appear to be in the wrong place, in the middle of the ocean:



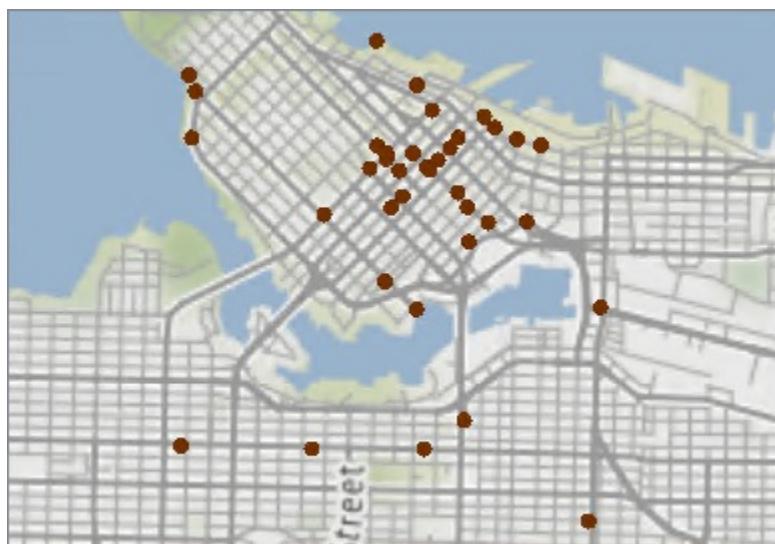
This is because the GPX format does not record its coordinate system inside the dataset. So, to get it to display properly, we have to open it in Data Inspector and specify the coordinate system.

Go to the FME Data Inspector. Select File > Open Dataset from the menu bar. Doing so opens the dialog titled "Select Dataset to View."

Set the format type and select the GPX dataset. You can specify the coordinate system in the bottom-right corner. Type in "LL83" and click OK:



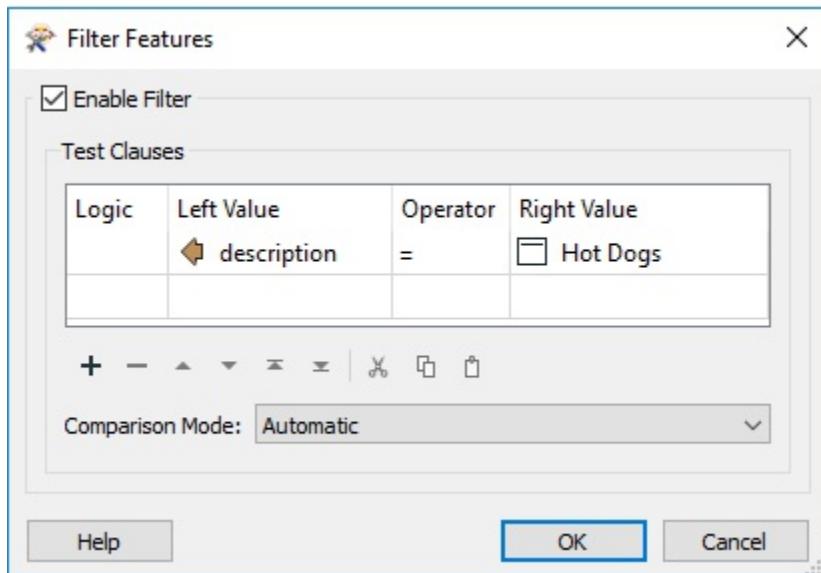
Click OK, and the dataset will be opened for you to verify that it is correct (points should appear in Vancouver, BC):



Map tiles by [Stamen Design](#), under CC-BY-3.0. Data by [OpenStreetMap](#), under CC-BY-SA.

## 5) Filter Data

All this talk of food is making you hungry. It must be lunchtime. To find somewhere to get a quick lunch, filter the data to show hot dog vendors in the city:



You should see in the Navigator that 38 out of 53 food vendors serve hot dogs.

---

## CONGRATULATIONS

*By completing this exercise you proved you know how to:*

- Create and run a workspace in FME Workbench
- Carry out basic 'schema mapping' in FME Workbench
- Open a dataset in the FME Data Inspector
- Filter data using test clauses in the FME Data Inspector

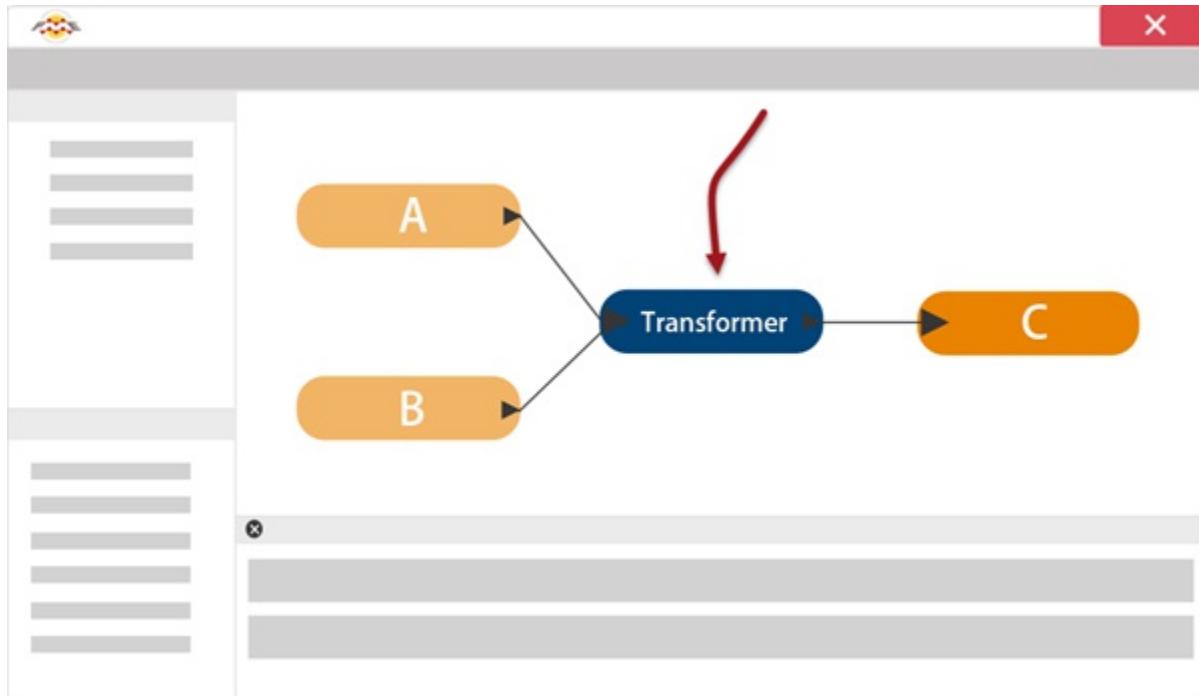
## Data Transformation

**Data transformation** is the ability to manipulate data during a data translation, even to the extent of having an output greater than the sum of the inputs!



## What is Data Transformation?

**Data transformation** is FME's ability to manipulate data. The transformation step occurs during the process of format translation. Data is read, transformed, and then written to the chosen format.



## Data Transformation Types

Data transformation can be subdivided into two distinct types: *structural transformation* and *content transformation*.

### Structural Transformation

Structural transformation is perhaps better called 'reorganization'. It refers to FME's ability to channel data from source to destination in an almost infinite number of arrangements.

This includes the ability to merge data (as in the image above), divide data, re-order data, and define custom data structures.

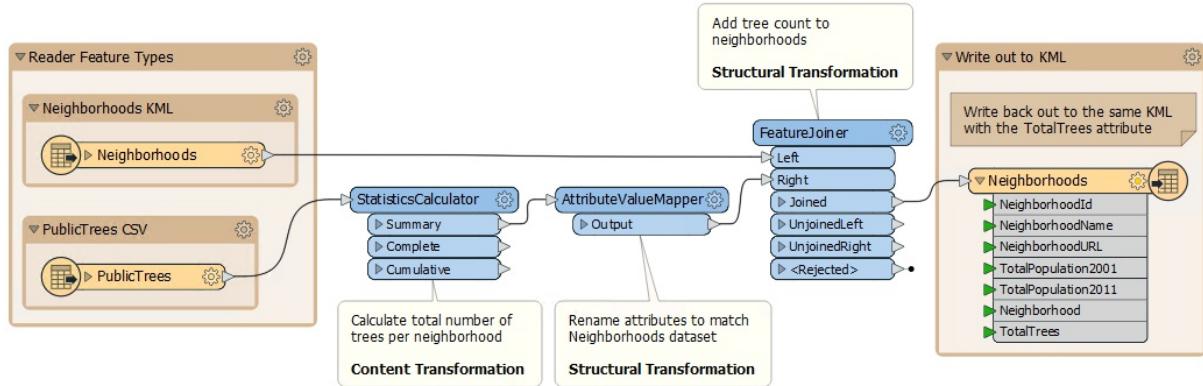
Transforming the structure of a dataset is carried out by manipulating its schema.

### Content Transformation

Content transformation is perhaps better called 'revision.' It refers to the ability to alter the content of a dataset.

Manipulating a feature's geometry or calculating new attribute values is the best example of how FME can transform content.

Content transformation can take place independently or alongside structural transformation:



## Structural Transformation

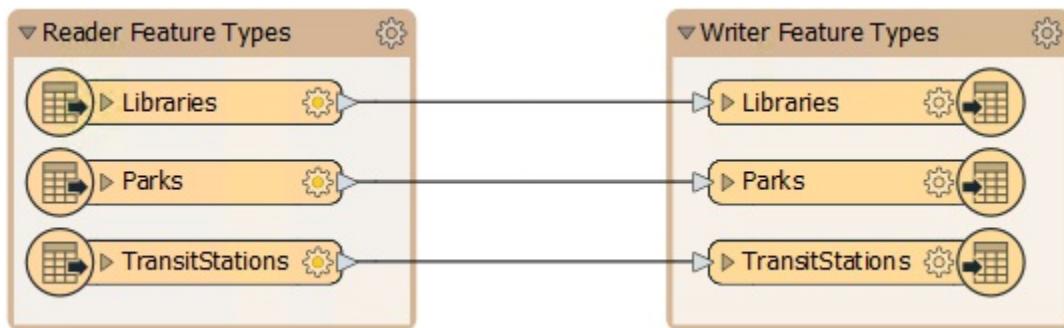
Transforming a dataset's structure requires using FME to manipulate *schemas*. FME uses the term "schema," but you may know this as *data model*.

## Schema Concepts

A **schema** defines the structure of a dataset. Each dataset has its unique schema; it includes layers, attributes, and other rules that define or restrict its content.

### Schema Representation

When a new workspace is created, FME scans the source datasets. It creates a **reader** whose feature types are added to the left side of the workspace canvas and a **writer** whose feature types are added to the right side of the workspace canvas:



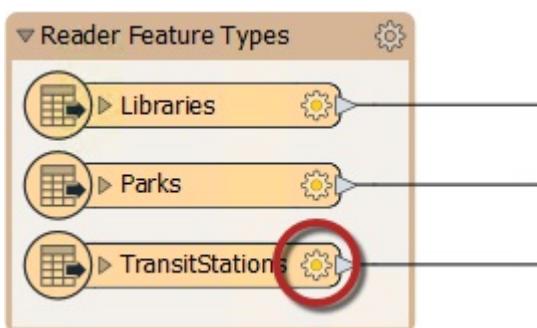
### TIP

Each object in this illustration represents a subdivision in the source dataset. In FME terminology these objects are called **feature types**. FME supports 400+ formats and there are almost as many terms for the way data is subdivided. The most common terms are layer, table, class, category, level, or object.

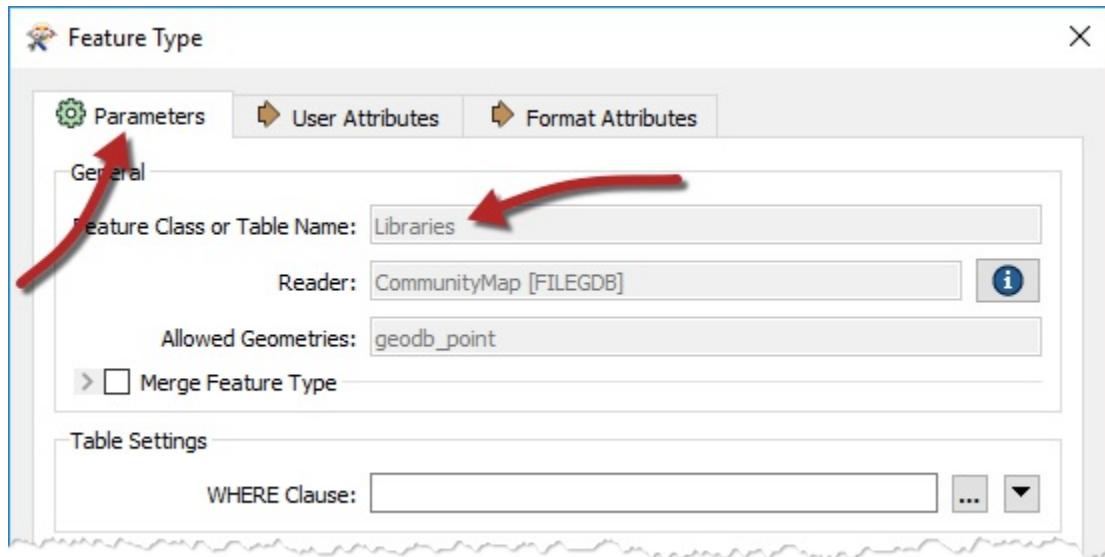
Although the general FME term for these subdivisions is **feature type**, all dialogs in FME Workbench use format-specific terminology where the correct term is applicable.

## Reader Schema

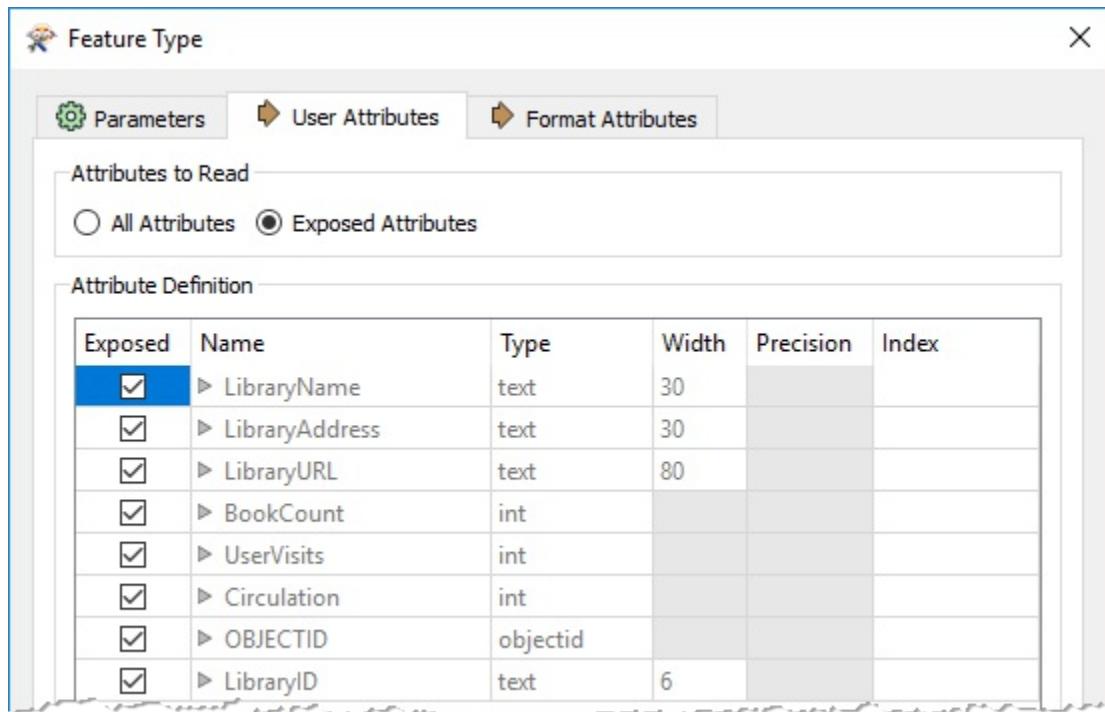
For the reader, more information about the schema is revealed by clicking the cog-wheel icon on each feature type object:



This Feature Type dialog has a number of tabs. Under the Parameters tab is a set of general parameters, such as the name of the feature type (in this case Libraries) the allowed geometry types, and the name of the parent dataset:



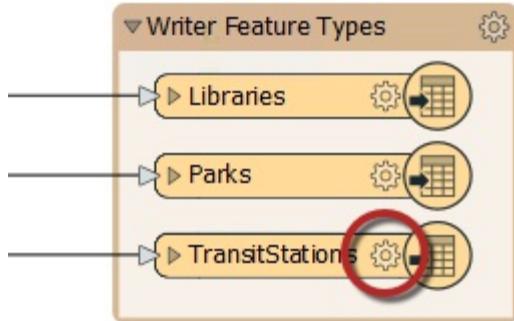
The User Attributes tab shows a list of attributes. Each attribute is defined by its name, data type, width, and number of decimal places:



Each feature type has a different name and can also have a completely different set of attributes. All of this information goes to make up the reader schema. It is literally "**what we have**".

## Writer Schema

As with the reader, each writer has a set of detailed schema information accessed by opening the dialog for a feature type:



By default, the writer schema ("**what we want**") is a mirror image of the source, so the output from the translation will be a duplicate of the input. This feature allows users to translate from format to format without further edits (*Quick Translation*).

If "*what we want*" is different from the default schema definition, we have to change it using a technique called **Schema Editing**.

## Schema Editing

**Schema editing** is the process of altering the writer schema to customize the structure of the output data. One good example is renaming an attribute field.

After editing, the source schema still represents "*what we have*," but the destination schema now truly does represent "*what we want*."

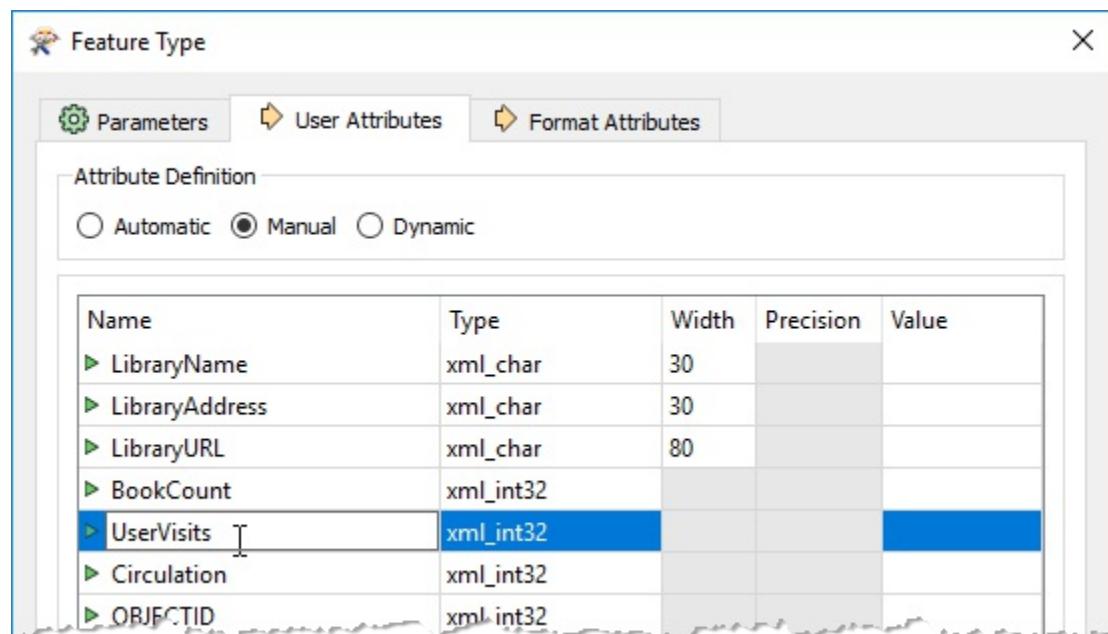
## Editable Components

There are a number of edits that can be performed, including, but not limited to the following:

### Attribute Renaming

Attributes on the destination schema can be renamed, such as renaming POSTALCODE to PSTLCODE.

To rename an attribute open the Feature Type dialog and click the User Attributes tab. Click on the attribute to be renamed and enter the new name:



### Attribute Type Changes

Any attribute on the writer schema can have a change of type; for example, changing from an integer field to a character field.

To change an attribute type open the User Attributes tab and set the Type field to the new type:

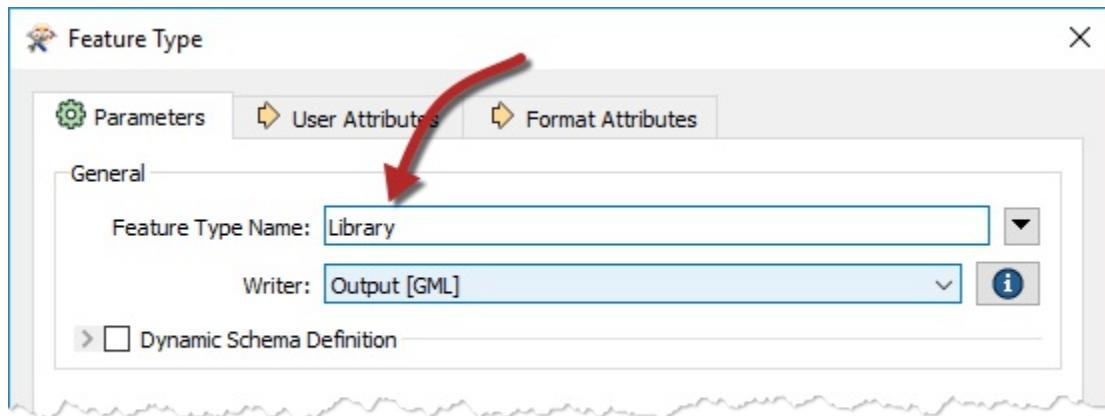
Name	Type	Width	Precision	Value
▶ LibraryName	xml_char	30		
▶ LibraryAddress	xml_char	30		
▶ LibraryURL	xml_char	80		
▶ BookCount	xml_int32			
▶ UserVisits	xml_boolean			
▶ Circulation	xml_buffer			
▶ OBJECTID	xml_byte			
▶ LibraryID	xml_char			
▶	xml_date			
	xml_datetime			
	xml_decimal			
	xml_geometry			
	xml_int16			

### TIP

The Type column for an attribute shows only values that match the permitted types for that data format. For example, an Oracle schema permits attribute types of varchar or clob. MapInfo does not support these data types so they would never appear in a MapInfo schema. The above screenshot shows data types for the GML format.

### Feature Type Renaming

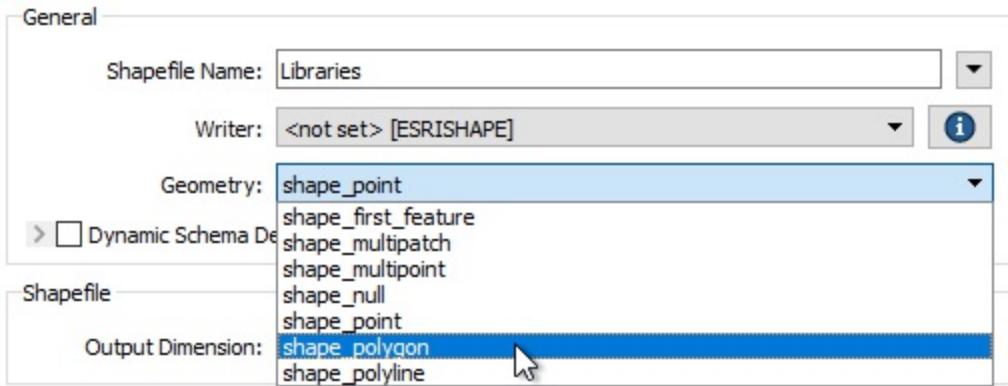
To rename a writer feature type, open the Parameters tab, click in the Feature Type Name field (the label may vary according to the format type), and edit the name as required:



You can also rename a feature type by clicking on it in the Workbench canvas and pressing the F2 key.

### Geometry Type Changes

To change the permitted geometry for a feature type (for example, change the permitted geometries from lines to points), open the Parameters tab and change the permitted geometry type:



This field is only available where the format requires a strict decision on geometry type. Where formats allow any mix of geometry in a single feature type, this setting is not shown.

Once the user has made all the required changes to the writer schema, the workspace reflects "*what we want*" - but it doesn't yet specify how the reader and writer schemas should be connected together. This is the next task and is called **Schema Mapping**.

### TIP

*You might be wondering "what would happen if I edited the **reader** schema, instead of the writer"?*

*Well, by default you can't! The schema fields for a reader are greyed out to prevent this, since changes would put the schema definition out of sync with the source dataset:*



*There are a few, rare, use cases - but they're outside the scope of this training course!*

## Schema Mapping

Schema mapping is the second key part of data restructuring in FME.

In FME Workbench, one side of the workspace shows the reader schema ("what we have") and the other side shows the writer schema ("what we want"). Initially, the two schemas are automatically joined when the workspace is created, but when edits occur these connections are usually lost.

**Schema Mapping** is the process of connecting the reader schema to the writer schema in a way that ensures the correct reader features are sent to the correct writer feature types and the correct reader attributes are sent to the correct writer attributes.

FME permits mapping from source to destination in any arrangement desired. There are no restrictions on what feature types or attributes may be mapped.

### FME Lizard says...

*I think of schema editing and schema mapping as a means of reorganizing data.*

*A good analogy is a wardrobe full of clothes. When the wardrobe is reorganized, you throw out what you no longer need, reserve space for new clothes that you're planning to get, and move existing items into a more usable arrangement.*

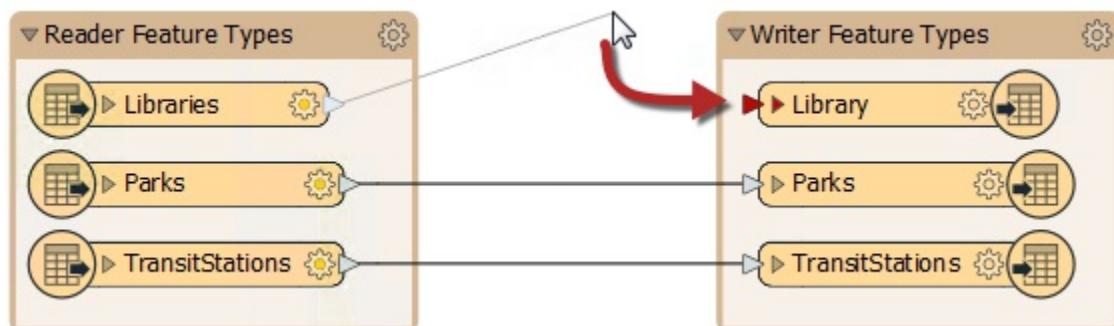
*The same holds true for spatial data restructuring: it's the act of reorganizing data to make it more usable.*

## Feature Mapping

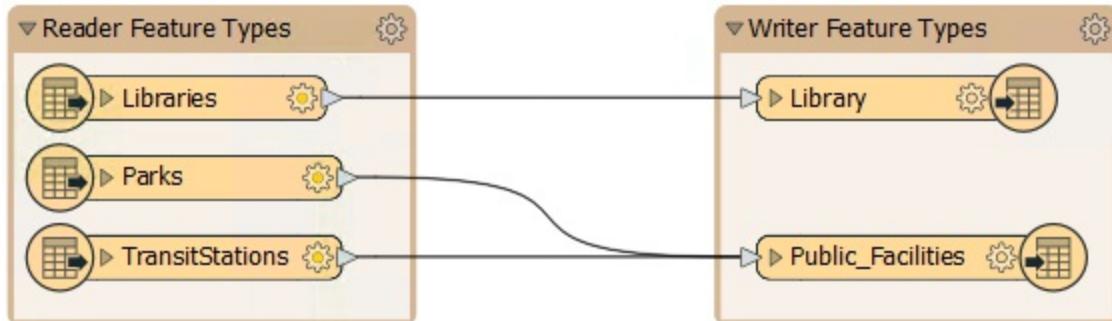
**Feature mapping** is the process of connecting source feature types to destination feature types.

Feature mapping is carried out by clicking the output port of a reader feature type, dragging the arrowhead to the input port of a writer feature type, and releasing the mouse button. A thick, black line represents connections.

Here, a connecting line from the source to the destination feature type is created by dragging the arrowhead from the source to the destination:



Merging and splitting of data is permitted:



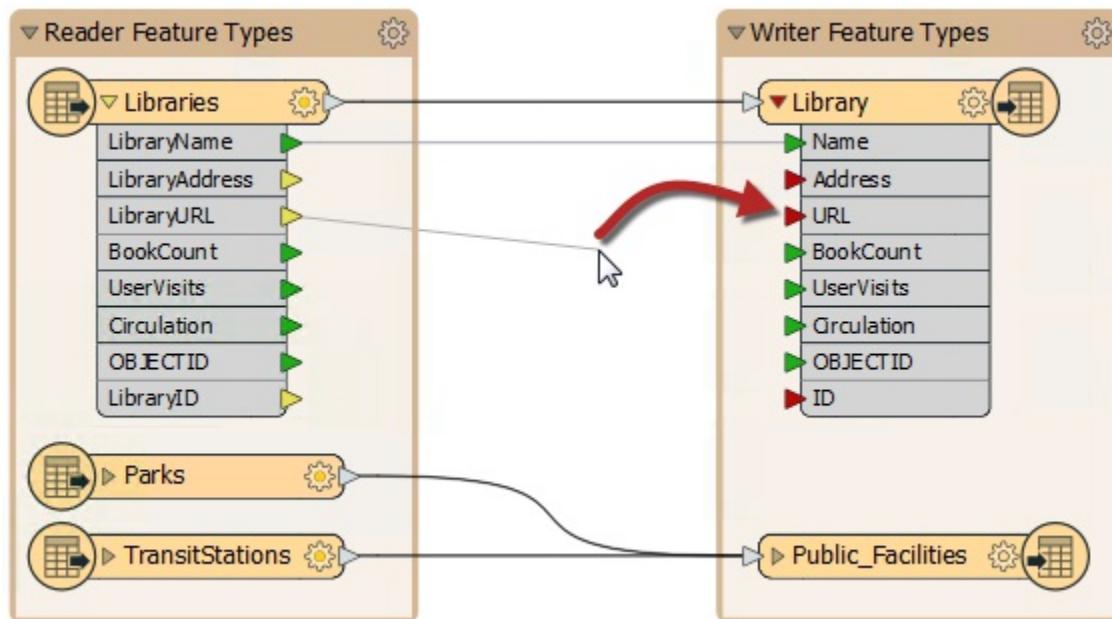
Here a user requires both Parks, and TransitStations to be on a single layer in the output and so merges those two reader feature types into one writer feature type (Public\_Facilities).

## Attribute Mapping

**Attribute mapping** is the process of connecting reader attributes to writer attributes.

Attribute mapping is performed by clicking the output port of a reader attribute, dragging the arrowhead to the input port of a writer attribute, and releasing the mouse button. Attribute connections are shown with a thinner, gray line.

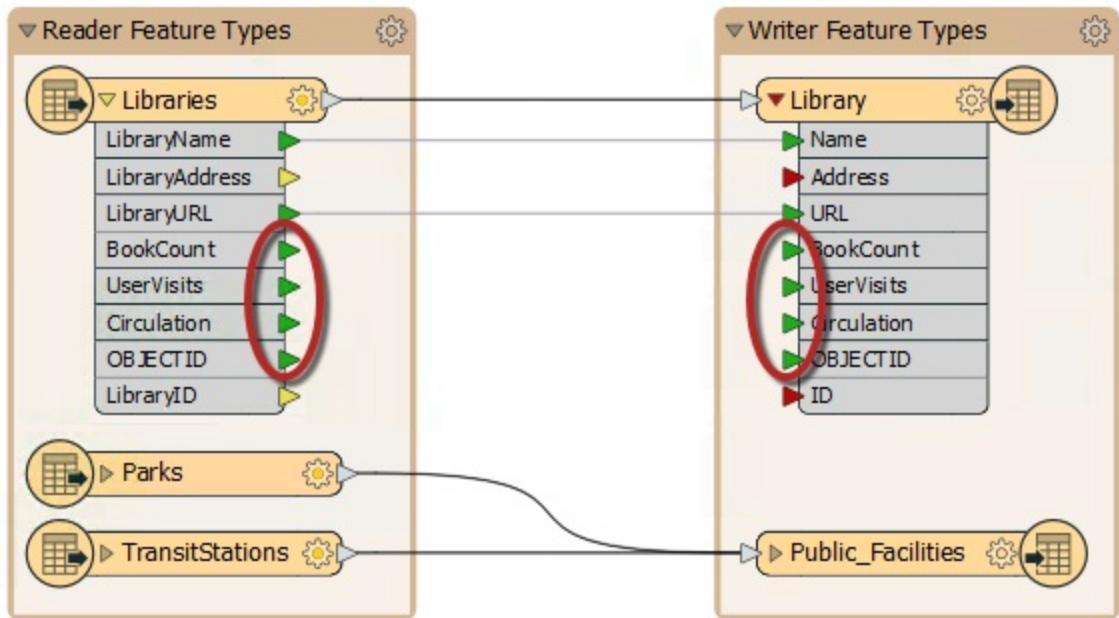
Here feature mapping has been carried out already and attribute connections are being made:



Notice the green, yellow, and red color-coding that indicates which attributes are connected.

Green ports indicate a connected attribute. Yellow ports indicate a reader attribute that's unconnected to a writer. Red ports indicate a writer attribute that's unconnected to a reader.

Attributes with the same name in reader and writer feature types are connected automatically, even though a connecting line might not be visible; the port color is the key:



## WARNING

*Names are case-sensitive, therefore ROADS is not the same as roads, Roads, or roads.  
That's important to know if you are relying on automatic attribute connections!*

## Exercise 1

## Grounds Maintenance Project - Schema Editing

<b>Data</b>	City Parks (MapInfo TAB)
<b>Overall Goal</b>	Calculate the size and average size of each park in the city, to use in Grounds Maintenance estimates for grass cutting, hedge trimming, etc.
<b>Demonstrates</b>	Structural Transformation, Schema Editing
<b>Start Workspace</b>	None
<b>End Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex1-Complete.fmw

You have just landed a job as a technical analyst in the GIS department of your local city.

The team responsible for maintaining parks and other grassed areas needs to know the area and facilities of each park in order to plan their budget for the upcoming year. You have been assigned to this project and will use FME to provide a dataset of this information.

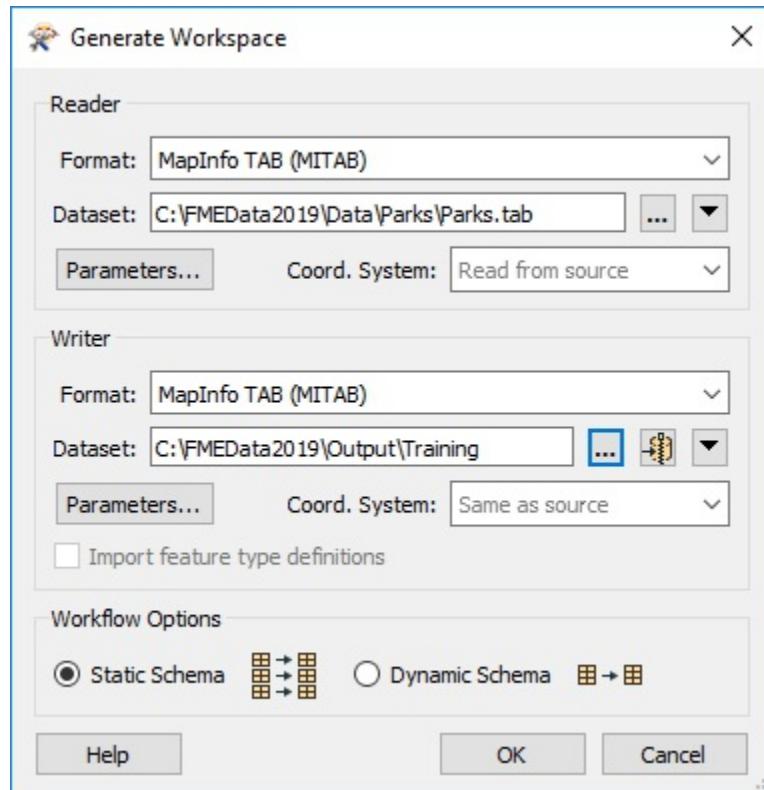
The first step in this example is to rename existing attributes and create new ones in preparation for the later area calculations.

### 1) Start FME Workbench

Start FME Workbench, then use the Generate Workspace dialog to create a workspace using these parameters:

<b>Reader Format</b>	MapInfo TAB (MITAB)
<b>Reader Dataset</b>	C:\FMEData2019\Data\Parks\Parks.tab
<b>Writer Format</b>	MapInfo TAB (MITAB)
<b>Writer Dataset</b>	C:\FMEData2019\Output\Training

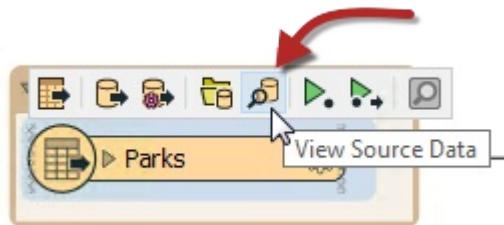
Yes! Here we write back to the same format of data we are reading from! You may click the Parameters buttons in the Generate Workspace dialog to check the reader/writer parameters, but none of them need changing in this exercise. Note that the writer needs only a folder and not a specific file name.



## 2) View Data

Before we can begin to manipulate our data, we should first look at it and see what we are working with.

Click on the Parks reader feature type to bring up the popup menu. Then from the popup menu, click on the View Source Data button:



The source data will then open up in the Visual Preview window. In this exercise we are interested in setting up the writer to have the attribute names we want as well as assigning the data the correct data type. Becoming familiar with how the output data is structured is important. For the MapInfo (MITAB) format, these are the data types:

- **char:** A string character, no math operations can be performed
- **float:** A number with a decimal value
- **integer:** A whole number, no decimals (32-bits)
- **smallint:** A whole number, no decimals (16-bits)

There are other data types that are covered in greater detail in the [MapInfo TAB \(MITAB\) documentation](#). For our dataset we have some character data (3-4, 7-9), integer data (5) and small integer data (1-2, 6):

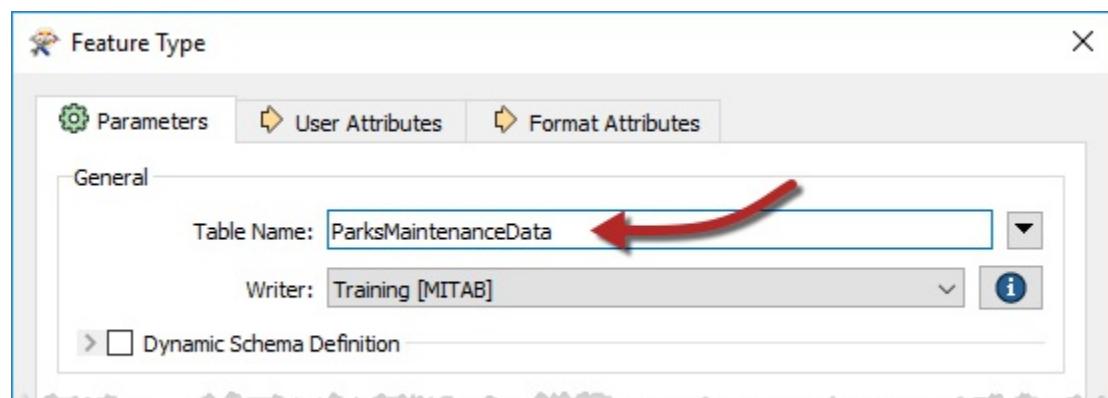
ParkId	RefParkId	ParkName	NeighborhoodName	VisitorCount	TreeCount	DogPark	Washrooms	SpecialFeatures
1	1	-9999 <missing>	Kitsilano	9406	10 N	<missing>	N	<missing>
2	2	208 Rosemary Brow...	Kitsilano	13100	8 N	N	N	N
3	3	141 Tea Swamp Park	Mount Pleasant	11275	2 N	N	N	N
4	4	-9999 <missing>	Strathcona	9755	6 N	<missing>	N	<missing>
5	5	202 Morton Park	West End	14977	4 N	N	N	N
6	6	-9999 McBride Park	Kitsilano	15053	9 N	<missing>	N	<missing>

We will need to make sure that the writer is assigning these data types correctly as well we want to remove the extra attribute we do not need.

## 3) Rename Feature Type

FME creates a workspace where the destination schema matches the source. However, the end user of the data has requested parts of the schema are cleaned up.

Inspect the writer feature type parameters. Click in the field labeled Table Name (remember this label is format-specific and in MapInfo we deal with "tables") and change the name from Parks to ParksMaintenanceData:



#### 4) Update Attributes

Now inspect the user attributes. They will look like this:

The screenshot shows the 'User Attributes' tab of the Attribute Definition dialog. At the top, there are three radio button options: 'Automatic' (unchecked), 'Manual' (checked), and 'Dynamic' (unchecked). Below this is a table with columns: Name, Type, Width, Precision, Value, and Index. The table contains the following data:

Name	Type	Width	Precision	Value	Index
ParkId	smallint				
RefParkId	smallint				
ParkName	char	40			
NeighborhoodName	char	40			
VisitorCount	smallint				
TreeCount	smallint				
DogPark	char	1			
Washrooms	char	1			
SpecialFeatures	char	1			
▶					

At the bottom of the dialog are several icons: a plus sign (+), a minus sign (-), a double minus sign (⊖), a double plus sign (⊕), a double arrow (↔), a double arrow with a circle (⟳), and a double arrow with a square (⟲). To the right of these icons is a search bar labeled 'Filter'.

These must be cleaned up so that unnecessary information is removed. Other attributes need to be updated and some extra ones added to store the calculation results. So carry out the following actions:

Delete Attribute	RefParkID	
Delete Attribute	DogPark	
Delete Attribute	Washrooms	
Delete Attribute	SpecialFeatures	
Rename Attribute	from: NeighborhoodName	to: Neighborhood
Change Type (VisitorCount)	from: smallint	to: integer
Add Attribute	ParkArea	type: Float
Add Attribute	AverageParkArea	type: Float

The attribute list should now look like this:

Parameters    User Attributes    Format Attributes

Attribute Definition

Automatic  Manual  Dynamic

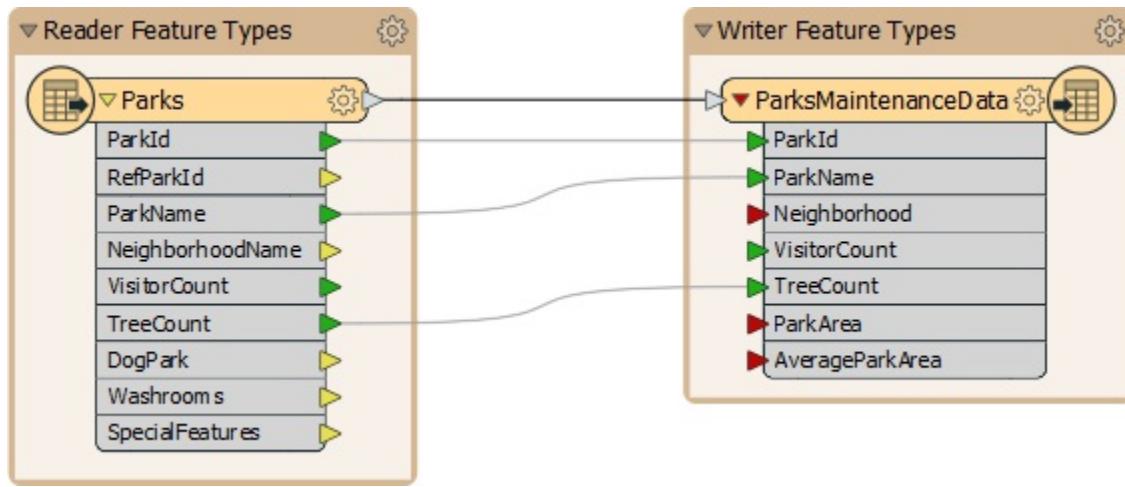
Name	Type	Width	Precision	Value	Index
ParkId	smallint				
ParkName	char	40			
Neighborhood	char	40			
VisitorCount	integer				
TreeCount	smallint				
ParkArea	float				
AverageParkArea	float				
>					

+ - ▲ ▼ = = ⌂ ⌃ ⌄ ⌅ Filter

Now when the workspace is run the output will be named ParksMaintenanceData.tab and will contain an updated attribute schema.

## 5) Un-Expose Source Attributes

The workspace will now look like this:



### NEW

New for FME 2019.0: objects on the canvas will automatically be resized (as in the above screenshot) to fit feature type names.

Notice there are several source attributes that are not going to be used in the workspace or sent to the output. We can tidy the workspace by hiding these.

Inspect the User Attributes tab on the reader feature type parameters. It will look like this:

The screenshot shows the 'User Attributes' tab of the Attribute Definition dialog. It contains a table with columns: Exposed, Name, Type, Width, Precision, and Index. The 'Exposed' column has checkboxes for each row. The 'Name' column lists attributes: ParkId, RefParkId, ParkName, NeighborhoodName, VisitorCount, TreeCount, DogPark, Washrooms, and SpecialFeatures. The 'Type' column shows types: smallint, smallint, char, char, smallint, smallint, char, char, and char respectively. The 'Width' column has values 40, 40, 1, 1, and 1. The 'Precision' and 'Index' columns are empty. At the bottom are 'Filter' and 'Select All' buttons.

Exposed	Name	Type	Width	Precision	Index
<input checked="" type="checkbox"/>	► ParkId	smallint			
<input checked="" type="checkbox"/>	► RefParkId	smallint			
<input checked="" type="checkbox"/>	► ParkName	char	40		
<input checked="" type="checkbox"/>	► NeighborhoodName	char	40		
<input checked="" type="checkbox"/>	► VisitorCount	smallint			
<input checked="" type="checkbox"/>	► TreeCount	smallint			
<input checked="" type="checkbox"/>	► DogPark	char	1		
<input checked="" type="checkbox"/>	► Washrooms	char	1		
<input checked="" type="checkbox"/>	► SpecialFeatures	char	1		

Uncheck the checkbox for the following attributes, which we do not need:

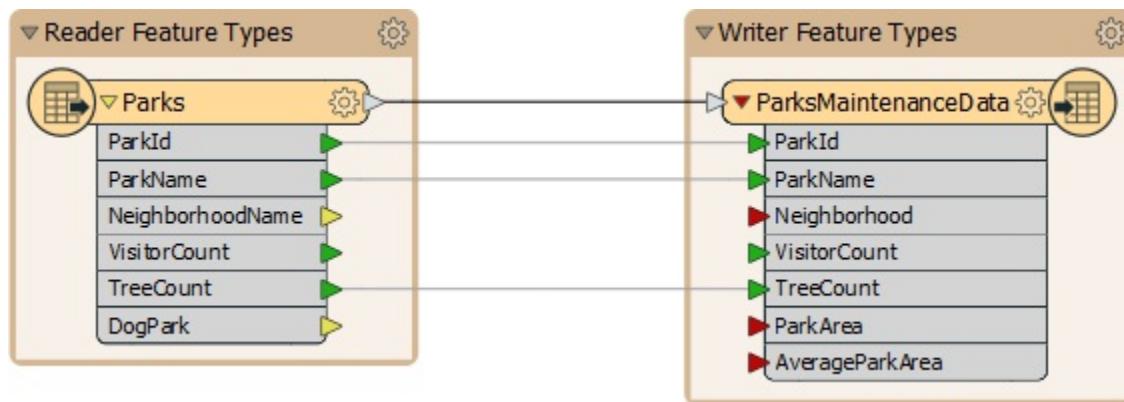
- RefParkID
- Washrooms
- SpecialFeatures

This is the list of attributes we deleted, except for DogParks, which we will make use of in the translation.

Click Apply/OK to confirm the changes.

## 6) Save the Workspace

Save the workspace – it will be completed in further examples. It should now look like this:



### FME Lizard says...

*Some Writer attributes have red connection arrows because there is nothing to map them to yet (ParkArea or AverageParkArea) or the name has changed (Neighborhood). While VisitorCount is green but doesn't show a connection line, this is because the data is still the same yet the output data type has changed.*

*You can still run this workspace just to see what the output looks like anyway.*

---

## CONGRATULATIONS

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

- *Edit the attributes on a writer schema*
- *Edit the output feature type name on a writer schema*
- *Hide attributes on a reader schema*

## Transformation with Transformers

Besides Schema Editing and Schema Mapping, transformation can be carried out using objects called *transformers*.

### What is a Transformer?

As the name suggests, a transformer is an FME Workbench object that carries out the transformation of features. There are lots of FME transformers, each of which carries out many different operations.

Transformers are connected somewhere between the reader and writer feature types, so that data flows from the reader, through a transformation process, and on to the writer.

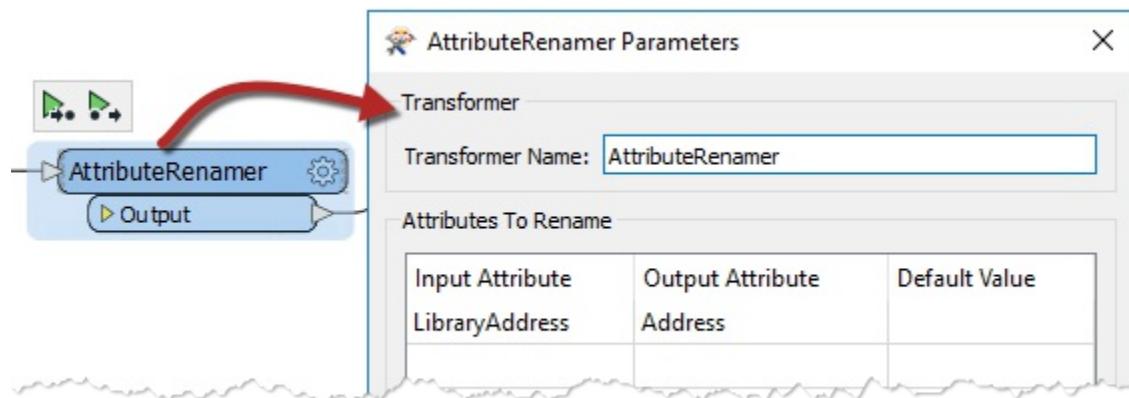
Transformers usually appear in the canvas window as rectangular, light-blue objects.

#### TIP

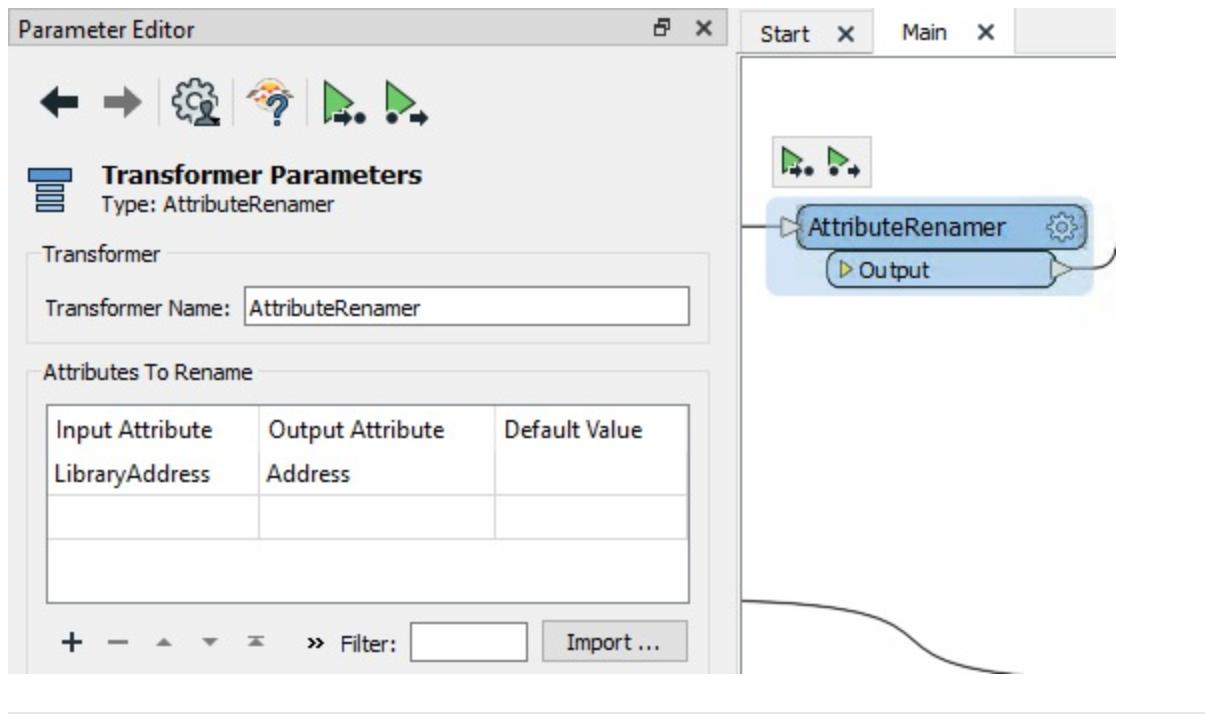
To add a transformer, just start typing on the canvas. For example, to find an AttributeManager you would start typing "AttributeMan" and the quick add will appear. There will be more information on this in Chapter 4: Transformers.

## Transformer Parameters

Each transformer may have a number of parameters (settings). Parameters can be accessed (like feature types) by clicking the cogwheel icon:



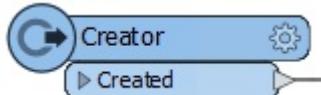
Alternatively, if the Parameter Editor window is open, parameters can be found there simply by clicking on the transformer (or any other canvas object):



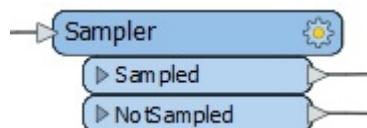
## Color-Coded Parameter Buttons

The parameter button on a transformer is color-coded to reflect the status of the settings.

A blue parameter button indicates that the transformer parameters have been checked and amended as required and that the transformer is ready to use.



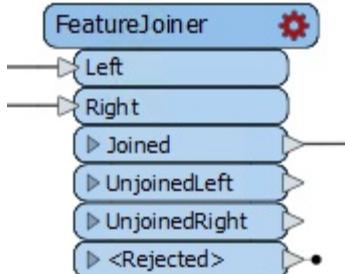
A yellow parameter button indicates that the default parameters have not yet been checked. The transformer can be used in this state, but the results may be unpredictable.



### TIP

*If the Parameter Editor window is open then you will rarely see a yellow icon, because a transformer's parameters are automatically opened and assumed to be reviewed. If that window is open, you should be sure to check it to ensure you don't miss setting a parameter that you need.*

A red parameter button indicates that there is at least one parameter for which FME cannot supply a default value. The parameter must be provided with a value before using the transformer.



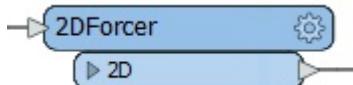
## FME Lizard says...

*Please be sure to check your parameters before you try your translation. Your workspaces just won't work if there are any red-flagged transformers in them!*

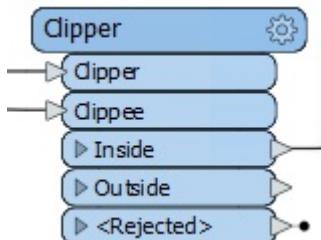
## Transformer Ports

Far from having just a single input and output, a transformer can have multiple input ports, multiple output ports, or both.

This 2DForcer transformer has a single input and output port.



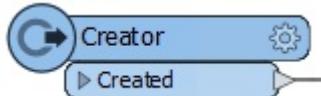
This Clipper has multiple input and output ports. Notice that not all of the output ports are – or need to be – connected.



This Terminator has just a single input port...

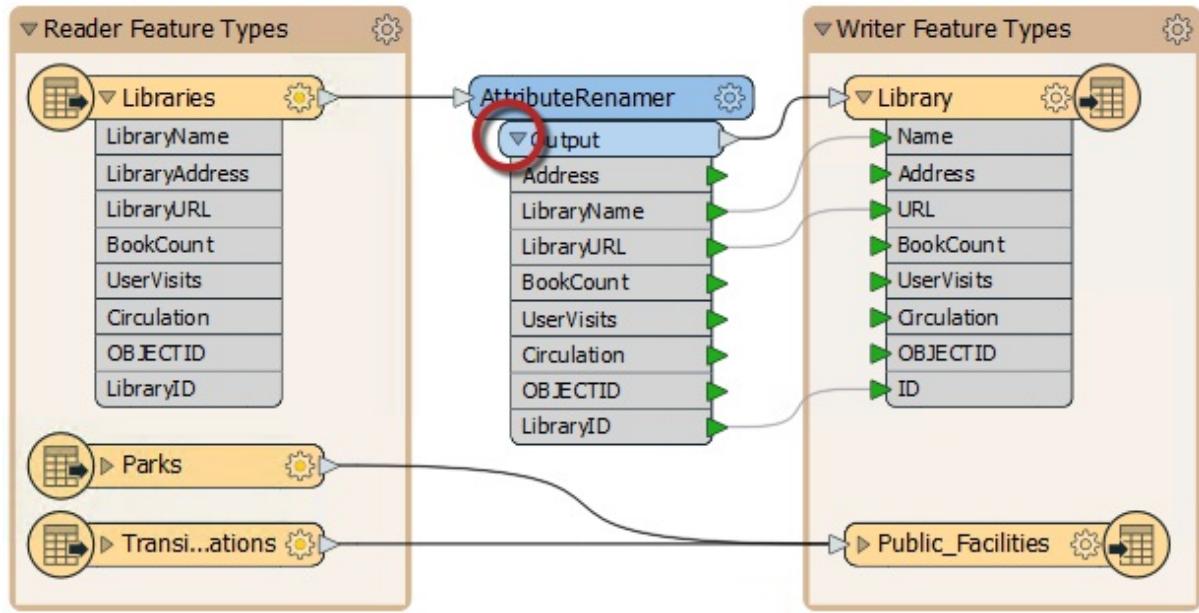


...whereas this Creator has only a single output port!



## Transformer Attributes

Click on the drop-down arrow of a transformer output port to see all of the attributes that exit the transformer. This list includes all changes applied within the transformer.



This feature lets one visualize which attributes have been created, lost, or otherwise transformed within the transformer.

## Exercise 2

# Grounds Maintenance Project - Structural Transformation

<b>Data</b>	City Parks (MapInfo TAB)
<b>Overall Goal</b>	Calculate the size and average size of each park in the city, to use in Grounds Maintenance estimates for grass cutting, hedge trimming, etc.
<b>Demonstrates</b>	Structural Transformation with Transformers
<b>Start Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex2-Begin.fmw
<b>End Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex2-Complete.fmw C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex2-Complete-Advanced.fmw

Let's continue your work on the grounds maintenance project.

In case you forgot, the team responsible for maintaining parks and other grassed areas needs to know the area and facilities of each park to plan their budget for the upcoming year.

In this part of the project, we'll filter out dog parks from the source data (as these have a different scale of maintenance costs) and write them to the log window. We'll also handle the renamed attribute NeighborhoodName.

### 1) Start FME Workbench

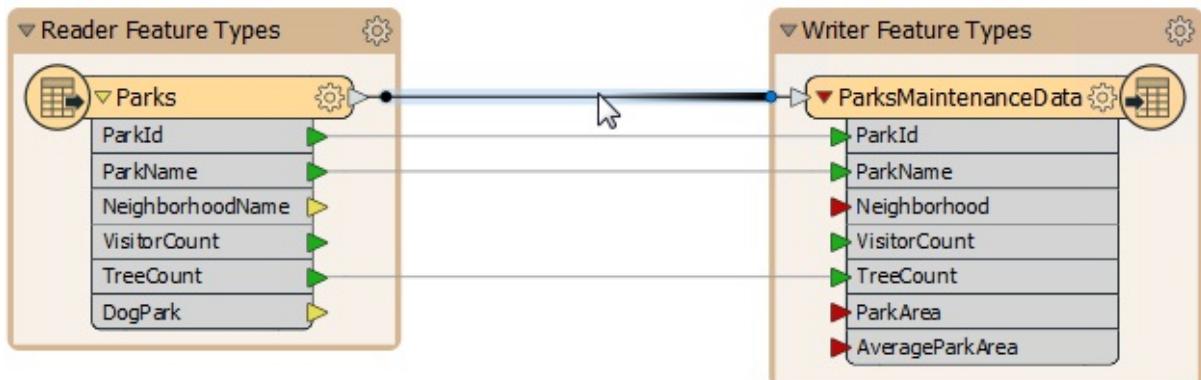
Start FME Workbench and open the workspace from the previous exercise. Alternatively, you can open C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex2-Begin.fmw.

### 2) Add an AttributeManager Transformer

Let's first handle the source attribute NeighborhoodName, which was renamed Neighborhood on the writer but not yet connected.

We could do this by simply drawing a connection, but it's generally better to use a transformer. There are two transformers we could use. One is called the AttributeRenamer and the other - which we shall use here - is the AttributeManager.

Click on the feature connection from reader to writer feature type:



Start to type the phrase "AttributeManager," which will bring up the Quick Add Menu. This method is how we can find and place transformers in the workspace. As you type, FME searches for a matching transformer. When the list is short enough for you to see the AttributeManager, select it from the dialog (double-click on it):

attributem

**FME Transformers**

- AttributeManager** (highlighted)
- NullAttributeMapper
- BulkAttributeMapper

**Custom Transfomers**

**FME Hub Transformers**

**Readers**

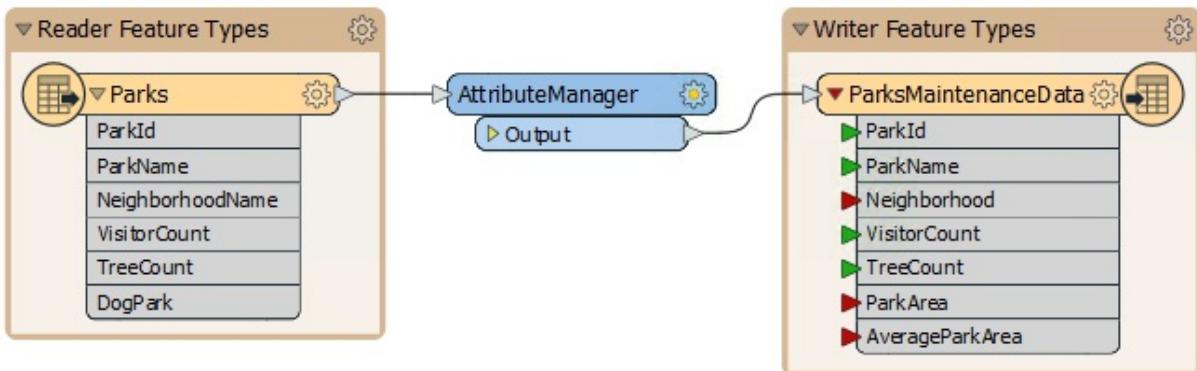
**Writers**

**AttributeManager**

Alters multiple attributes through adding, renaming, copying, deleting and re-ordering. Sets values for new, existing, and modified attributes to any combination of constants, attribute values, conditionals, expressions, and parameters.

[Browse Additional Help ...](#)

Doing so will place an AttributeManager transformer:



### TIP

For a great tip on adding transformers see #5 in our list of [The Top Ten FME Tips of All Time!](#)

### 3) Set the AttributeManager Parameters

Double-click on the AttributeManager to open its parameters. It will look like this:

Transformer

Transformer Name: AttributeManager

> Advanced: Attribute Value Handling

Attribute Actions

Input Attribute	Output Attribute	Attribute Value	Action
ParkId	ParkId		Do Nothing
ParkName	ParkName		Do Nothing
NeighborhoodName	NeighborhoodName		Do Nothing
VisitorCount	VisitorCount		Do Nothing
TreeCount	TreeCount		Do Nothing
DogPark	DogPark		Do Nothing
<Add new Attribute>			

+ - ▲ ▼ ⌂ ⌃ Filter: | Import ... C ▾

Notice that all of the attributes on the stream in which it is connected will automatically appear in the dialog.

Where the Input Attribute field is set to NeighborhoodName, click in the Output Attribute field. Click on the button for the drop-down list and in there choose Neighborhood as the new attribute name to use:

Attribute Actions

Input Attribute	Output Attribute	Attribute Value	Action
ParkId	ParkId		Do Nothing
ParkName	ParkName		Do Nothing
NeighborhoodName	NeighborhoodName	<input type="button" value="..."/>	Do Nothing
VisitorCount	mapinfo_text_spacing		Do Nothing
TreeCount	mapinfo_text_string		Do Nothing
DogPark	mapinfo_text_width		Do Nothing
	mapinfo_type		Do Nothing
	Neighborhood	<input type="button" value="..."/>	
	ParkArea		
	ParkId		
	ParkName		
	TreeCount		
	VisitorCount		

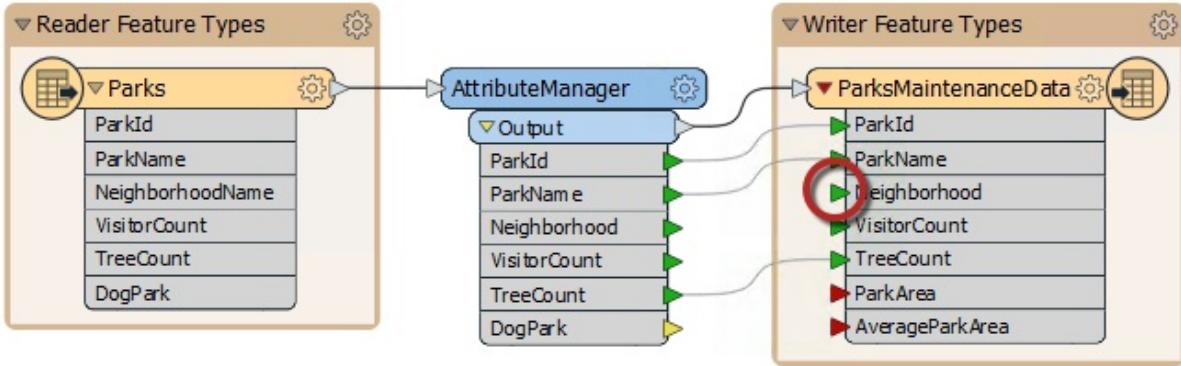
+ - ▲ ▼ ⌂ ⌃ Import ... C ▾

In response, the Action field will change to read *Rename*.

### TIP

*Neighborhood only appears in the list because it already exists on the writer schema. If we had done this step before editing the writer schema, we would have had to manually enter the new attribute name in this dialog.*

Click OK to close the dialog. Now you will see the Neighborhood attribute is flagged with a green arrow, to confirm that a value is being provided to that attribute.

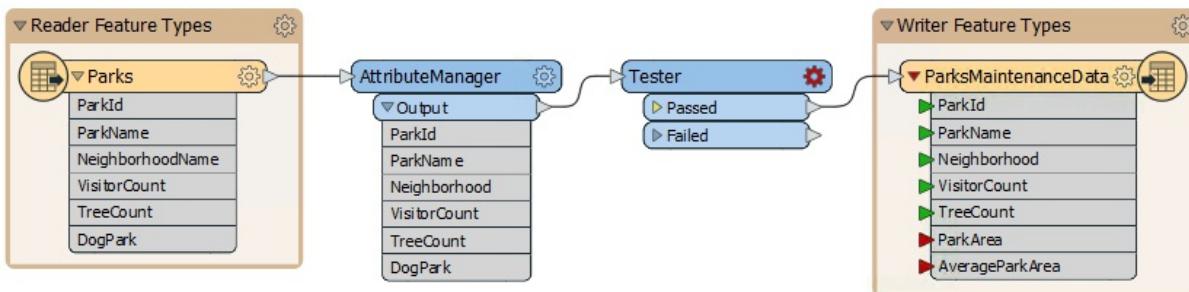


#### 4) Add a Tester Transformer

Now we should remove dog parks from the data because these have their own set of costs.

We can do this with a Tester transformer. Click on the connection from the AttributeManager output port to the ParksMaintenanceData feature type on the Writer.

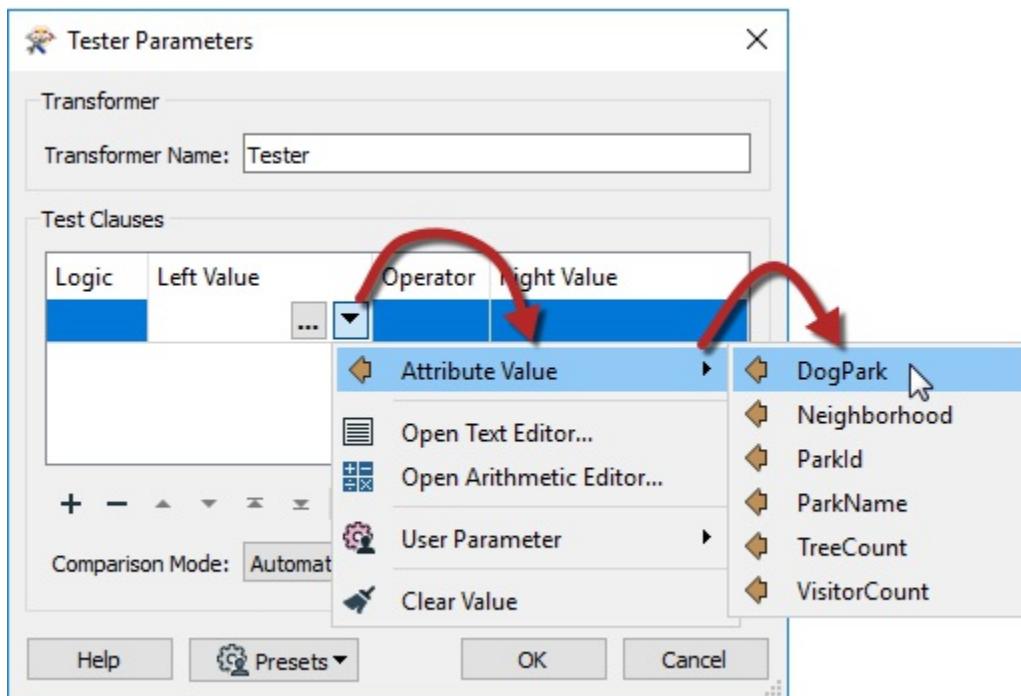
Start typing the word Tester. When you spot the Tester transformer, double-click on it to add one to the workspace. After tidying up the layout of the canvas, the workspace will now look like this:



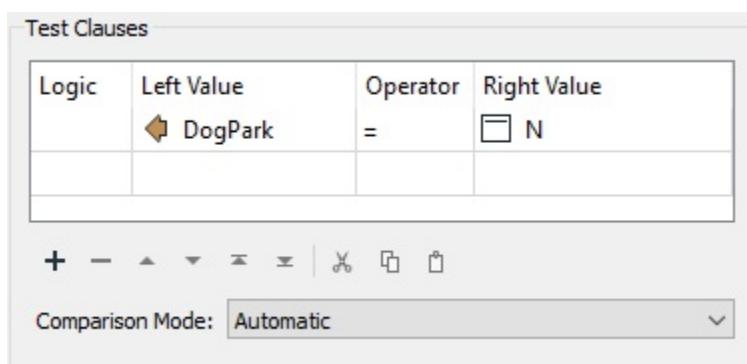
Notice that the Passed output port is the one connected by default.

#### 5) Set the Tester Parameters

Inspect the parameters for the Tester transformer. Click in the Left Value field and from there click the down arrow and choose Attribute Value > DogPark:



For the Right Value click into the field and type the value N. The operator field should be filled in automatically as the equals sign (=), which is what we want in this case.



Click OK to accept the values and close the dialog.

## NEW

*If you have used an older version of FME, you will notice that the Tester transformer looks a bit different in FME 2019.0. It still has the same great functionality, now with a Logic section to help streamline conditional statements.*

Test Clauses			
Logic	Left Value	Operator	Right Value
	◀ DogPark	=	<input type="checkbox"/> N
AND OR AND NOT OR NOT ( AND ( OR ( AND (NOT OR (NOT			

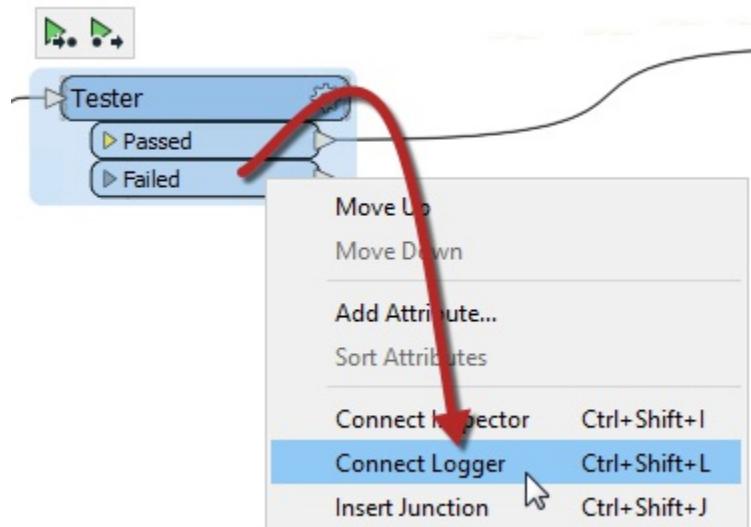
## TIP

The test is for DogParks=N because we want to keep those features, and it is the Passed port that is connected. If the test was DogParks=Y then the Failed port would be the one to connect.

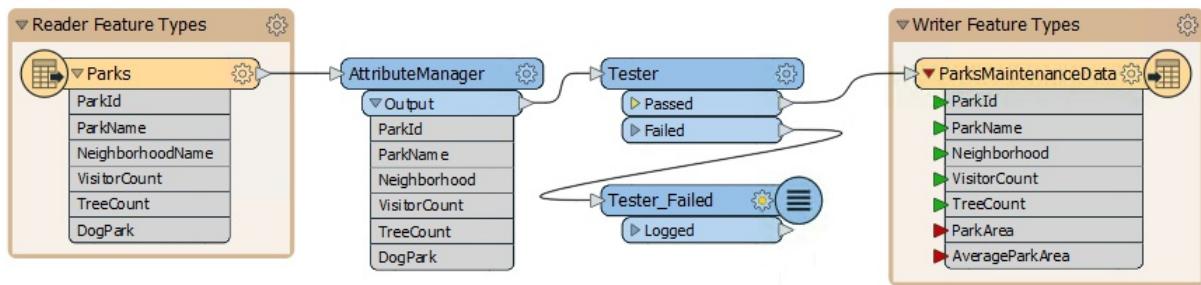
## 6) Add a Logger Transformer

We are now filtering out dog parks from our data, using a test on an attribute value. The question is, what should we do with the features we have filtered out? There are many things we could do, but for now, we'll log the information to the FME log window.

To do this, right-click on the Tester Failed port and choose the option Connect Logger:



A Logger transformer will be added to the workspace. This transformer will record all incoming features to the log window:



Loggers inserted by this method are named after – and reported in the log with – the output port they are connected to, here `Tester_Failed`.

## 7) Run Workspace

Save and run the workspace. It is not yet complete, but running it will prove that everything is working correctly up to this point.

To confirm it ran correctly, you should have 73 features coming from the `Tester:Passed` output port and 7 from the `Tester:Failed` output port.

## Advanced Exercise

*Readers and writers on the canvas had a bookmark object added around them automatically. You can do the same to the transformers by selecting them all and either clicking **Bookmark** on the toolbar or pressing the **Ctrl+B** keys. Give the bookmark a name of your choice that describes the actions being carried out.*

*A bookmark is an example of what we call **Best Practice** in FME.*

## CONGRATULATIONS

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

- Add transformers to a workspace
- Carry out schema mapping with the `AttributeManager` transformer
- Filter data using the `Tester` transformer
- Record data using the `Logger` transformer
- Add bookmarks to a workspace

## Content Transformation

Content transformations are those that operate on the components of a feature.

### What is a Feature?

A **feature** in FME is an individual item within the translation. For spatial data, a feature is generally a geometric object (with or without a set of related attributes).

For tabular data, a feature is generally a record in a database, a row in a spreadsheet, or a line in a text file. Each column or cell is known as an **attribute**.



Features in FME have a flexible, generic representation that is unrelated to the format from which they originated. That means any transformer can operate on any FME feature, regardless of its source format. Sometimes content transformation operates on single features, sometimes on multiple features at once.

### FME Lizard says...

*You can think of Content Transformation as altering or editing data.*

*The wardrobe analogy still works here. You might take your clothes from the wardrobe to clean them, or alter them, or repair them, or dye them a new color, or all sorts of other tasks, before returning them to their place.*

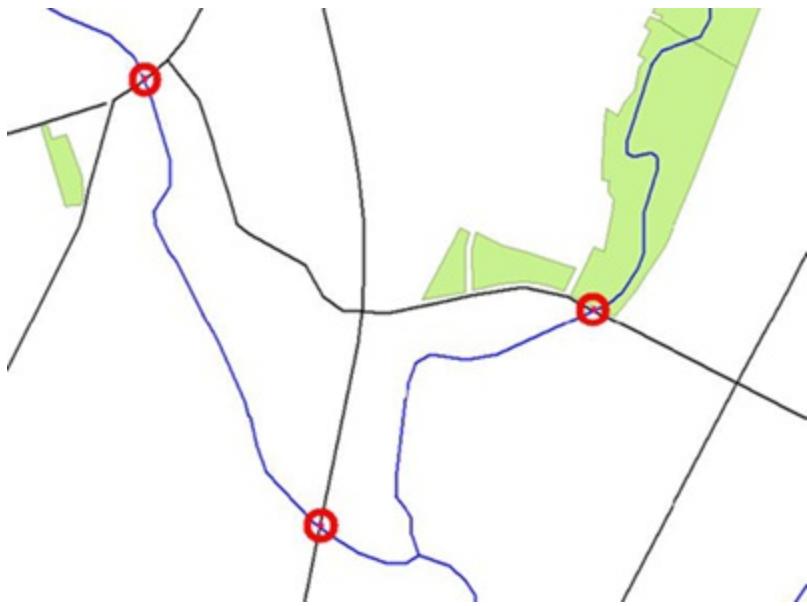
*The same holds true for spatial data transformation: it's the act of fixing up your data to be cleaner and in the style you really want.*

## Geometric Transformation

**Geometric Transformation** is the act of restructuring the spatial component of an FME feature. In other words, the geometry of the feature undergoes some form of change to produce a different output.

Some examples of geometric transformation include the following:

- **Generalization** – a cartographic process that restructures data so it's easily visualized at a given map scale
- **Warping** – adjustment of the size and shape of a set of features to more closely match a set of reference data
- **Topology Computation** – conversion of a set of linear features into a node/line structure
- **Line Intersection** - calculation of the intersection points between line features



Here roads have been intersected with rivers to produce points that mark the location of bridges.

## Attribute Transformation

**Attribute Transformation** is the act of restructuring the tabular components of an FME feature. In other words, the attributes undergo some form of change to produce a different output.

Some examples of attribute transformation are:

- **Concatenation** – joining together of two or more attributes
- **Splitting** – splitting one attribute into many, which is the opposite of Concatenation
- **Measurement** – measuring a feature's length or area to create a new attribute
- **ID Creation** – creating a unique ID number for a particular feature

In this example of attribute concatenation, each line of the address is concatenated to return a single line address.

Address1	Suite 1200,+
Address2	9639-137a Street,+
City	Surrey,+
Province	British Columbia,+
PostalCode	V3T 0M1
= Address      Suite 1200, 9639-137a Street, Surrey, British Columbia, V3T 0M1	

## Transformers in Series

Much like a set of components in an electrical circuit, a series of Workbench transformers can be connected to have a cumulative effect on a set of features.

## Chaining Transformers

Even with the large number of transformers available in FME, users frequently need a combination - or chain of transformers - instead of a single one.

A string of transformers that graphically represent an overall workflow is a key concept of FME:



In this example, a DuplicateFilter transformer removes duplicate polygon features. A Dissolver transformer merges each remaining (unique) polygon with its neighbor where there is a common boundary. Finally, each merged area gains an ID number from the Counter transformer.

## Exercise 3

# Grounds Maintenance Project - Calculating Statistics

<b>Data</b>	City Parks (MapInfo TAB)
<b>Overall Goal</b>	Calculate the size and average size of each park in the city, to use in Grounds Maintenance estimates for grass cutting, hedge trimming, etc.
<b>Demonstrates</b>	Content Transformation. Schema Mapping
<b>Start Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex3-Begin.fmw
<b>End Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex3-Complete.fmw C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex3-Complete-Advanced.fmw

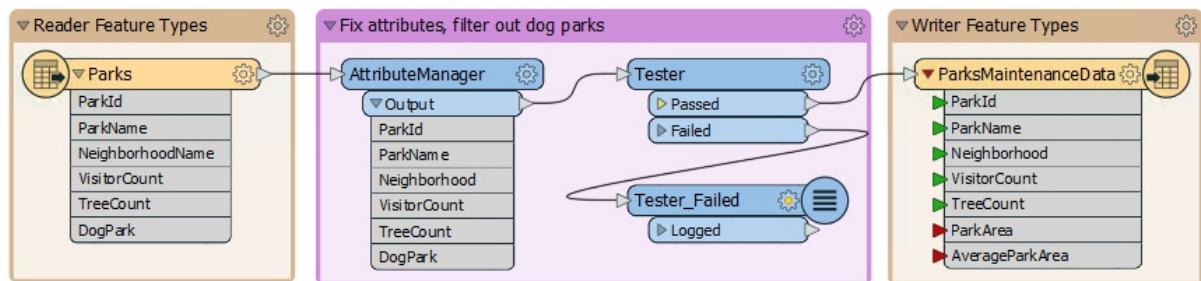
Let's continue your work on the grounds maintenance project.

In case you forgot, the team responsible for maintaining parks and other grassed areas needs to know the area and facilities of each park to plan their budget for the upcoming year.

In this part of the project, we'll calculate the size and average size of each park, and ensure that information is correctly mapped to the destination schema.

### 1) Start FME Workbench

Start FME Workbench (if necessary) and open the workspace from Exercise 2. Alternatively you can open C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex3-Begin.fmw



### 2) Add an AreaCalculator Transformer

To measure the area of each park feature, an AreaCalculator transformer must be used. "Calculator" is the term for when FME computes new attribute values.

Click on the connection between Tester:Passed port and the writer feature type *ParksMaintenanceData*. Start typing the letters "areac". You will see the Quick Add list of matching transformers appear beneath.

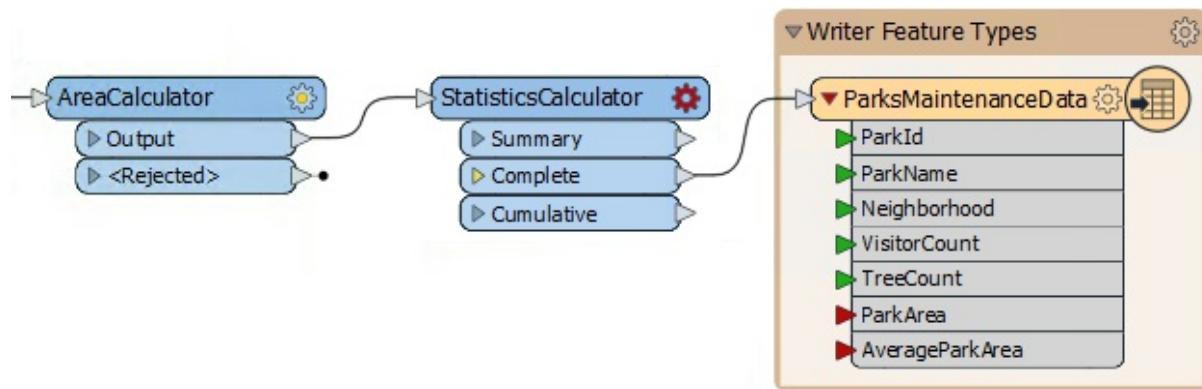
Select the transformer named AreaCalculator by double-clicking it:

areac

<p><b>FME Transformers</b></p> <ul style="list-style-type: none"> <li> AreaCalculator</li> </ul> <p><b>Custom Transformers</b></p> <p><b>FME Hub Transformers</b></p> <ul style="list-style-type: none"> <li> ArcGISOnlineSe...AreaCalculator</li> <li> AverageAreaCalculator</li> <li> GeographicAreaCalculator</li> </ul> <p><b>Readers</b></p> <p><b>Writers</b></p>	<p><b>AreaCalculator</b></p> <p>Calculates the area of a polygonal object and stores the value in an attribute.</p> <p><a href="#">Browse Additional Help ...</a></p>
---	---

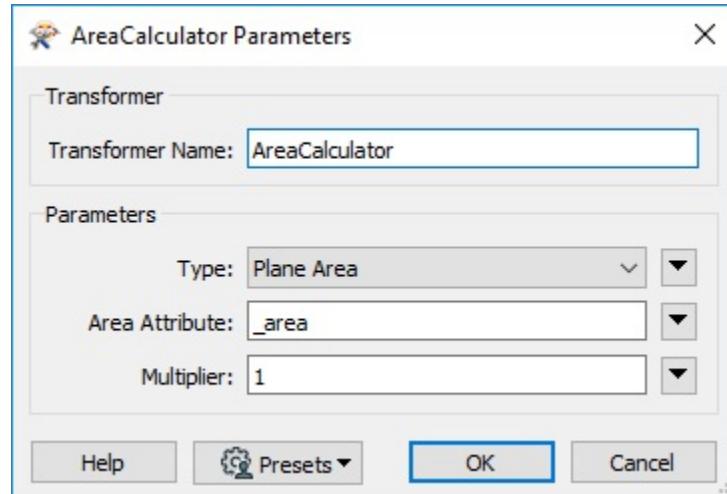
### 3) Add a StatisticsCalculator Transformer

Using the same method, place a StatisticsCalculator transformer between the AreaCalculator:Output port and the *ParksMaintenanceData* feature type. By default the output connection will be on the Summary port, but we want the complete dataset, so move the connection to the Complete port:

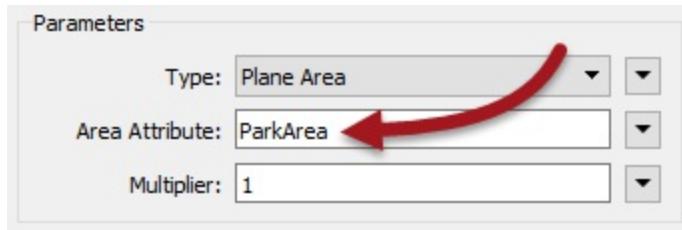


### 4) Check the AreaCalculator Parameters

Inspect the AreaCalculator parameters:



The default settings cause the calculated value to be placed into an attribute called `_area`. However, the `ParksMaintenanceData` schema requires an attribute called `ParkArea`, so change this parameter to create the correct attribute:

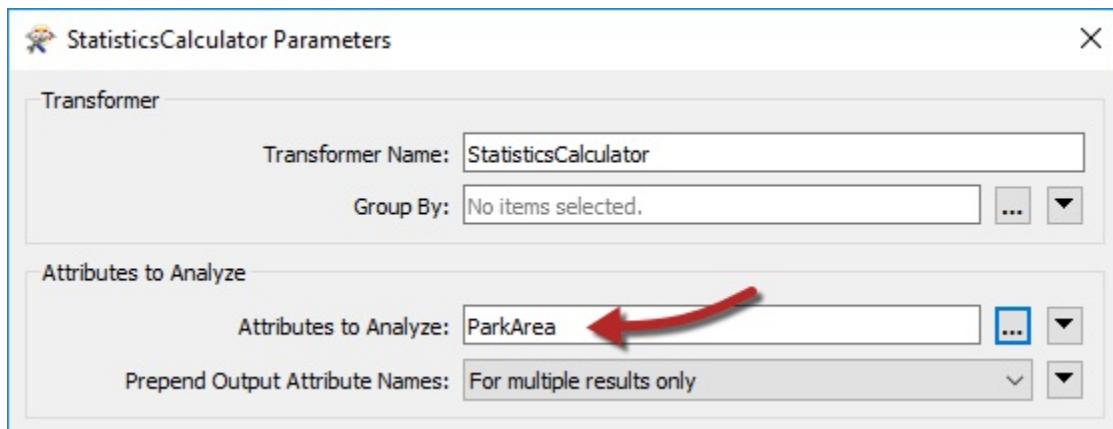


Once you click OK or Apply, notice that the attribute on the writer feature type is now flagged as connected.

## 5) Check the StatisticsCalculator Parameters

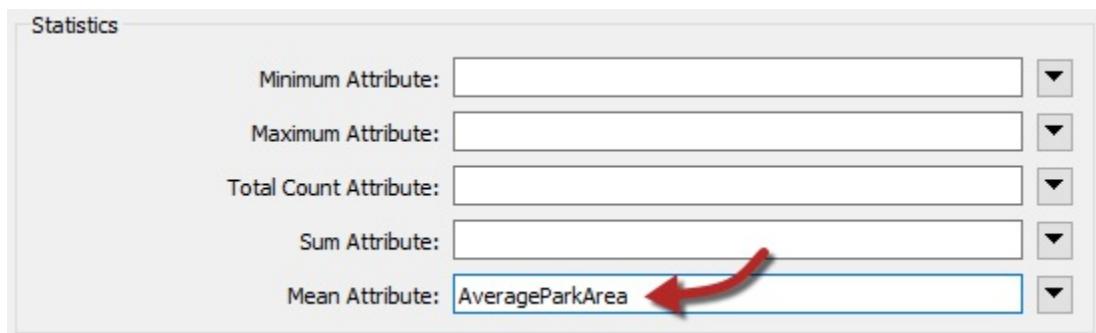
A red cogwheel icon indicates the StatisticsCalculator has parameters that need to be defined.

Inspect the StatisticsCalculator transformer's parameters. The Attribute to Analyze is the one containing the calculated area; so select ParkArea:



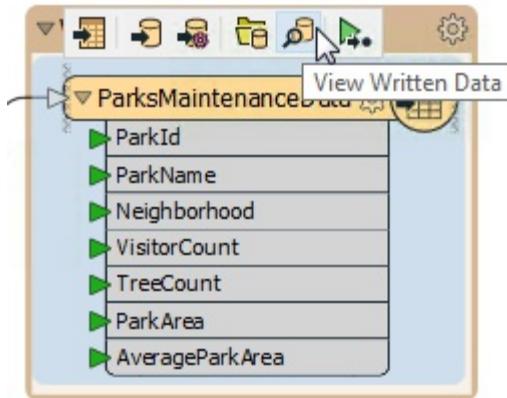
Examine what the default setting is for an attribute name for average (mean) park size. Currently, it doesn't match the `ParksMaintenanceData` schema, which requires an attribute named `AverageParkArea`.

Change the Mean Attribute from `_mean` to `AverageParkArea`. For Best Practice reasons, delete/unset any StatisticsCalculator output attributes that aren't required (for example `_min` and `_sum`).



## 6) Run the Workspace

Add another bookmark if you wish and run the workspace. Once the workspace has finished running, click on the ParksMaintenanceData writer feature type to bring up the popup menu, then click on the View Written Data button to view the data in Visual Preview.



The results will then be displayed in the Visual Preview window:

Visual Preview

Table

ParksMaintenanceData

	ParkId	ParkName	Neighborhood	VisitorCount	TreeCount	ParkArea	AverageParkArea
1	1	<missing>	Kitsilano	9406		10 448.1246806660...	70248.27339128363
2	2	Rosemary Brow...	Kitsilano	13100		8 1035.077080825...	70248.27339128363
3	3	Tea Swamp Park	Mount Pleasant	11275		2 2631.263986182...	70248.27339128363
4	4	<missing>	Strathcona	9755		6 1984.836357179...	70248.27339128363
5	5	Morton Park	West End	14977		4 2197.318199650...	70248.27339128363
6	6	Mcbride Park	Kitsilano	15053		9 17125.71708398...	70248.27339128363
7	7	Granville Park	Fairview	15185		13 19655.70536297...	70248.27339128363
8	8	<missing>	Mount Pleasant	8061		3 3123.723072199...	70248.27339128363
9	9	Creekside Park	Mount Pleasant	12321		10 23975.70526441...	70248.27339128363
10	10	China Creek So...	Mount Pleasant	12968		8 14750.38029474...	70248.27339128363
11	11	Barclay Heritag...	West End	12918		8 5647.902223034...	70248.27339128363

The Table View window shows the area of each park and the average area of all parks.

## 7) Save the Workspace

Save the workspace – it will be completed in further examples.

### Advanced Exercise

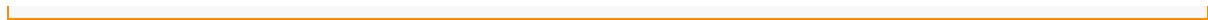
Notice that the numbers in the Table View show the results have been calculated to 12 decimal places. This is in excess of the precision that you require. As an advanced task - if you have time - use the AttributeRounder transformer to reduce the values to just 2 decimal places.

If you wish, you can also calculate the smallest, largest, and total park areas; but don't forget to add them to the writer schema if you want them to appear in the output.

### CONGRATULATIONS

By completing this exercise you have learned how to:

- Carry out content transformation with transformers (*AreaCalculator*, *StatisticsCalculator*)
- Manage transformer connections using pop-up buttons
- Use transformer parameters to create attributes that match the writer schema



## Feature Count Display

Look back at the previous example's translation. Did you notice that while the workspace was running, each connection was updated with the number of features that had passed along it?



The final feature counts show that 80 features were read, 73 passed the Tester while seven failed (for being dog parks) and went on to the Logger.

The Log window confirms the number of features written and lists the features that failed the Tester.

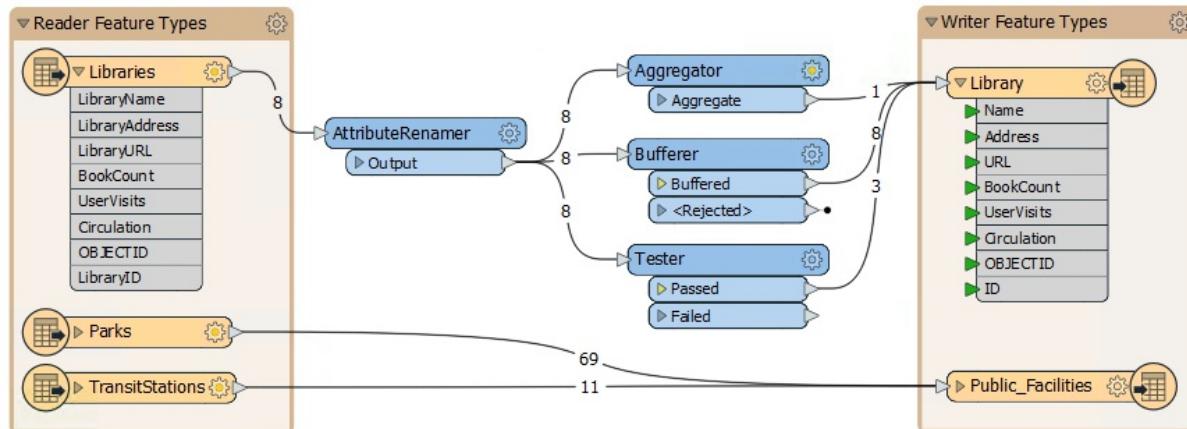
These numbers help analyze the results of a workspace and provide a reference for debugging if the output differs from what was expected.

## Transformers in Parallel

A **stream** is a flow of data represented by connections in the workspace. A key concept in FME is the ability to have multiple parallel streams within a workspace.

## Multiple Streams

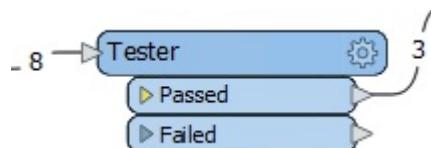
Multiple streams are useful when a user needs to process the same data but in many different ways. A workspace author can turn one stream into several, or combine several streams of data into one, as required:



Here an author is creating three data streams, each of which is processed separately then combined back into a single stream.

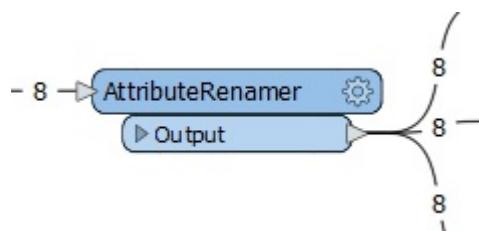
## Creating Multiple Streams

Creating multiple data streams can occur in a number of ways. Sometimes a transformer with multiple output ports (a Tester transformer is a good illustration of this) will divide (or filter) data with several possible output streams:



Here data is divided into two streams, one of which is not connected to anything.

Additionally, a full stream of data can be duplicated by simply making multiple connections out of a single output port. In effect it creates a set of data for each connection:

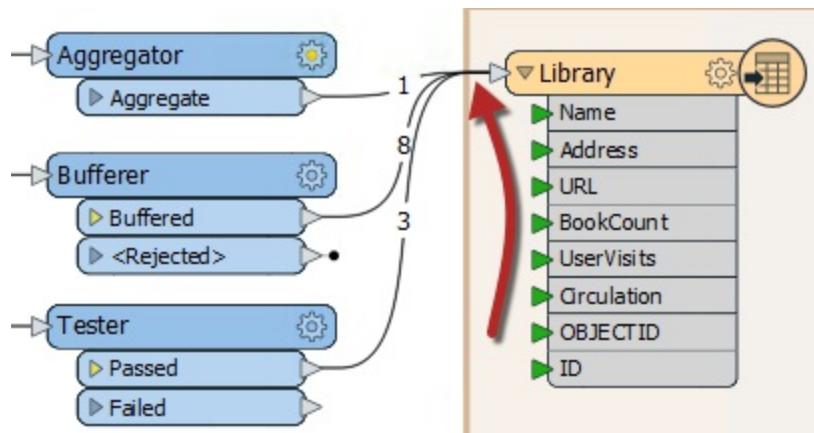


Here FME reads 8 features but, because there are multiple connections, creates multiple copies of the data.

## Bringing Together Multiple Streams

When multiple streams are connected to the same input port, no merging takes place. The data is simply accumulated into a single stream. This is often called a *union*.

Here, three streams of data converge into a writer feature type:



No merging takes place; the data simply accumulates into 12 distinct features in the output dataset. Think of it as three pipes of water emptying into a single container.

To carry out actual merging of data requires a specific transformer such as the FeatureMerger or FeatureJoiner.

## Exercise 4

# Grounds Maintenance Project - Labelling Features

<b>Data</b>	City Parks (MapInfo TAB)
<b>Overall Goal</b>	Calculate the size and average size of each park in the city, to use in Grounds Maintenance estimates for grass cutting, hedge trimming, etc.
<b>Demonstrates</b>	Content transformation with parallel transformers
<b>Start Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex4-Begin.fmw
<b>End Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex4-Complete.fmw C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex4-Complete-Advanced.fmw

Let's continue your work on the grounds maintenance project.

In this part of the project, we'll create a label for each park and write it to a new output layer. This step is best done using a parallel stream of data.

### 1) Start FME Workbench

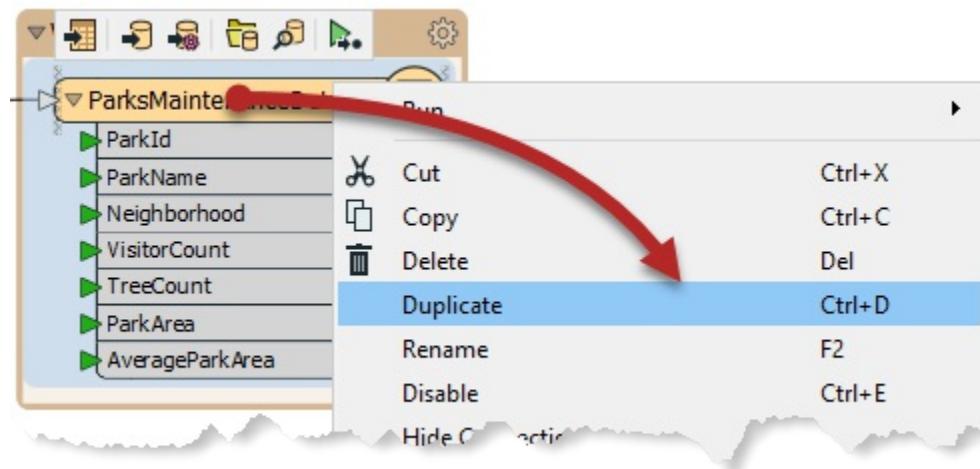
Start FME Workbench (if necessary) and open the workspace from Exercise 3. Alternatively you can open C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex4-Begin.fmw

The previous exercise measured park areas with the AreaCalculator. Now we are asked to add this information as labels to the output dataset.

This can be accomplished by using the LabelPointReplacer transformer but first, let's add another writer feature type to contain the labels.

### 2) Create New Writer Feature Type

Because we want to write label features to a separate layer (table) in the output, we need to create another feature type object on the canvas. There is more about this in a later chapter, but for now, right-click the writer feature type and choose the option Duplicate. Doing so creates a new feature type (here, a layer, but for other formats could be a table or other grouping of data) in the output dataset.



Now clean up this feature type's schema. View the feature type's dialog and rename the new feature type to ParkLabels. In the User Attributes tab delete all of the existing user attributes.

### TIP

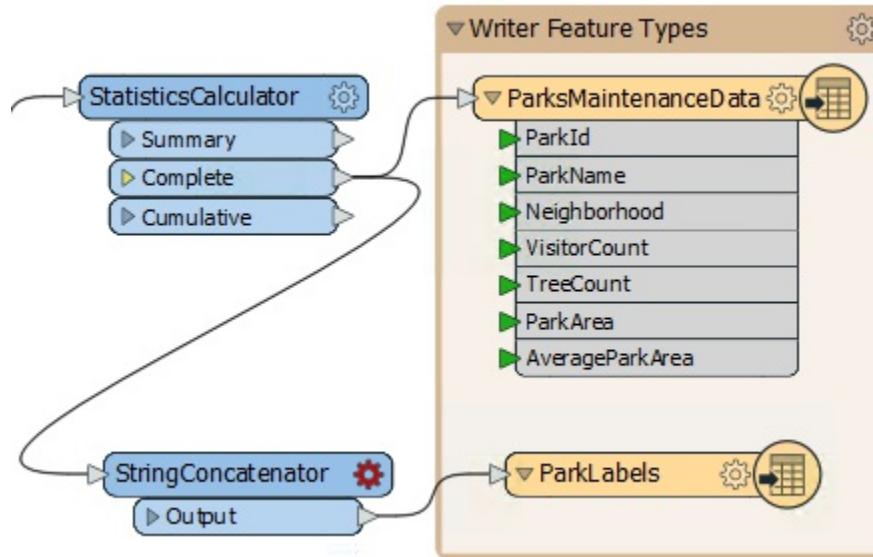
You can quickly remove all of the existing user attributes by switching the Attribute Definition to

*Automatic and then back to Manual. The Automatic will clear it because there are currently no attributes coming in. This only works for feature types that have not been connected to the rest of the workspace yet*

### 3) Place a StringConcatenator Transformer

Click onto a blank area of canvas. Type "StringConcatenator" to add a transformer of this type.

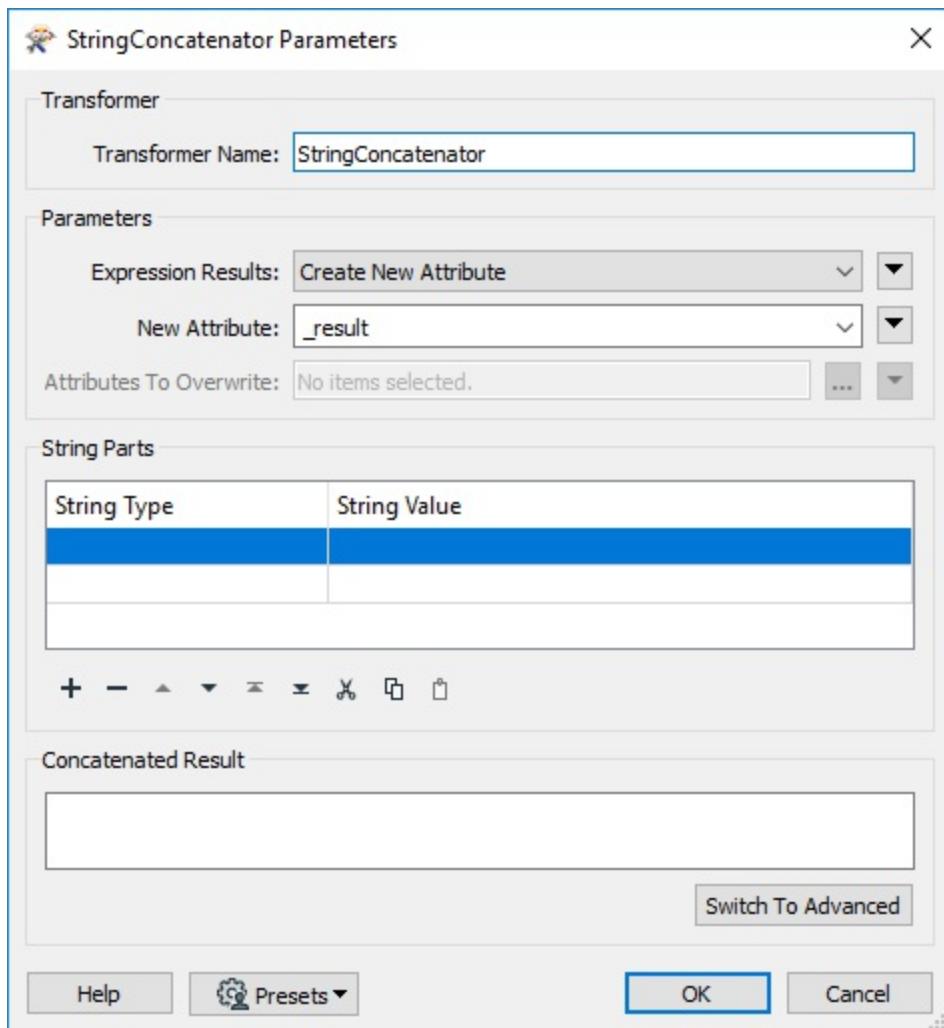
Connect it to the Complete port of the StatisticsCalculator by dragging a second connection from there to the new transformer.



Make a new connection from the StringConcatenator to the ParkLabels feature type.

### 4) Check the StringConcatenator Parameters

View the parameters for the StringConcatenator transformer. There are both basic and advanced dialogs, and the basic one looks like this:



Enter *LabelText* as the name for the new attribute to create.

In the String Parts section, set the following four parts:

String Type	String Value
Attribute Value	ParkName
New Line	
Attribute Value	ParkArea
Constant	sq meters

Be sure to include a space character in the constant before "sq meters".

**Parameters**

Expression Results:	Create New Attribute
New Attribute:	LabelText
Attributes To Overwrite:	No items selected.

**String Parts**

String Type	String Value
Attribute Value	ParkName
New Line	
Attribute Value	ParkArea
Constant	Sq Meters

**Concatenated Result**

```
@Value(ParkName)
@Value(ParkArea)Sq Meters
```

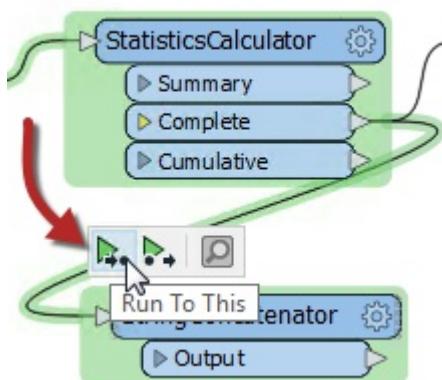
**Switch To Advanced**

### TIP

You may find it quicker to switch to the Advanced editor dialog and enter the content directly:  
`@Value(ParkName)`  
`@Value(ParkArea) sq meters`

### 5) Run the Workspace

Let's run the workspace to ensure that the StringConcatenator is doing what we expected. Ensure that Feature Caching is enabled, and then click on the StringConcatenator to open the popup menu. Then click on Run to This, that way the StringConcatenator will run, but we won't be rewriting the data out:



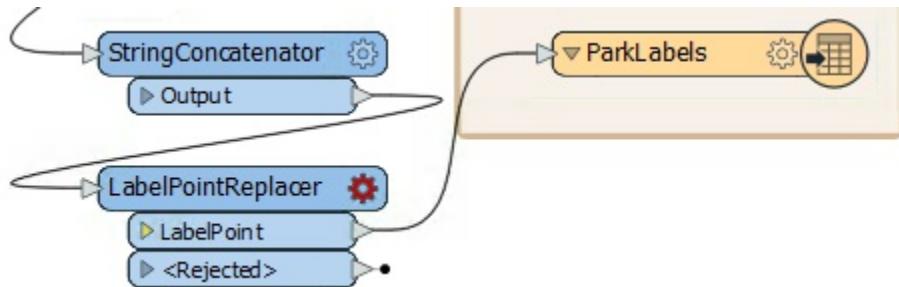
Then click the magnifying glass on the output port to view the cache in Visual Preview. If you double-click on any of the features in the LabelText column, a dialog will appear that will show the label strings formatted correctly:

ParkId	ParkName	Neighborhood	VisitorCount	TreeCount	DogPark	ParkArea	AverageParkArea	LabelText
1	<missing>	Kitsilano	9406	10	N	448.1246806660...	70248.27339128363	448.1246806660...
2	Rosemary Brow...	Kitsilano	13100	8	N	1035.077080825...	70248.27339128363	Rosemary Brow...
3	Tea Swamp Park	Mount Pleasant	11275	2	N	2631.263986182...	70248.27339128363	Tea Swamp Pa...
4	<missing>	Strathcona	9755	6	N	684.83635779...	70248.27339128363	684.83635779...
5	Morton Park	West End	14977	10	N	123.723072199...	70248.27339128363	Morton Park21...
6	Mcbride Park	Kitsilano	15053	12	N	123.723072199...	70248.27339128363	Mcbride Park17...
7	Granville Park	Fairview	15185	14	N	123.723072199...	70248.27339128363	Granville Park1...
8	<missing>	Mount Pleasant	8061	8	N	123.723072199...	70248.27339128363	Creekside Park2...
9	Creekside Park	Mount Pleasant	12321	10	N	123.723072199...	70248.27339128363	China Creek So...
10	China Creek So...	Mount Pleasant	12968	12	N	123.723072199...	70248.27339128363	Barclay Heritag...
11	Barclay Heritag...	West End	12918	14	N	123.723072199...	70248.27339128363	Barclay Heritag...

## 6) Place a LabelPointReplacer Transformer

Click onto the connection between StringConcatenator:Output and the ParkLabels feature type. Type "LabelPointReplacer" to add a transformer of this type.

The new transformer will be added and automatically connected between those two objects.



## 7) Check LabelPointReplacer Parameters

Inspect the LabelPointReplacer parameters.

Firstly click the drop-down arrow to the right of the Label parameter:

Transformer
Transformer Name: LabelPointReplacer

Parameters

Label:  ...

Label Height:

Always Rotate Label: Yes

Attribute Value

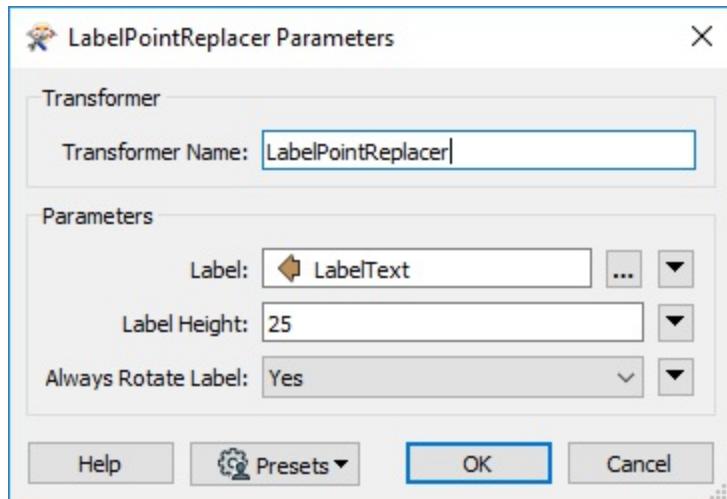
- Attribute Value
- Open Text Editor...
- User Parameter
- Conditional Value...
- Clear Value

- AverageParkArea
- DogPark
- LabelText
- Neighborhood
- ParkArea
- ParkId
- ParkName
- TreeCount
- VisitorCount

Select Attribute Value > LabelText to select the label previously defined in the StringConcatenator.

Now click in the Label Height field and type 25 (that's 25 working units, which in this case is meters).

The "Always Rotate Label" parameter can be left to its default value.



## TIP

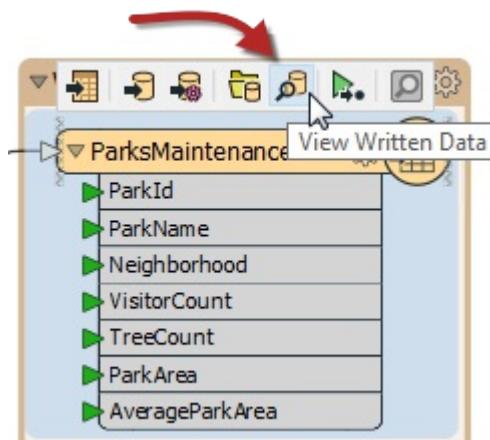
Many parameter fields (like Label Height) can be set either as a constant value (by typing it in) or set to an attribute by clicking the drop down arrow and selecting Attribute Value.

And - as you'll see shortly - it's also possible to construct a parameter value directly inside the transformer settings

## 8) Run the Translation

Add another bookmark if you wish, then run the translation, since the translation is cached up to the StringConcatenator it will only take a moment to write out the data.

Click on the ParksMaintenanceData to open the popup menu, then click on View Written Data:



The data will open in Visual Preview, but we want to view both our outputs together, so we will have to move to FME Data Inspector. In Visual Preview, click on the Open in Data Inspector button or hit Ctrl + Alt + D. FME Data Inspector will open and display the data in the current view:

Visual Preview

Table

ParksMaintenanceData

ParkId	ParkName	Neighborhood	VisitorCount	TreeCount	ParkArea
1	<missing>	Kitsilano	9406	10	448.1
2	Rosemary Brow...	Kitsilano	13100	8	1035.0
3	Tea Swamp Park	Mount Pleasant	11275	2	2631.
5	Morton Park	West End	14977	4	2197.5
6	Mcbride Park	Kitsilano	15053	9	17125.

Now with FME Data Inspector open, click on the Add data button on the menu bar. This will add data to the current view so we can view the labels on top of the parks:



In the Select Dataset to Add dialog, set the Format to MapInfo TAB (MITAB) and then browse to the ParkLabels Dataset:

```
C:\FMEData2019\Output\ParkLabels.tab
```

Click OK, and the labels should now be on the parks. Notice that the output is in two layers in two files. Use the FME Data Inspector to open both output files in the same view.



Map tiles by [Stamen Design](#), under [CC-BY-3.0](#). Data by [OpenStreetMap](#), under [CC-BY-SA](#).

## Advanced Exercise

Now you know how to create a new feature type in the output, how to test data, and how to use parallel streams, why not try this task: Identify which parks are smaller than average and which parks are larger than average, and write them out to different feature types.

## CONGRATULATIONS

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

- *Create a new writer feature type*
- *Use multiple streams of transformers in a single workspace*
- *Use the StringConcatenator to construct a string for use elsewhere*
- *Use an attribute as the value of a transformer's parameter*

## Group-By Processing

Group-By parameters allow features to be processed in groups by a single FME transformer.

### What is a Group?

FME transformers carry out transformations on either one feature at a time, or on a whole set of features at once.

For example, the *AreaCalculator* transformer operates on one feature at a time (to measure the area of a single polygon feature). We call it a **feature-based transformer**.

The *StatisticsCalculator* operates on multiple features at a time (to calculate an average value for them all). In FME we call this set of features a **group** and the transformer is a **group-based transformer**.

### Creating Groups

So a group is simply a defined set of features being processed by a transformer. By default, a group-based transformer treats ALL the features that it is fed, as a single group.

However, such transformers also have a **Group-By** parameter. This parameter lets the user define several groups based upon the value of an attribute.

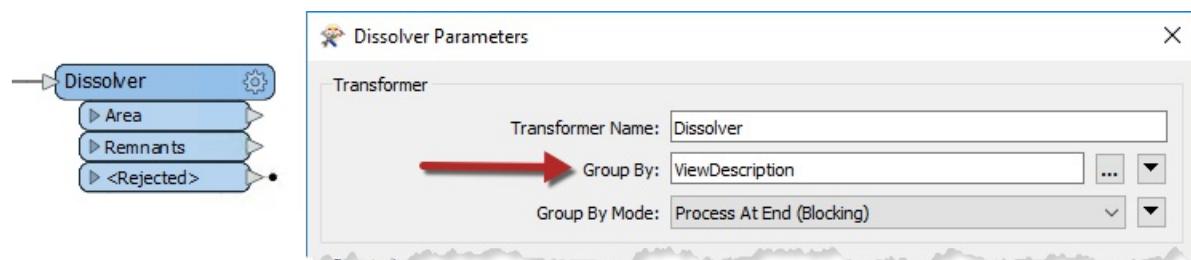
#### FME Lizard says...

*To illustrate groups let's consider calculating the mean age of FME users. The default group for the calculation includes **all** FME users.*

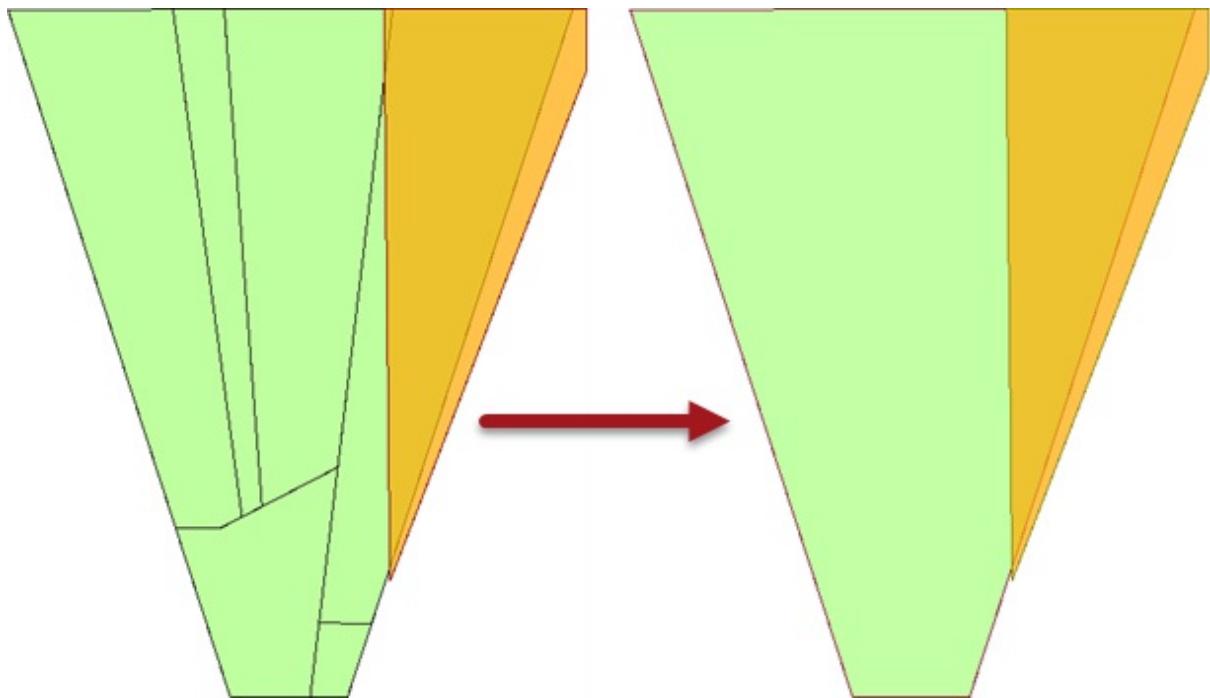
*But you could instead divide everyone up into men and women, creating two groups, and calculate average age per gender. Or you could divide everyone into their nationality, and calculate average age per country.*

*This is the same as having a gender (or nationality) attribute in a dataset and selecting that in an FME group-by parameter.*

Here, a Dissolver transformer is being used to dissolve (merge) a number of polygon features. The selected Group-By attribute is *ViewDescription*:



FME creates a series of groups for overlaying, where the features in each group share the same value for the *ViewDescription* attribute. The practical outcome is that polygon dissolving takes place only where line features share the same description:



## Group By Mode

When grouping features, there are two different ways the transformer can handle the group. The first way is to hold all of the features until all of the features have come through the transformer, this is referred to as blocking. This is set by using the Process at End (Blocking) Group By Mode.

The other way is to pre-sort your data into groups first by using a transformer like the Sorter. Then once your data is grouped, use the Process When Group Changes (Advanced) Group By Mode. This mode will push the features through the transformer after each group which will help speed up performance. Only use this option when your data is pre-sorted.

## Exercise 5

# Grounds Maintenance Project - Neighborhood Averages

<b>Data</b>	City Parks (MapInfo TAB)
<b>Overall Goal</b>	Calculate the size and average size of each park in the city, to use in Grounds Maintenance estimates for grass cutting, hedge trimming, etc.
<b>Demonstrates</b>	Group-By Processing
<b>Start Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex5-Begin.fmw
<b>End Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex5-Complete.fmw

Let's continue your work on the grounds maintenance project.

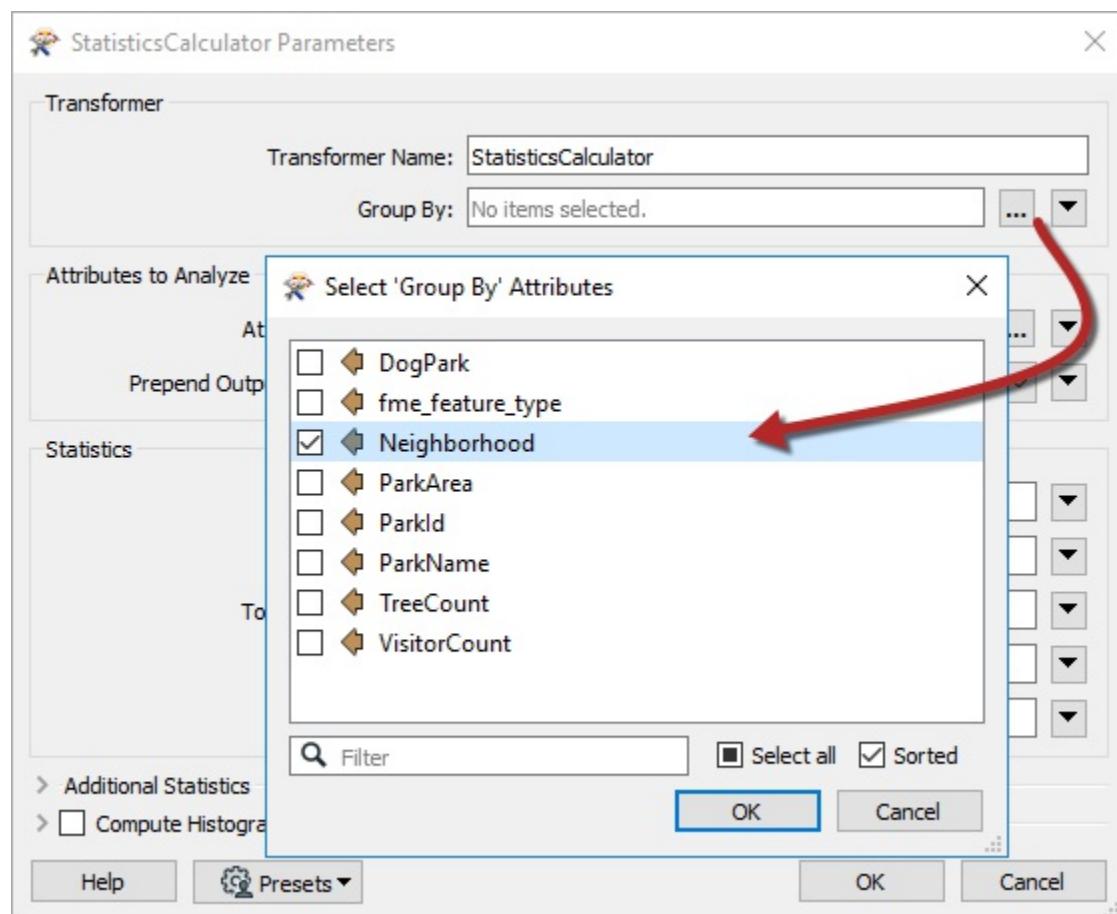
The parks team has decided that they do not want the average area of parks for the city as a whole. Instead, they want the average size of parks in each neighborhood; so let's do that for them.

### 1) Start FME Workbench

Start FME Workbench (if necessary) and open the workspace from Exercise 4. Alternatively you can open C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex5-Begin.fmw

### 2) Set Group-By in the StatisticsCalculator

This is a straightforward task. View the parameters for the StatisticsCalculator transformer and click the 'browse' button next to the Group By parameter. Select the attribute called Neighborhood:



Click OK/Apply to apply the changes to the transformer.

### 3) Run the Workspace

Click on the StatisticsCalculator and then click on Run To This. Inspect the Summary output port in Visual Preview. The Summary output port creates a summarized table to any of the statistics you are calculating.

You should see that each neighborhood now has its own value for AverageParkArea:

Table		
StatisticsCalculator_Summary		
	Neighborhood	AverageParkArea
1	Downtown	10493.657892171956
2	Fairview	14973.303765384839
3	Kitsilano	23986.343825070326
4	Mount Pleasant	11660.252762014796
5	Strathcona	10196.964710694112
6	West End	300747.86774834385

## CONGRATULATIONS

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

- Use the group-by parameter in FME transformers

## Coordinate System Transformation

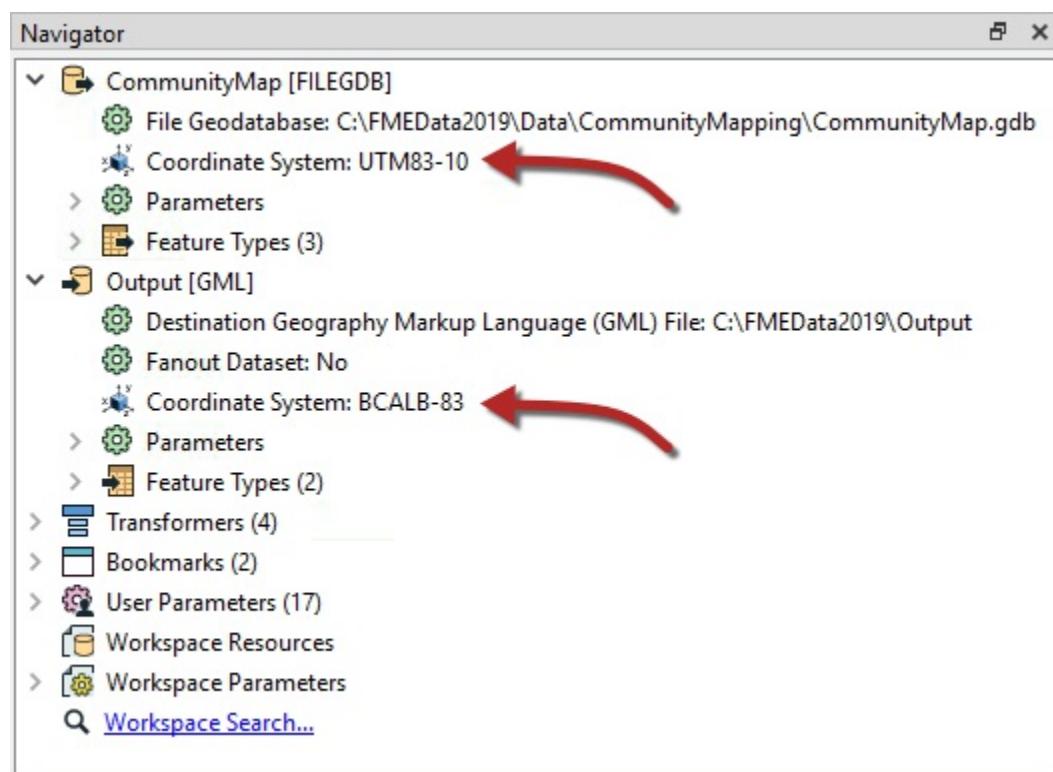
To be located in a particular space on the Earth's surface, the majority of spatial data is related to a particular spatial reference.

Some users call this location of data a "projection," but projection is just one component of what we call a **coordinate system**. A coordinate system includes *projection*, *datum*, *ellipsoid*, *units*, and sometimes a *quadrant*.

## Coordinate System Settings

Each reader and writer within FME can be assigned a coordinate system. That coordinate system is set in the Navigator window of Workbench or the Generate Workspace dialog.

Like the source schema, the reader coordinate system is "**what we have**" and the writer coordinate system is "**what we want**". Here the source coordinate system has been defined as UTM83-10 and the destination as BCALB-83:



Each feature processed by the reader is tagged with the coordinate system defined in its parameter.

When a feature arrives at a writer, if it is tagged with a different coordinate system to what is defined for that writer, then FME automatically reprojects the data, so that the output is in the correct location.

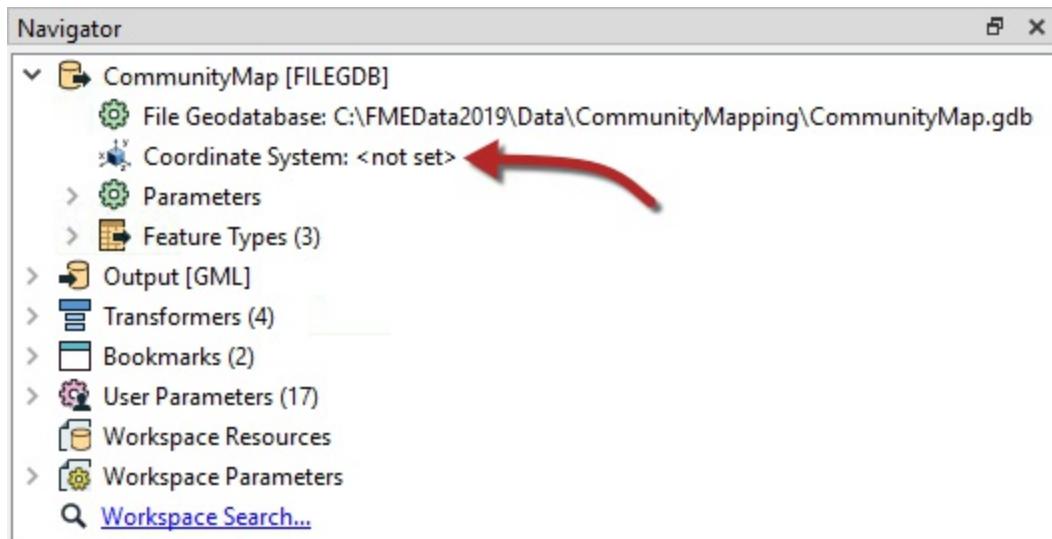
### TIP

Once tagged with a coordinate system, each feature retains this throughout the translation; FME knows what coordinate system it belongs to at all times.

This is important when carrying out geometric transformations (like calculating area) or when reading multiple datasets that belong to different coordinate systems (yes, FME will handle that).

## Automatic Detection of Coordinate Systems

It's not always necessary to set the coordinate system parameters manually. Some data formats (for example Esri Shapefile) are capable of storing information about the coordinate system in which they are held, and FME will retrieve this information where it can.



Here, because the reader coordinate system is marked <not set>, FME will try to determine the coordinate system from the source dataset. If it can't, then the feature will be tagged with a coordinate system of <unknown>.

There are a number of reprojection scenarios that may occur depending on the combination of coordinate system (CS) information available:

Dataset CS	Reader CS	Writer CS	Reprojection
N	Y	Y	Reprojects from Reader CS to Writer CS
Y	N	Y	Reprojects from Dataset CS to Writer CS
N	N	Y	Error: Cannot reproject without Dataset or Reader CS
Y	Y	Y	Reprojects from Reader CS to Writer CS

If the coordinate system is not set on the writer, then no reprojection will take place unless the output format requires it. For example, the KML format requires data to be in Latitude/Longitude. If neither the source dataset or the reader coordinate system is defined, then the translation will fail.

## Exercise 6

# Grounds Maintenance Project - Data Reprojection

<b>Data</b>	City Parks (MapInfo TAB)
<b>Overall Goal</b>	Calculate the size and average size of each park in the city, to use in Grounds Maintenance estimates for grass cutting, hedge trimming, etc.
<b>Demonstrates</b>	Data reprojection
<b>Start Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex6-Begin.fmw
<b>End Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex6-Complete.fmw C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex6-Complete-Advanced.fmw

Let's continue your work on the grounds maintenance project.

The parks team has decided that the output data should be in an Albers Equal Area projection (coordinate system = BCALB-83). They think it will take ages to set this up! We'll show them differently.

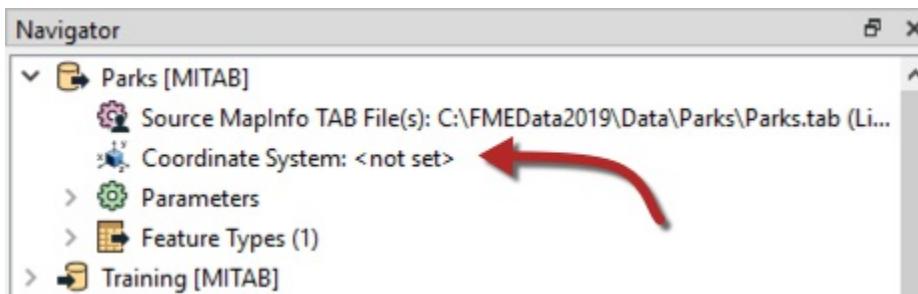
### 1) Start FME Workbench

Start FME Workbench (if necessary) and open the workspace from Exercise 5. Alternatively you can open C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex6-Begin.fmw

### 2) Edit Source (Reader) Coordinate System

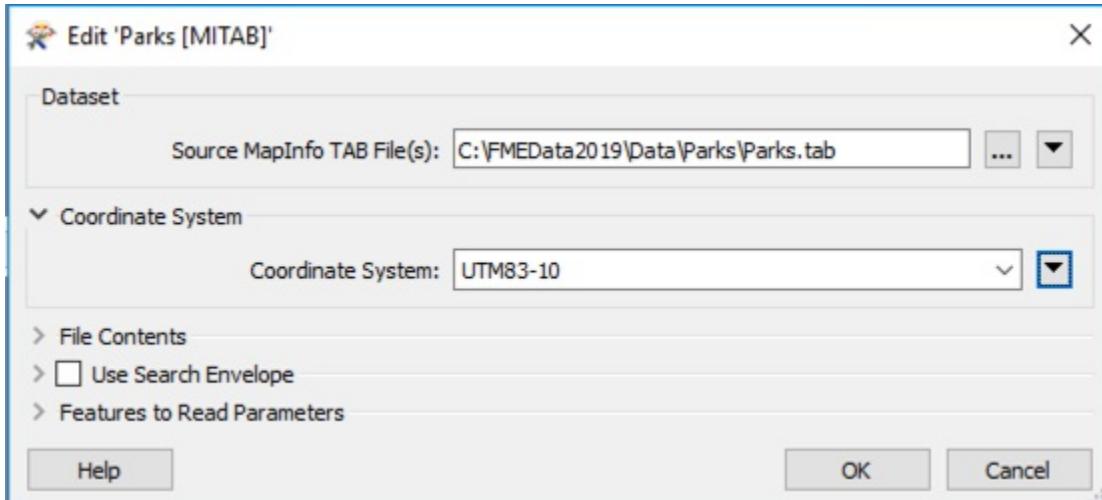
In the Navigator window locate the Parks [MITAB] reader, and expand its list of settings.

Locate the setting labeled 'Coordinate System'. The original value should be <not set>:



Double-click the reader Coordinate System parameter to open an edit dialog.

In the Coordinate System field enter the name UTM83-10 or select it from the Coordinate System Gallery by selecting "More Coordinate Systems..." from the bottom of the drop-down list:



## TIP

Remember, when a reader's Coordinate System parameter is defined as <not set> FME will automatically try to determine the correct coordinate system from the dataset itself.

When the source dataset is in a format that stores coordinate system information (as it does in this example) you can safely leave the parameter unset. So this step isn't really necessary.

However, you **MUST** set this parameter when you wish to reproject source data that does not store coordinate system information; otherwise an error will occur in the translation.

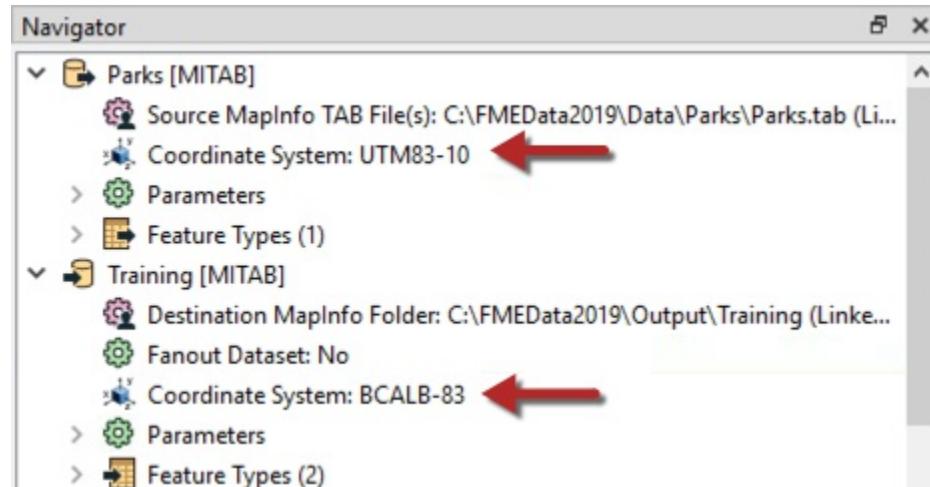
### 3) Edit Destination (Writer) Coordinate System

Now locate the coordinate system setting for the destination (writer) dataset.

Again the current value should be the default of <not set>.

Double-click the parameter and enter the coordinate system name BCALB-83 or select it from the Coordinate System Gallery by selecting "More Coordinate Systems..." from the bottom of the drop-down list.

The Navigator window will now look like this:



### 4) Run the Workspace

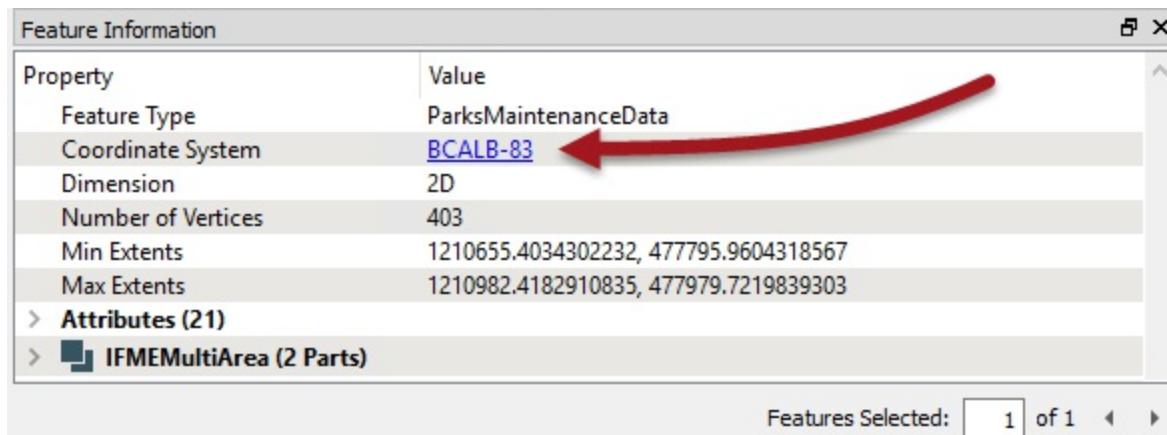
Save and then run the workspace.

In the log file you should be able to find:

```
FME Configuration: Source coordinate system for reader MITAB_1[MITAB] set to `UTM83-10'  
FME Configuration: Destination coordinate system set to `BCALB-83'
```

### 5) Inspect the Output

Open the newly reprojected dataset and select a feature. The Feature Information window should report that the data is now in BCALB-83. Optionally, click on the coordinate system name in that window; a new dialog will open to display all of the coordinate system parameters.



### TIP

If the background map is activated when a dataset is opened then the contents of that dataset are automatically reprojected to Spherical Mercator to match the background map. If you wish to see the data as it appears in its own coordinate system, then click on your background map name in the Visual Preview window, and select Background Map Off.

### Advanced Exercise

Instead of using the reader/writer parameters in the Navigator window, why not try this exercise using the Reprojector (or CSMMapReprojector) transformer? Where should the transformer be placed in the workspace and why is this important?

### CONGRATULATIONS

By completing this exercise you have learned how to:

- Use Coordinate System parameters to reproject spatial data
- Select features in the Data Inspector to inspect coordinate system information

## Module Review

This module introduced you to the concept of Data Transformation and explained how to use FME Workbench for more than just quick translations.

## What You Should Have Learned from this Module

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

### Theory

- A **schema** describes a dataset's structure, including its feature types, attributes, and geometries.
- **Schema editing** is the act of editing the destination schema to define better what is required out of the translation.
- The act of joining the source schema to the destination is called **schema mapping**. Differences between the two schemas lead to **structural transformation**.
- **Content transformation** is the modifying of data content during a translation. In FME Workbench data transformation is carried out using objects called **transformers**.
- **Groups** can be created using the Group-By option in a transformer's parameters.
- Splitting data into multiple streams creates multiple copies and not a division of data. Bringing together multiple streams combines the data instead of merging it.

### FME Skills

- The ability to edit a writer schema
- The ability to restructure data by mapping a reader to a writer schema, both manually and through transformers
- The ability to locate transformers in Workbench and use them to transform data content
- The ability to set transformer parameters in either the Parameter Editor or transformer dialogs
- The ability to define feature groups using the Group-By option
- The ability to reproject data using Workbench

### Further Reading

For further reading why not browse [Transformation articles](#) on our blog?

## Data Transformation Quiz

Each section ends with a quiz to test your new knowledge. Make your selection and click "Check my answers" to check each individual question. If you want an explanation for the answer, click "Explain".

**Note:** your score won't be tallied; this is just for review purposes.

**1)** What is the most common format translation defined with FME is from Esri Shapefile to which format?

- A. Esri Shapefile
- B. AutoCAD DWG
- C. Microsoft SQL
- D. GeoJSON
- E. Google KML
- F. Apple IMDF

**2)** Which three colors represent checked, need checking, and unset parameters on transformer objects?

- A. yellow, orange, green
- B. black, orange, purple
- C. green, yellow, red
- D. blue, yellow, red

**3)** If I use a transformer to remove irregularities (like self-intersecting loops) in the boundary of a polygon, what type of transformation is it?

- A. Structural Transformation of attributes
- B. Structural Transformation of geometry
- C. Content Transformation of attributes
- D. Content Transformation of geometry

**4)** Which of the following transformers do you think is "group-based"?

- A. StringFormatter
- B. Clipper
- C. Rotator
- D. AttributeRounder
- E. Dissolver

## Exercise 7

## Voting Analysis Project

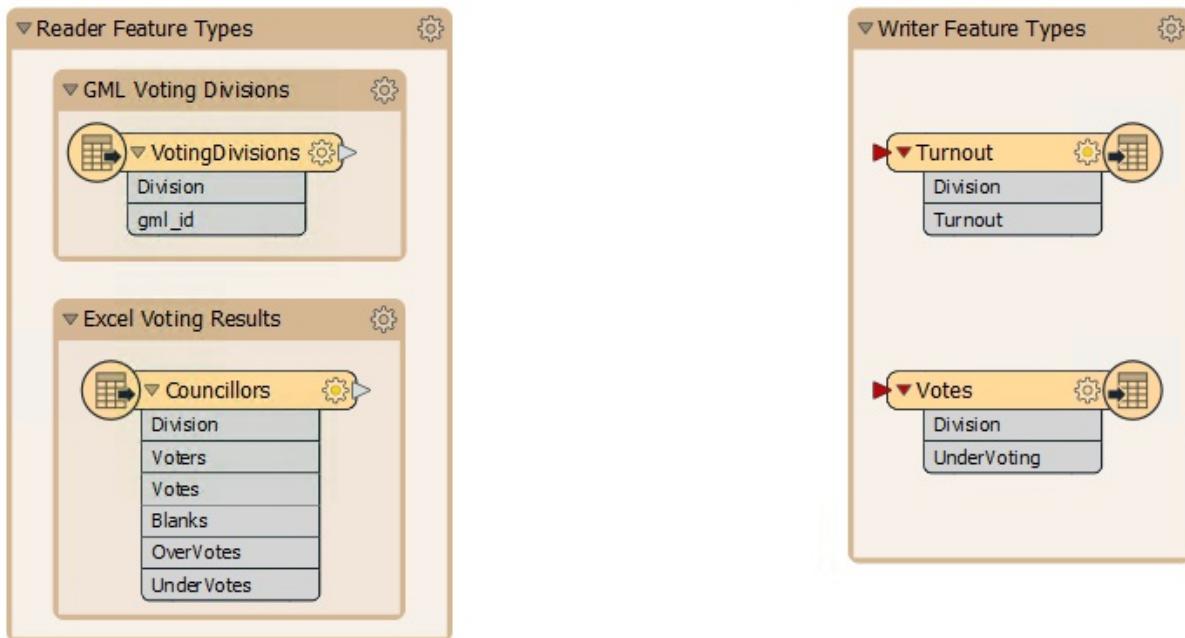
<b>Data</b>	Election Mapping (GML) Election Statistics (Microsoft Excel)
<b>Overall Goal</b>	Map statistics of voting patterns
<b>Demonstrates</b>	Data Transformation
<b>Start Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex7-Begin.fmw
<b>End Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex7-Complete.fmw C:\FMEData2019\Workspaces\DesktopBasic\DataTransformation-Ex7-Complete-Advanced.fmw

In a break from grounds maintenance projects, the municipal Elections Officer has heard about your skills and asked for help identifying voting divisions that had a low turnout at the last election, or divisions where voters had difficulties understanding the process.

He asks for your help, and you suggest the results should be presented in Google Earth so that staff can view them without having to use a full GIS system.

### 1) View Data

Start FME Workbench and open the starting workspace. It already has readers and writers added to handle the data; all we need to do is carry out the transformation, but first, let's inspect the data to know what we are working with:



Click on VotingDivisions to open the popup menu and click on the View Source Data button to view the data in the Visual Preview Window. Take note of the attributes in the table view.

Repeat this step with the Councillors and take note of the attributes.

### TIP

*If you need to view two or more reader feature types at the same time, FME Data Inspector is a useful tool as it opens each dataset in a different tab so you can flip back and forth.*

To do this, when viewing your data in Visual Preview, click on the magnifying glass on the left-hand side to open the dataset in FME Data Inspector. Repeat with other feature types.

Table	
VotingDivisions	
gml_id	gml_parent_id
1 i42a483603-57a...	<missing>
2 idd7ae6cc6-381...	<missing>
3 idc1449819-747...	<missing>
4 id6d87e2c8-9e4...	<missing>
5 10400004005-051...	<missing>
6 idf1507-1-1...	<missing>

Notice that both datasets have a Division attribute by which to identify each voting division (area). The Excel data is non-spatial but has a set of other voting attributes:

- **Voters**: Number of registered voters
- **Votes**: Number of voters who voted
- **Blanks**: Number of voters who left a blank or spoiled vote
- **OverVotes**: Number of voters who voted for too many candidates
- **UnderVotes**: Number of votes not cast

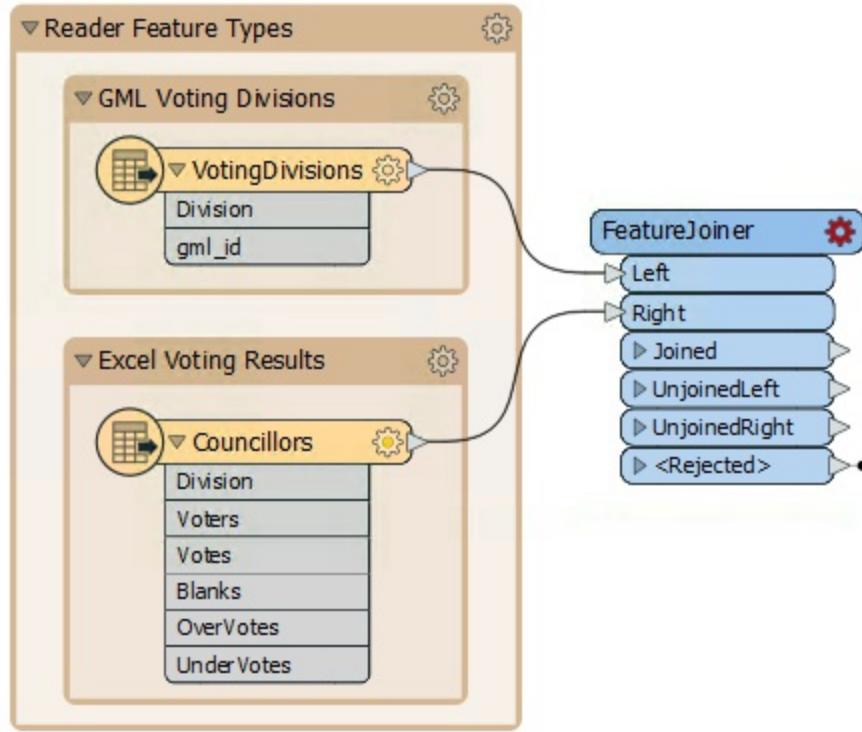
The OverVotes and UnderVotes attributes are an indicator of how well the voting process was understood. Each voter gets to vote for up to 10 candidates (out of 30).

OverVotes are those voters who voted for more than ten candidates. UnderVotes are the number of votes that could have been cast, but were not; for example, the voter only voted for four candidates instead of ten, giving six undervotes.

## 2) Add a FeatureJoiner Transformer

The first task is to join the statistical election data onto the actual features. We'll use a FeatureJoiner transformer to do this. A FeatureJoiner is a way to join or merge features. In this case, we are merging election result records onto election boundary features.

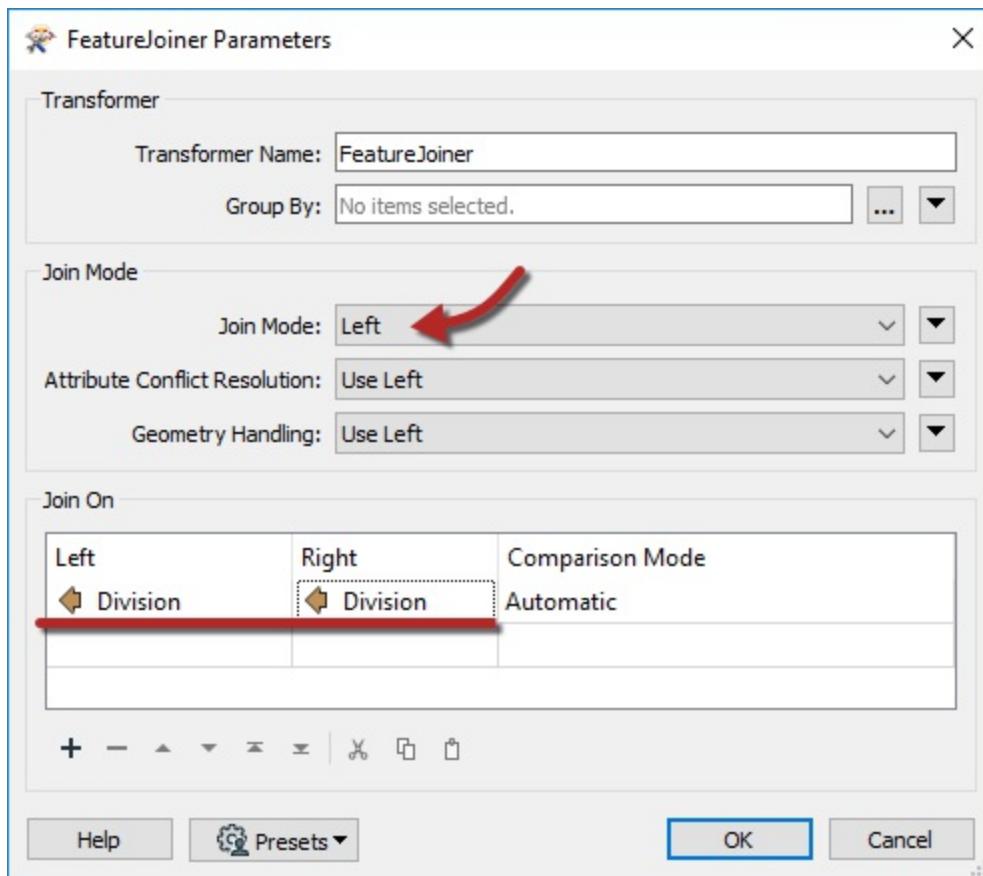
Add a FeatureJoiner transformer. Connect the VotingDivisions data to the Left port, and the Councillors (result) data to the Right port:



### 3) Set the FeatureJoiner Parameters

View the FeatureJoiner parameters. Because we want all of the voting division features we will do a Left join; therefore set the Join Mode to Left.

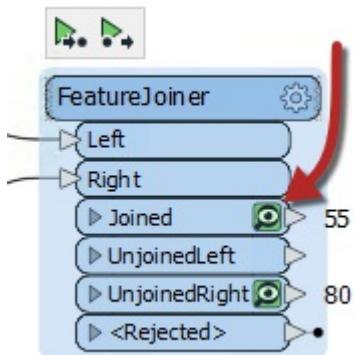
For both the Left and Right join fields, click in the field and choose the `Division` attribute from the drop-down list. This attribute is the common key by which we join our data:



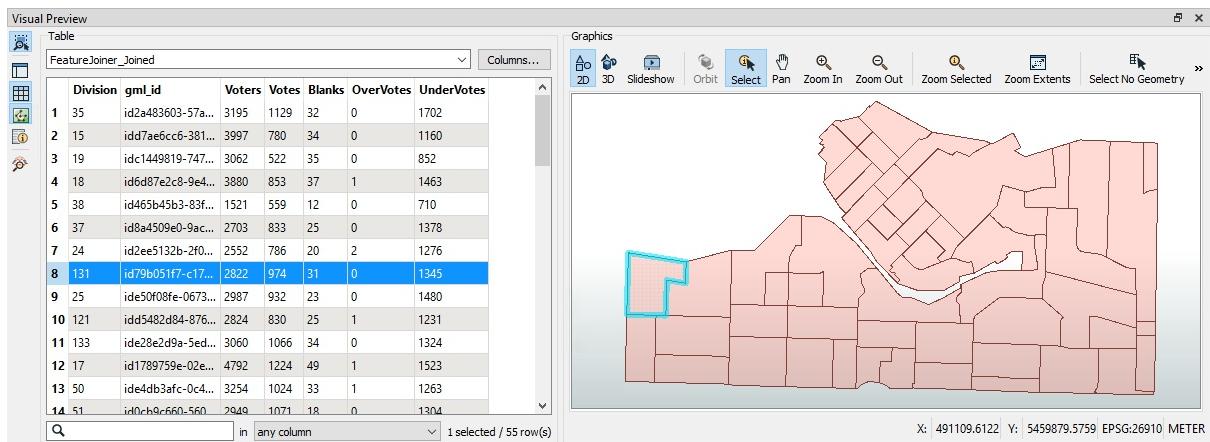
#### 4) View Data

Run the translation to create a feature cache, ignoring any warning or log message that reports Unexpected Input.

Then click on the FeatureJoiner:Joined output port to view the newly joined data in Visual Preview:



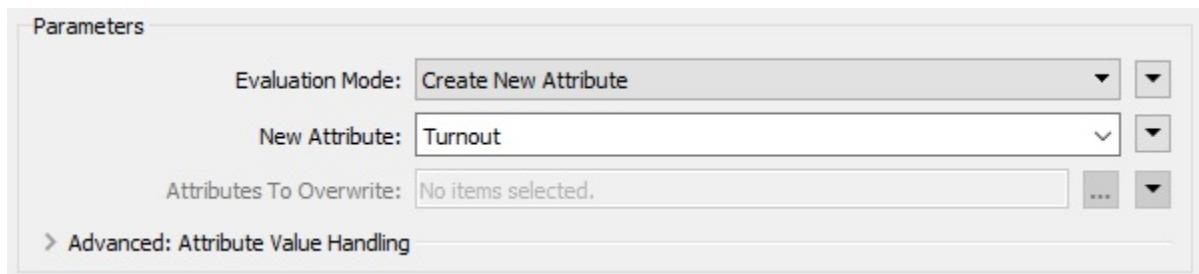
Examine the data to ensure all division polygons now include a set of attribute data copied from the Excel spreadsheet:



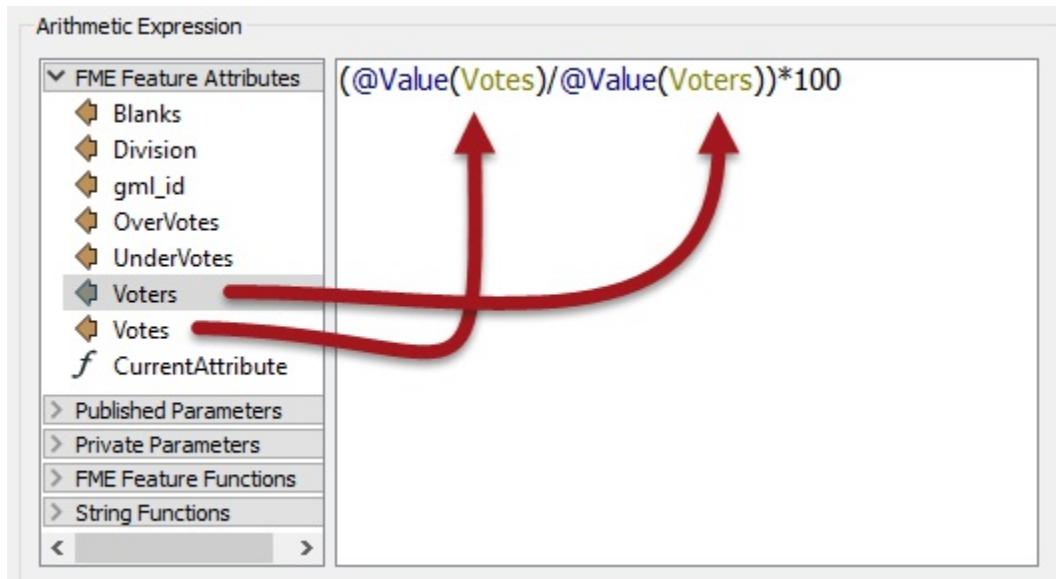
## 5) Add an ExpressionEvaluator Transformer

Now that we have the numbers we need, we can start to calculate some statistics. To do this, we'll use an ExpressionEvaluator transformer first to calculate the voter turnout percentage for each division.

Place an ExpressionEvaluator transformer after the FeatureJoiner - connect it to the FeatureJoiner:Joined output port. View the transformer's parameters. Set the New Attribute to Turnout (to match what we have on the destination schema):



In the Arithmetic Expression section, set the expression to:



You don't need to type the `@Value(Votes)` and `@Value(Voters)` part in, it can be obtained by double-clicking on the attributes in the list to the left under FME Feature Attributes.

If you wish, you can run the translation again, and it will only run the ExpressionEvaluator as everything before it is already cached. Then click on the Output port to view the results in the Visual Preview window.

## 6) Add another ExpressionEvaluator Transformer

Using a similar technique, add a second ExpressionEvaluator to calculate the number of UnderVotes per voter and put it in an attribute that matches the output schema which will be UnderVoting. Set the expression to:

```
@Value(UnderVotes)/@Value(Voters)
```

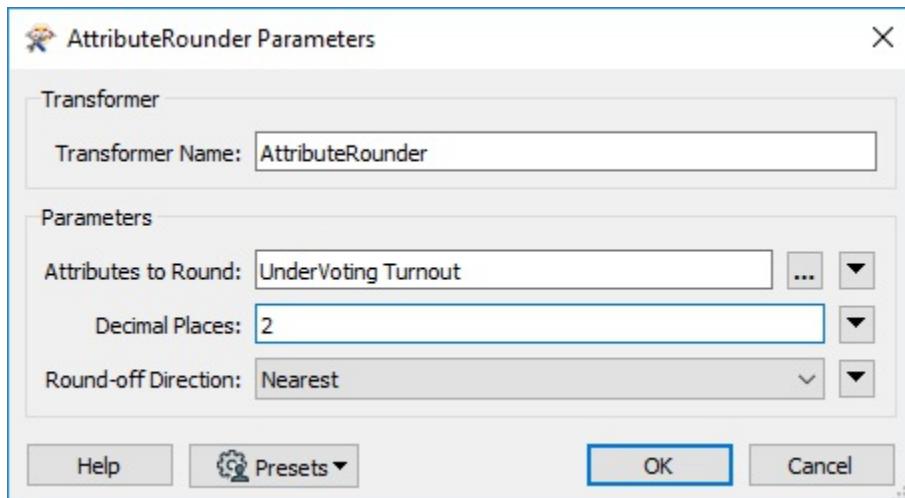
**Note:** This isn't a percentage, like the previous calculation.

Feel free to add a bookmark around the two ExpressionEvaluator transformers.

## 7) Add an AttributeRounder Transformer

It's a bit excessive to calculate our statistics to 13 decimal places or more. We should truncate these numbers a bit. To do this place an AttributeRounder transformer after the second ExpressionEvaluator.

For the parameters, under Attributes to Round select the newly created Turnout and UnderVoting attributes. Set the number of decimal places to 2:

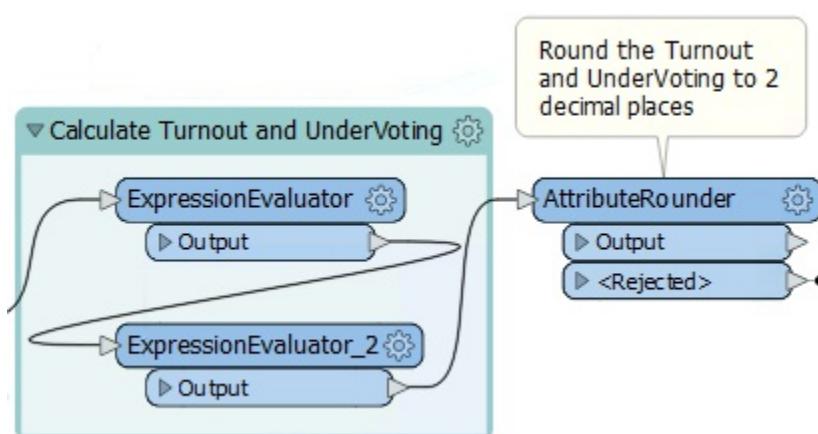


Again, run the workspace to check the results if you wish.

## 8) Add an Annotation

It doesn't make sense to add a bookmark to the AttributeRounder by itself, so instead right-click the transformer and choose the option to Attach Annotation.

Doing so will add a label to the transformer. Edit the content to say something like "Round the Turnout and Undervoting to two decimal places":

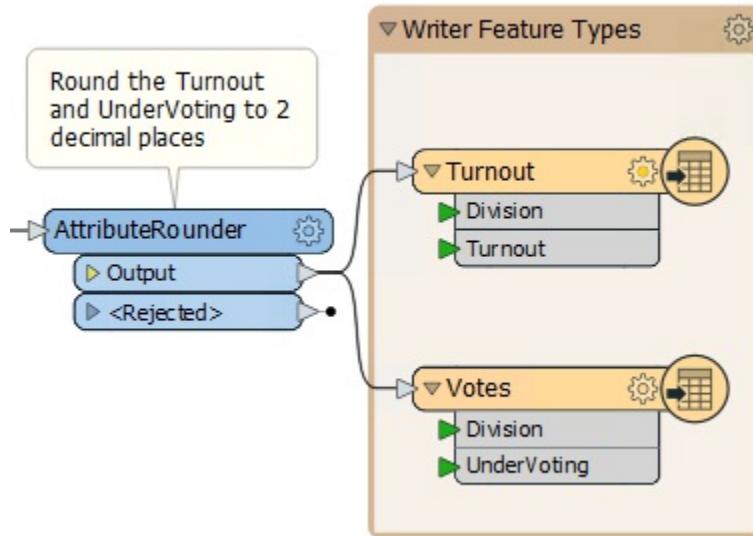


## TIP

Annotations will be covered in more detail in Chapter 5: Best Practice, but that shouldn't stop you from creating well documented workspaces now!

### 9) Connect the Schema

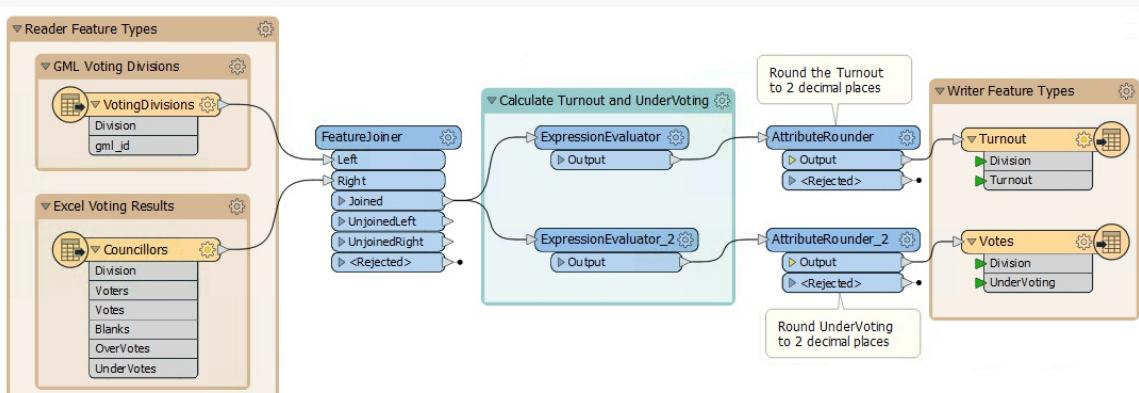
For the final step let's connect the AttributeRounder to the output schema. Simply make connections from the AttributeRounder to both writer feature types:



Run the workspace and examine the output in Google Earth to prove it has the correct attributes and is in the correct location.

## TIP

So this is obviously a form of parallel streams of data, but left until the last step. An alternative layout would be to split the data after the FeatureJoiner, like so:



There's no difference in the output so the only consideration is which is easier to create and which will perform better. The main method used should win on both counts.

## Advanced Exercise

The project is done, but the output is very plain. It would be much better to improve the look of the results and there are several ways to do this with KML.

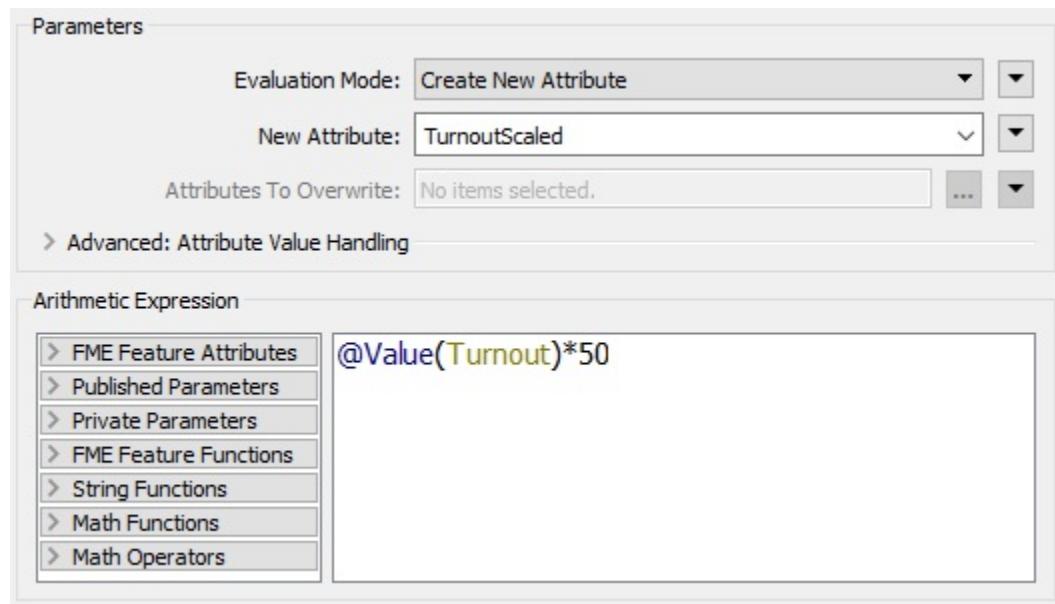
We could simply color the voting divisions differently according to their turnout/overvotes, but a more impressive method is to use three-dimensional blocks.

Follow the steps below to create three-dimensional shapes in the output KML dataset...

### 10) Add a Third ExpressionEvaluator Transformer

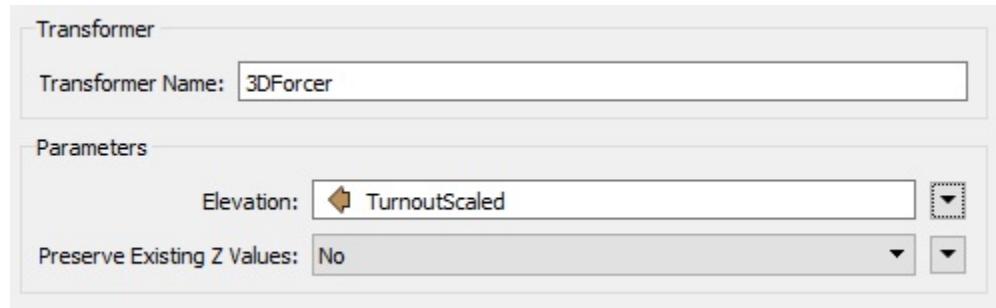
The height of each block should be proportional to the turnout for that division. However, for differences to be visible, the vertical scale will need some exaggeration.

Place an ExpressionEvaluator between the AttributeRounder and the Turnout feature type. Set the parameters to multiply the Turnout attribute by a value of 50. Put it into a new attribute called TurnoutScaled.



### 11) Add a 3DForcer Transformer

Add a 3DForcer transformer after the new ExpressionEvaluator. This will elevate the feature to the required height. In the parameters dialog set the elevation to Attribute Value > TurnoutScaled.

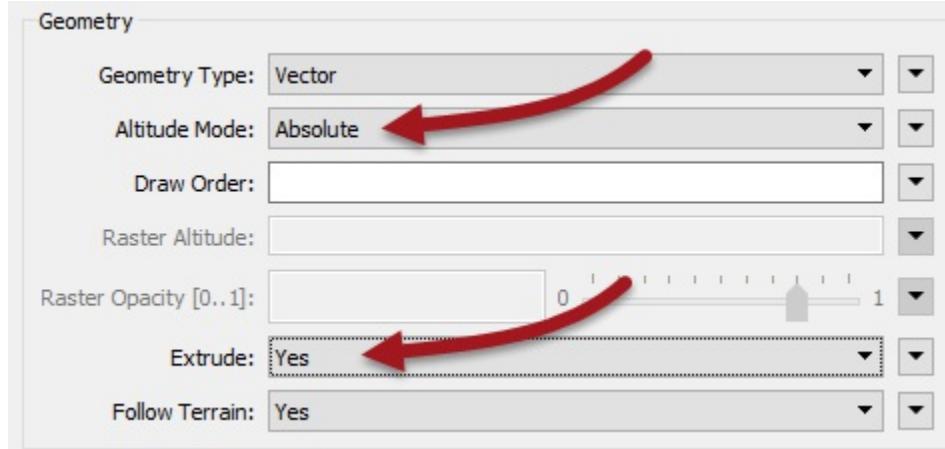


### 12) Add a KMLPropertySetter Transformer

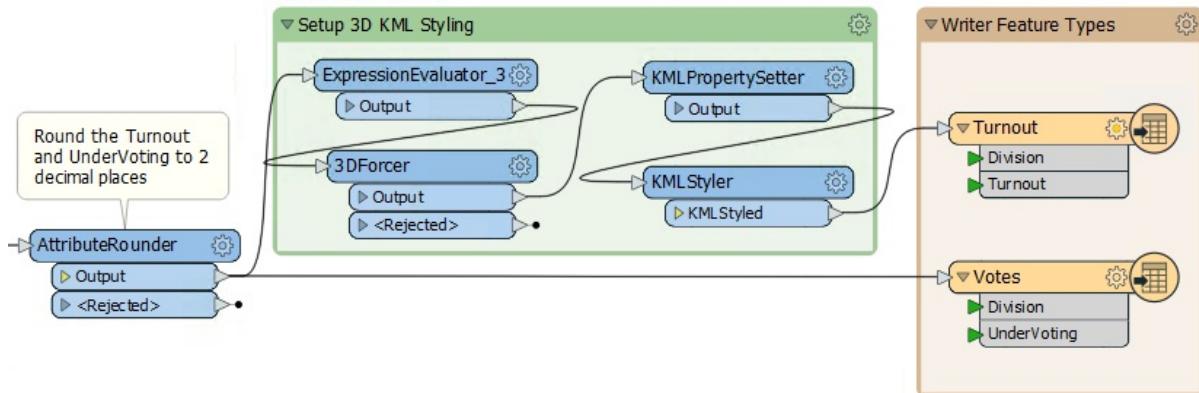
Add a KMLPropertySetter transformer after the 3DForcer. This transformer will allow us to set up the 3D blocks in

the output. Set the geometry parameters as follows:

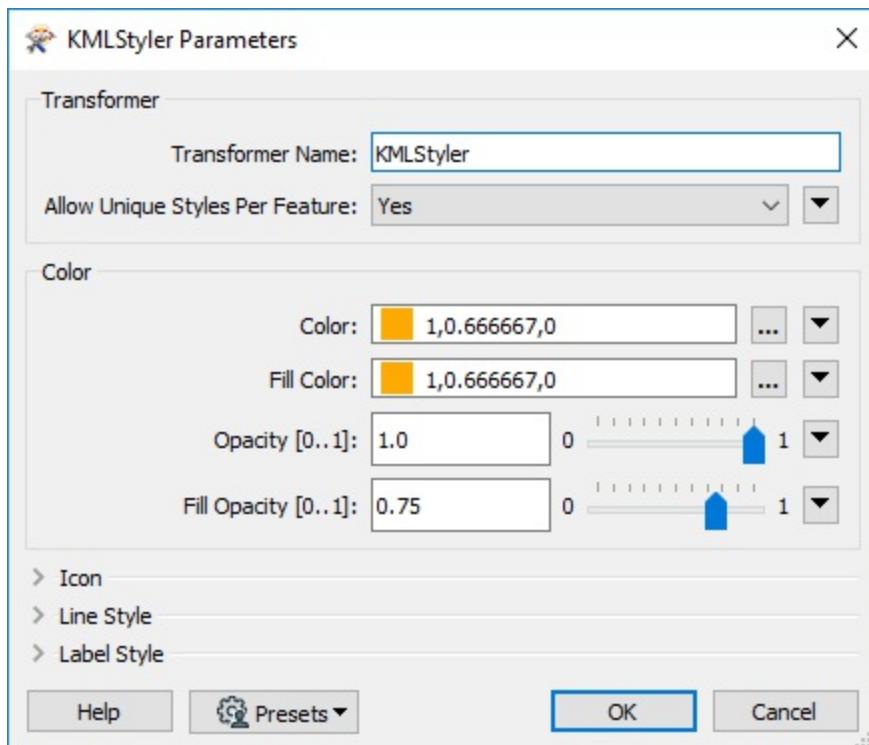
- Altitude Mode: Absolute
- Extrude: Yes



**13) Add a KMLStyler Transformer** Finally, add a KMLStyler transformer. The workspace will now look like this:



Check the parameters. Select a color and fill color for the features. Increase the fill opacity to around 0.75.



Save and run the workspace. In Google Earth the output should now look like this:



These 3D blocks will show users where the voter turnout is high/low in the city.

If you wish, repeat these steps to give a 3D representation to the UnderVoting statistics.

## CONGRATULATIONS

*By completing this exercise you proved you know how to:*

- *Carry out content transformation with transformers (ExpressionEvaluator, AttributeRounder, 3DForcer)*
- *Use transformer parameters to create attributes that match the writer schema*
- *Use multiple streams of transformers in a single workspace*

*You also learned how to:*

- *Merge multiple streams of data using a common key (FeatureJoiner)*
- *Use FME's built-in maths editor dialog*
- *Use transformers to set a symbology (style) for output features*

## Workspace Design

A basic workspace typically reads in data, transforms it then writes it out.

However, there are other methods for constructing more advanced workspaces, and for directing the flow of data through a workspace in unique ways.



Some example uses for these techniques might be:

- To design large-scale workspaces a small section at a time
- To read data from multiple formats within a single workspace
- To carry out actions *after* a dataset has been written
- To use data stored on web services
- To test run individual parts of a workspace

## Prototyping

In the classical sense, prototyping means creating an incomplete application as a way to evaluate the feasibility of a project.

Here we'll stretch the definition to mean how to build a complex FME project incrementally; starting with an empty workspace and building it piece by piece to deliver a result that matches the final specification.

We will cover the techniques used for building a workspace incrementally, and how to handle data that is rejected by a transformer.

## Incremental Development

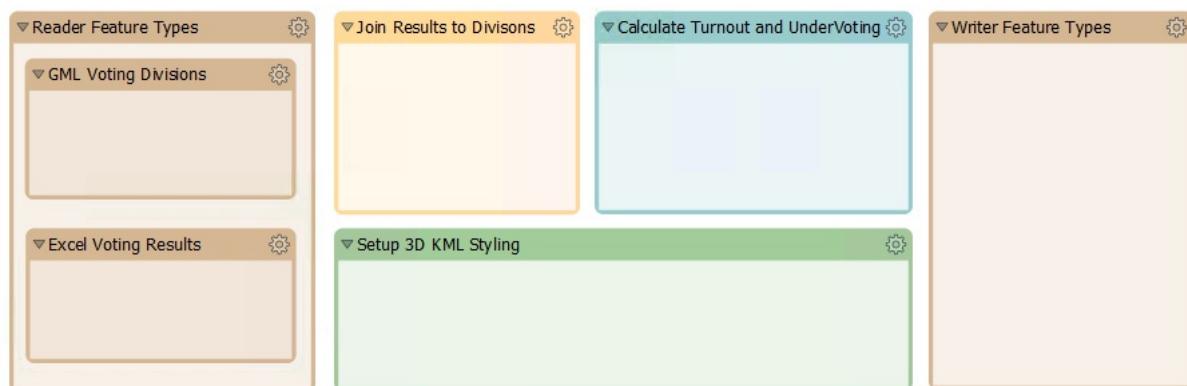
The key development technique for FME workspaces is [incremental updates](#).

The steps to this technique are:

- Plan your project as a series of small sections, each of which would fit inside a bookmark in FME
- Design and implement a section in FME Workbench. It should ideally be between 3-10 transformers.
- Test each section immediately after it is completed. That way you can identify problems at an early stage, and identify them more easily because only a few changes are being made in any increment.
- Repeat the process, saving the workspace and testing it whenever you've added a new section

Although a range of 3-10 transformers is an arbitrary number, the more transformers you add, the more difficult it would be to identify the source of any problems. Beyond ten transformers is the point at which you should consider chopping that process into smaller sections.

Here an author has planned their workspace by laying it out as a set of bookmarks on the canvas:



Now the author can complete and test each section at a time, keeping the overall goal in mind at the same time.

### WARNING

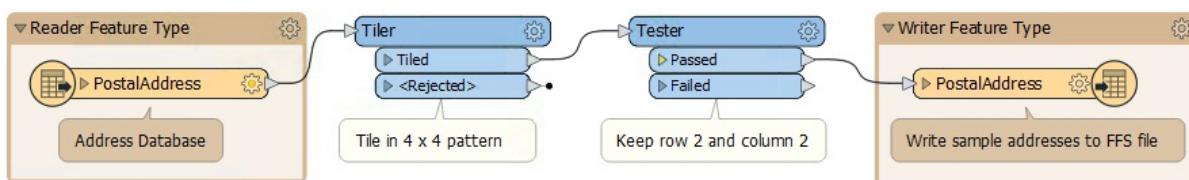
*It can be all too easy to start developing a workspace and forget to save it at all! FME keeps a recover file as soon as the workspace is saved for the first time, but until then you are running the risk of an irretrievable loss.*

## Source Data

When the FME project is large and complex, it's likely that the source data will be large and complex too. So when creating a workspace in small increments, testing each part in turn, it's better to avoid using the entire dataset.

It's better to use a sample of source data for testing. In fact, it's better to create a small sample of data - extracting it from your source and writing to a neutral format like FFS - rather than randomly sampling the data for each test run.

Sampling is particularly useful for databases because it also avoids the problems of waiting for network traffic and database responses.



Here the workspace author is extracting a section of source data by reading from a database, splitting it into tiles, and writing just one tile to the FFS format. This one tile can be used for prototyping a solution in a way that is representative of the entire source database table.

Another transformer to use would be the Sampler, although the features selected by it would not be spatially adjacent.

---

## Version Control

When making a set of incremental changes to a workspace, it's easy to work on a single workspace file only. However, there are various problems with this:

- Faults cannot be easily tracked because there is no record of what has changed and when
- It is not easy to create different versions for different platforms
- If the workspace file is lost or corrupted, then the entire project is lost

Therefore, it is better to keep versioned workspaces, where a different copy is kept for each set of revisions. This precaution can be taken manually within the file system, or by using a repository tool like Git.

In fact, it is a good idea to keep and version all materials related to an FME project, including:

- Workspace files
- Python files
- Log Files
- Source Datasets

It's better not to store any information that is personal or that includes passwords. Also, there's no need to store temporary files.

---

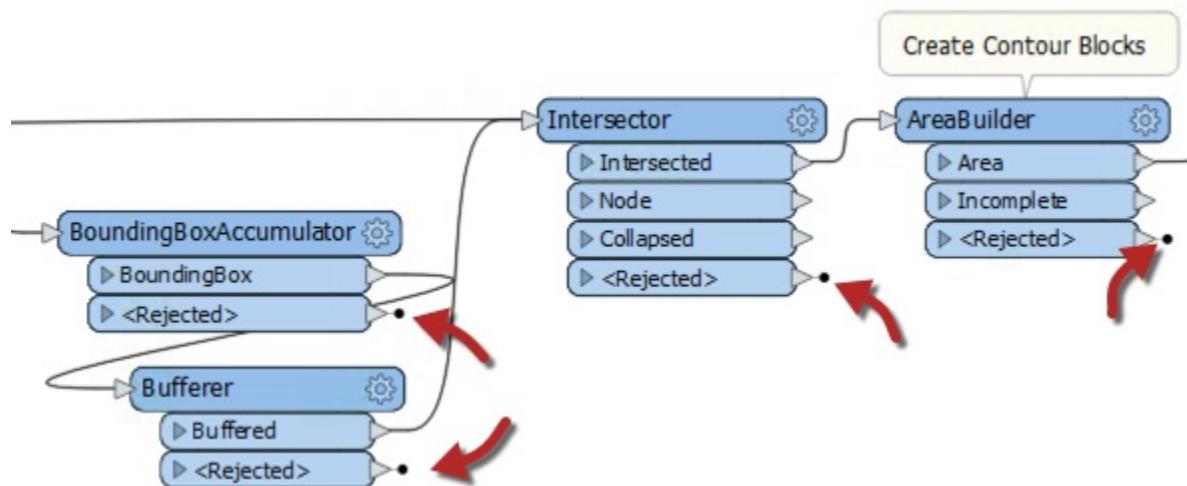
### TIP

*FME Server has a version control system built in. Even if you don't have an FME Server license, you can still install it for use as a repository system for workspaces and related files.*

## Rejected Features

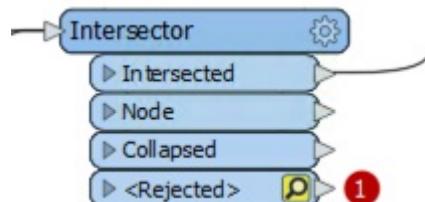
An important part of any workflow is handling data that fails to process. For example, where a feature with no geometry is sent into a geometry-based transformer like the AreaBuilder.

FME handles such failures by outputting the data through <Rejected> ports, which are found on many transformers:



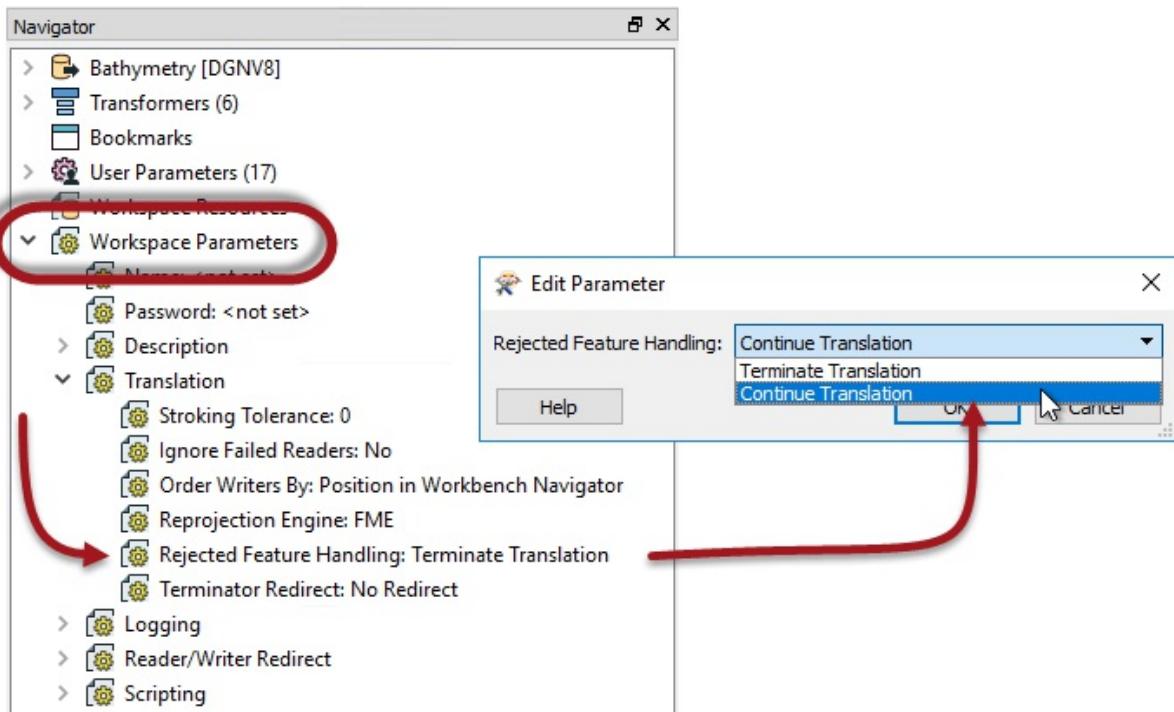
A parameter exists to control the action of <Rejected> ports and gives the workspace author a choice over what action to take.

When a feature is rejected, the translation will stop and a red circle with a number will appear on the <Rejected> port. You can click on the cache to inspect the feature and determine why it was rejected:



## Rejected Feature Handling

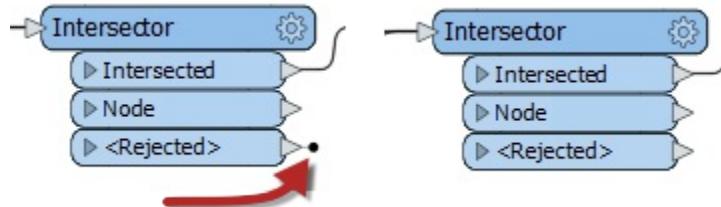
The parameter to control handling of rejected features can be found in the Navigator window, under Workspace Parameters:



The two options are *Terminate Translation* and *Continue Translation*.

When the parameter is set to *terminate*, then a feature that exits via a <Rejected> port causes the translation to stop. To visually denote this, the <Rejected> ports have a small black marker on them.

When the parameter is set to *continue*, then the translation will continue, regardless of how many features exit <Rejected> ports. In that case, the small black marker is removed:



In *terminate* mode, a rejected feature gets written to the log window with the error message:

**The below feature caused the translation to be terminated**

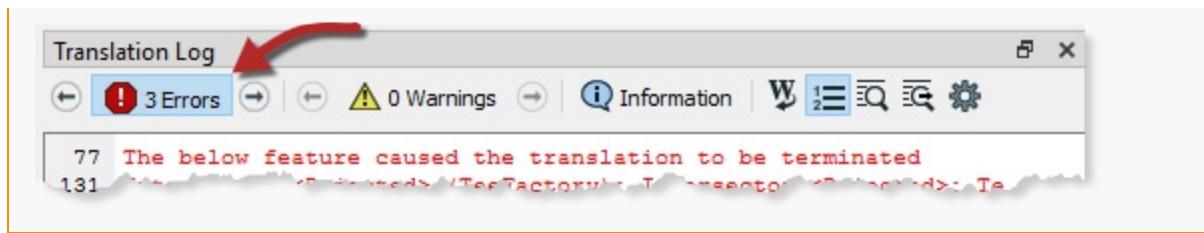
There will also be an error message relating to the transformer:

**Intsector\_<Rejected>(TeeFactory): Intsector\_<Rejected>: Termination Message: 'Intsector output a <Rejected> feature.'**

This error is useful because it tells the author which transformer experienced the failure.

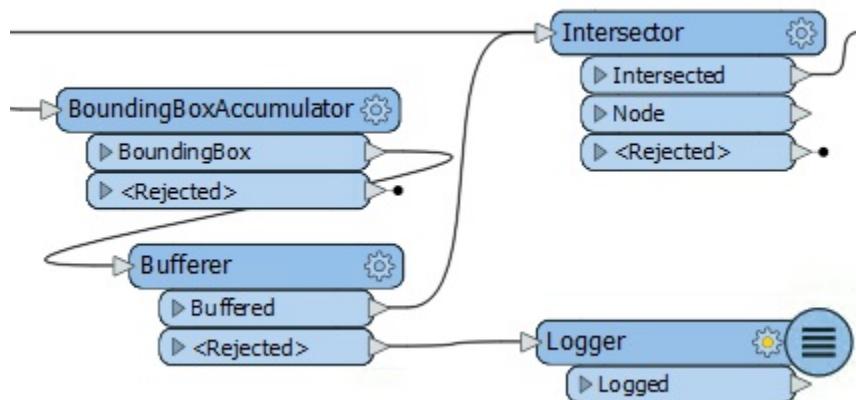
## TIP

To quickly find these error messages you can filter the Translation Log by clicking on the Errors button.



## Mixed Mode

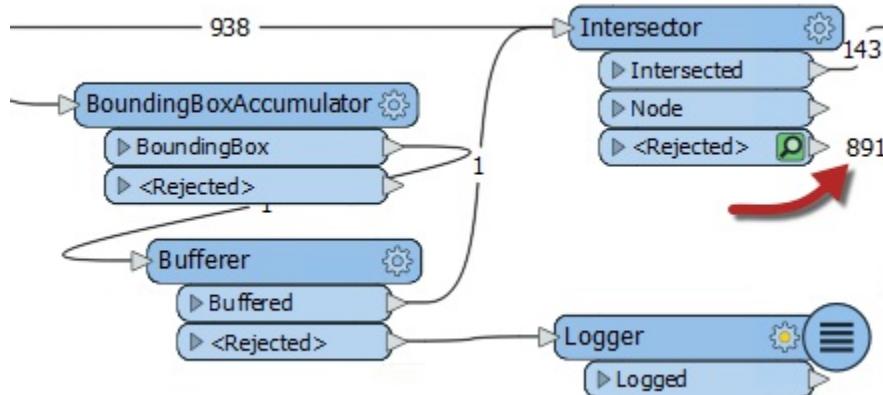
In *terminate* mode, a rejected feature will cause the translation to stop, provided that the <Rejected> port is connected to a further object:



In short, an author can create a mixed mode, where some transformers stop the translation on rejecting a feature (the **Intersector** above), but others will handle the feature another way (the **Bufferer**). That way the author can try to handle rejected features that are expected, but stop the translation if there are truly unexpected failures.

## Feature Counts and Inspection

In continue mode, features that exit a <Rejected> port are counted and saved for inspection:



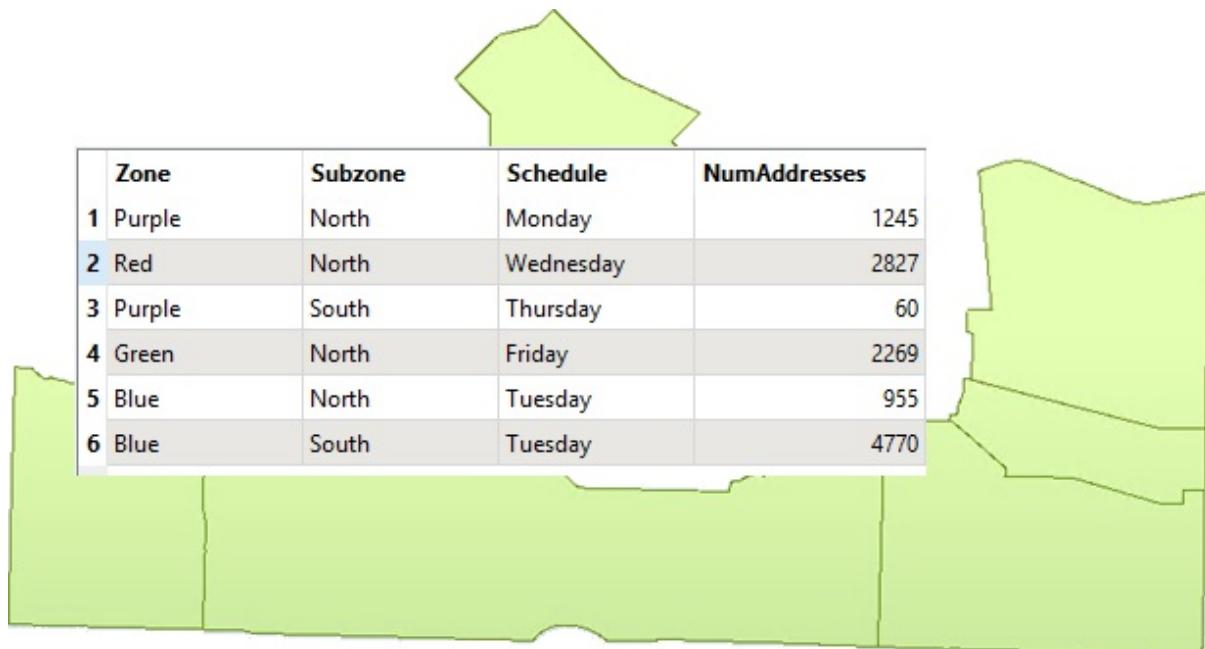
Features will be saved for inspection even if there is no **Logger** or other transformer attached. The number tells us how many features were rejected and the green icon can be clicked to inspect the data.

## Exercise 1

## Residential Garbage Collection Zones

<b>Data</b>	Addresses (Esri Geodatabase), Zones (MapInfo TAB)
<b>Overall Goal</b>	Create boundaries for residential garbage collection
<b>Demonstrates</b>	Workspace prototyping
<b>Start Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\Design-Ex1-Begin.fmw
<b>End Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\Design-Ex1-Complete.fmw

The city maintenance department has a dataset of garbage collection schedules, to assign residents to a collection on a particular day:



However, because of shifting demographics and zoning changes, they have decided that new boundaries should be drawn.

Your task is to use FME to create new boundaries. You must create five polygons, adjacent to each other, and with approximately the same number of residents in each. The analysis will be based on the city's address database. An estimate of the number of residents per address will be created depending on the zone type it falls within:

- Single-family residences: 2 adults
- Two-family residences: 4 adults
- Multi-family residences: 12 adults
- Comprehensive development zone: 8 adults
- Commercial properties: 1 adult

The output format shall be OGC GeoPackage.

To develop this workspace, it's necessary to consider what different steps might be required. We can then create sections with a bookmark and fill them in as we go along.

### 1) Plan Workspace

Let's plan this workspace together.

We need to read the address data (or a sample of it) and write the output to OGC GeoPackage. We need to know what zone type each address falls inside, which needs zoning data and a transformer to carry out a spatial join.

We also need to create a resident count based on the zone type and then divide the residents into five different areas. Finally, we need to group the addresses with a boundary shape around them.

In short, we need this set of actions:

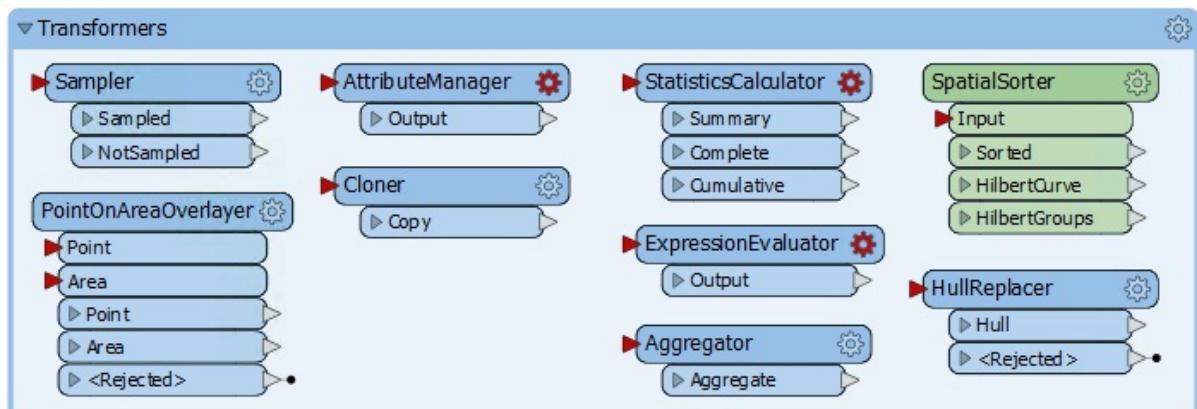
- Read/Sample Address database
- Read Zoning data
- Create a spatial join
- Calculate resident count
- Divide residents into five groups
- Aggregate the addresses into their group
- Create a boundary shape
- Write OGC GeoPackage

So, choose File > Open and select the beginning workspace

(C:\FMEData2019\Workspaces\DesktopBasic\Design-Ex1-Begin.fmw). It already has a set of bookmarks to represent these steps to be carried out but, as yet, we can't be sure which sections will be larger, so all bookmarks are the same size:



You'll also find a bookmark containing all of the transformers required for the exercise:

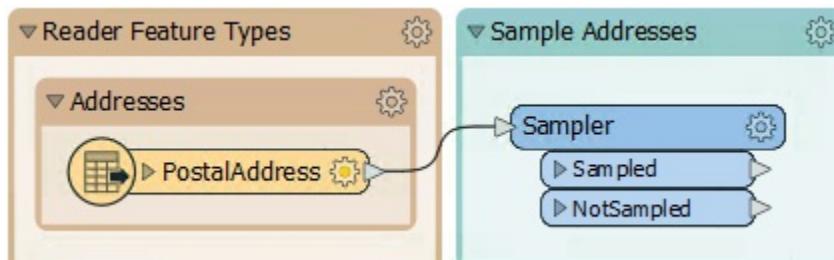


Or at least, these are the transformers we think will be required for the exercise!

## 2) Sample Source Data

There are more features in the address database than we need for workspace construction and testing, so let's reduce that to a smaller sample.

Rather than create a test dataset, here we'll use a Sampler transformer. There is a Sampler transformer in the "Transformers" bookmark, so simply move that transformer into the "Sample Addresses" bookmark and connect the PostalAddress feature type to it:



Inspect the Sampler's parameters. Set it to sample every 25th feature

Parameters

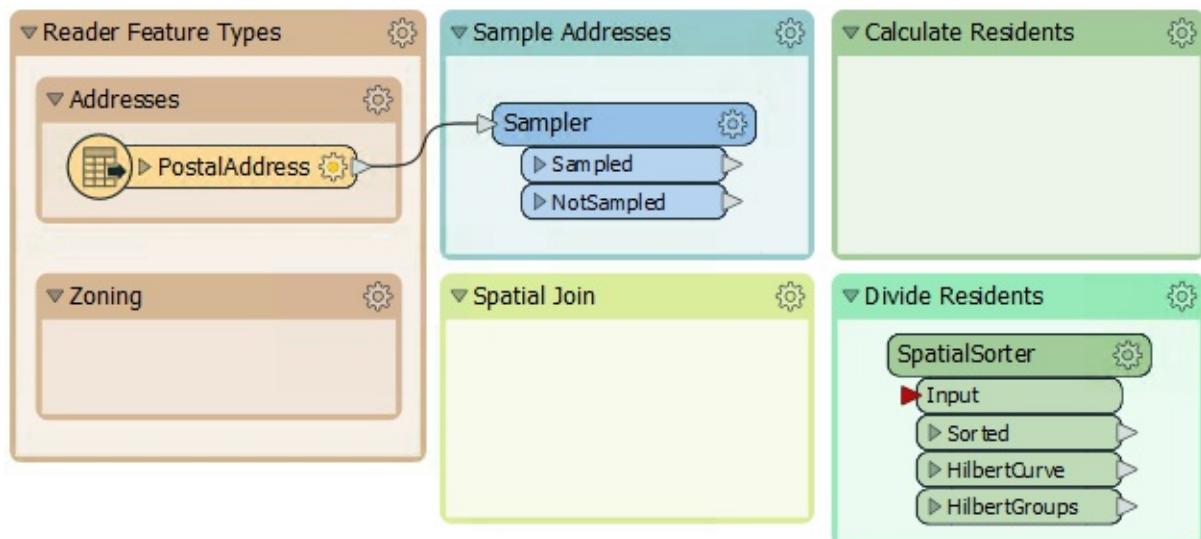
Sampling Rate (N):	25
Sampling Type:	Every Nth Feature
Randomize Sampling:	No

Run the workspace to be sure it is sampling the data correctly. Click on the magnifying glass on the Sampler:Sampled output port to view the data in the Visual Preview window. Out of 13,597 addresses, there should be 543 selected for use.

### 3) Divide Data into Groups

Before trying to add the Zoning dataset into the workspace, let's try and create groups from the basic dataset. We can do this with a custom transformer from the FME Hub, called the SpatialSorter.

So move the SpatialSorter from the "Transformers" bookmark to the "Divide Residents" bookmark:



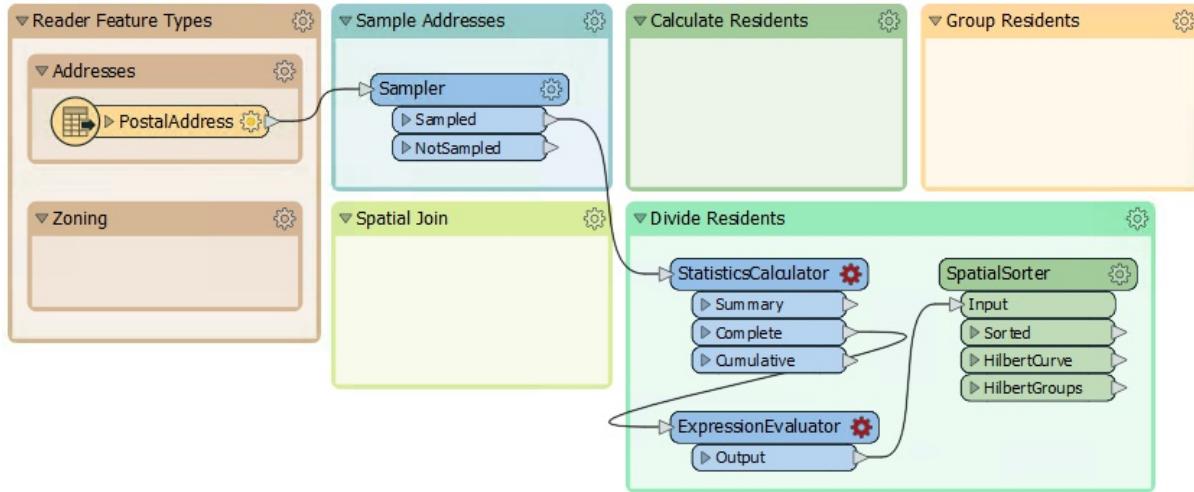
The SpatialSorter sorts data spatially (so features closer geographically become closer in the sorted output) and creates groups.

Check the parameters for this transformer. Notice that the group parameter asks for group size, not the number of groups. Therefore we'll need to calculate how many addresses there are when split into five groups.

### 4) Calculate Group Sizes

To calculate the number of addresses per group, we need the number of addresses and then divide that by five. We can do this with a combination of StatisticsCalculator and ExpressionEvaluator.

So, enlarge the Divide Residents bookmark as required and move the StatisticsCalculator and ExpressionEvaluator transformers from the "Transformers" bookmark. Connect them up to the Sampler:Sampled port like so:



## 5) Calculate Group Sizes

Inspect the parameters for the StatisticsCalculator. This transformer will tell us how many features there are (the Total Count). Pick an attribute for the Attributes to Analyze parameter. Because we only want to count features not create true statistics, it can be any attribute you like.

Under the Total Count Attribute parameter, enter the name TotalResidents. Erase the rest of the calculated attribute fields, so they are not calculated:

**Attributes to Analyze**

Attributes to Analyze: COUNTRY

Prepend Output Attribute Names: For multiple results only

**Statistics**

Minimum Attribute: (empty)

Maximum Attribute: (empty)

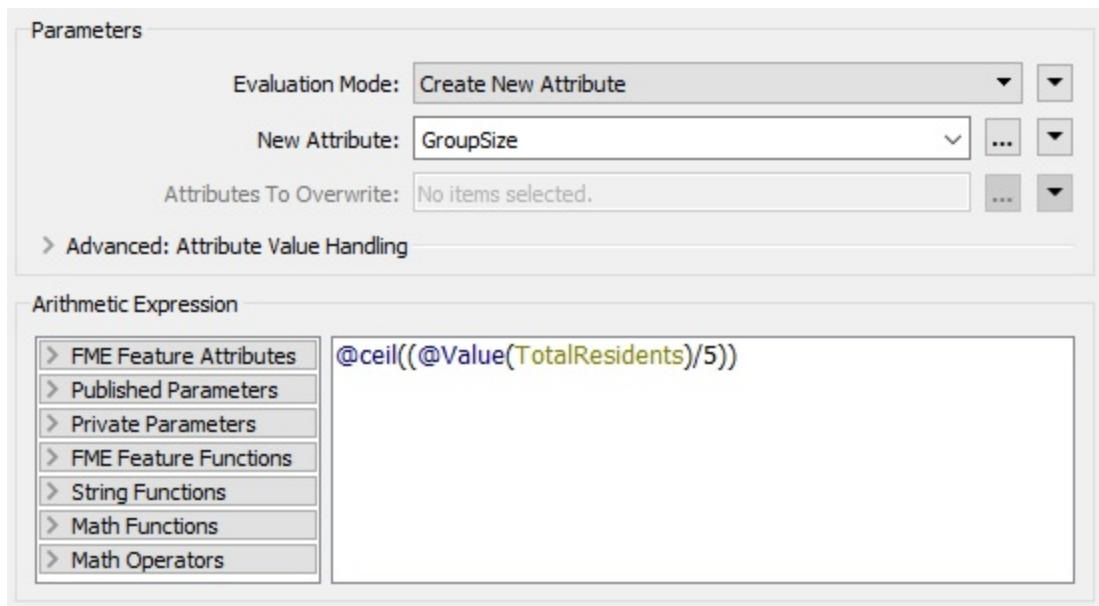
Total Count Attribute: TotalResidents

Sum Attribute: (empty)

Mean Attribute: (empty)

In the ExpressionEvaluator, enter GroupSize in the New Attribute parameter. In the Arithmetic Expression field enter the expression:

```
@ceil(@Value(TotalResidents)/5))
```



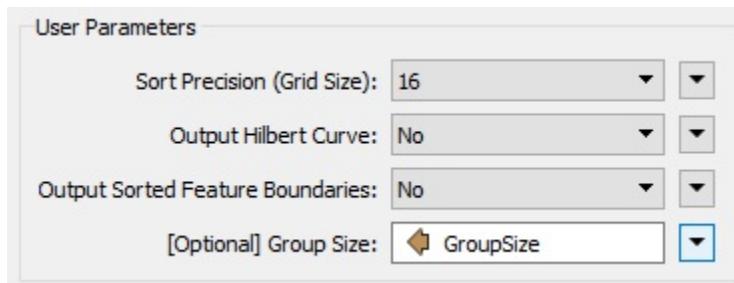
The simplest way is to double-click the ceil function to add it, then double-click the TotalResidents attribute, and manually add the /5 part.

This expression will divide the number of residents into five groups, rounding up. The rounding up part is essential, and it's what the ceil function does.

Run the translation and view the ExpressionEvaluator output in the Visual Preview window to provide that this part works. The TotalResidents should be 543 and the GroupSize should be 109 for each feature.

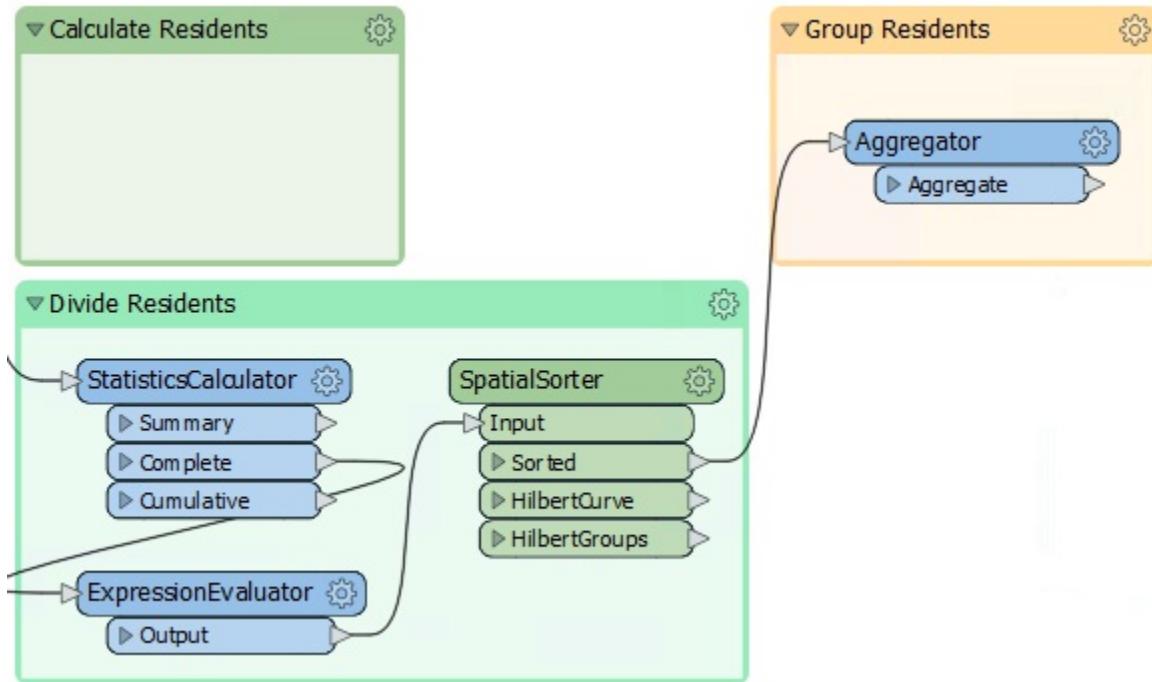
## 6) Group Residents

Now inspect the parameters for the SpatialSorter once more. We will leave Grid Size at 16 for now; this will give us a more coarse result but will run faster while we develop the workspace. Under the Group Size parameter, click the drop-down arrow and select Attribute Value > GroupSize:

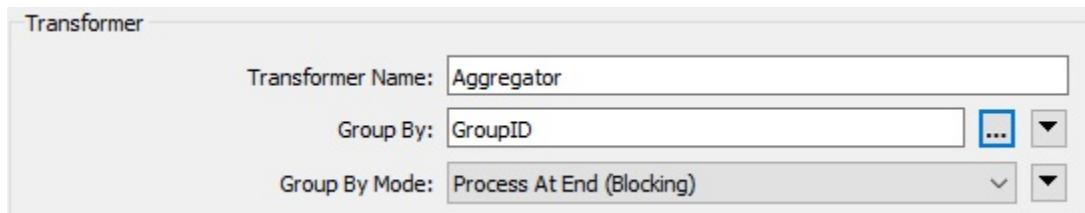


This sets the group size to the attribute just calculated.

To create groups of addresses, move the Aggregator transformer to the "Group Residents" bookmark, and connect it to the SpatialSorter:Sorted output port:



Inspect the parameters for the Aggregator. Set the Group By parameter to the GroupID attribute (in other words, aggregate features together in the groups created by the SpatialSorter):



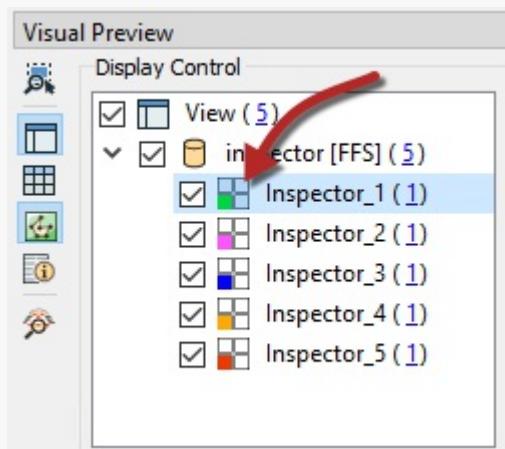
Using an Inspector transformer we can also set the Group By parameter, which will result in the data being represented with different colors for each group in the Visual Preview window. Connect an Inspector to the Aggregator:Aggregate output port and then in the parameters set the Group By to GroupID. Run the translation, and you should find there are five sets of point aggregates in the output, each of which has approximately the same number of point features:



## TIP

The Inspector transformer can be used to inspect data, but with Visual Preview and feature caching there isn't much reason to use it often. This case, where you want to add a group-by to help visualize your data in groups, is one of the cases where this transformer is useful.

You can change the color of the groups by clicking on the Toggle Display Control button on the left-hand side of visual preview. Then double-click on the four square icon to open the Geometry Drawing Styles dialog, where you can set the color. When the color is manually set, the four square icon will display the color.



Save the workspace, naming it GarbageCollection.fmw or something similar. Ideally, the name would include a version number (like GarbageCollection-v1.fmw), but it's not necessary for this exercise.

The next step in the workspace will be to add in the Zoning data, create a spatial join, and calculate how many residents live in each property based on each address' zoning type.

## CONGRATULATIONS

By completing this exercise you have learned how to:

- Plan a workspace development
- Create a workspace, section by section
- Restrict source data to a small sample
- Use a custom transformer from the FME Hub
- Carry out an arithmetic calculation
- Aggregate data into groups
- Save a workspace with a version number

## Reading and Writing Workflows

By default, the Generate Workspace dialog creates a workspace with a single reader and a single writer. However, this does not mean the workspace is forever limited to this. Additionally, FME can read/write data using:

- Multiple readers, multiple writers, or both
- FeatureReader and/or FeatureWriter transformers (*Covered in the next chapter*)
- Integration transformers such as the DropboxConnector, Email, HTTPCaller, etc.

## Workspace Components

A workspace is the primary element in an FME translation and is responsible for storing a translation definition. Think of a workspace as the container for all the functionality of a translation, which is stored in the following components:

### Readers and Writers

A **reader** is the FME term for the component in a translation that reads a source dataset. Likewise, a **writer** is the component that writes to a destination dataset.

Readers and writers are represented by entries in the Navigator window.

### Feature Types

**Feature type** is the FME term that describes a subset of records. Common alternatives for this term are *layer*, *table*, *sheet*, *feature class*, and *object class*. For example, each layer in a DWG file, or each table in an Oracle database, is defined by a feature type in FME.

Feature types are represented by objects that appear on the Workbench canvas.

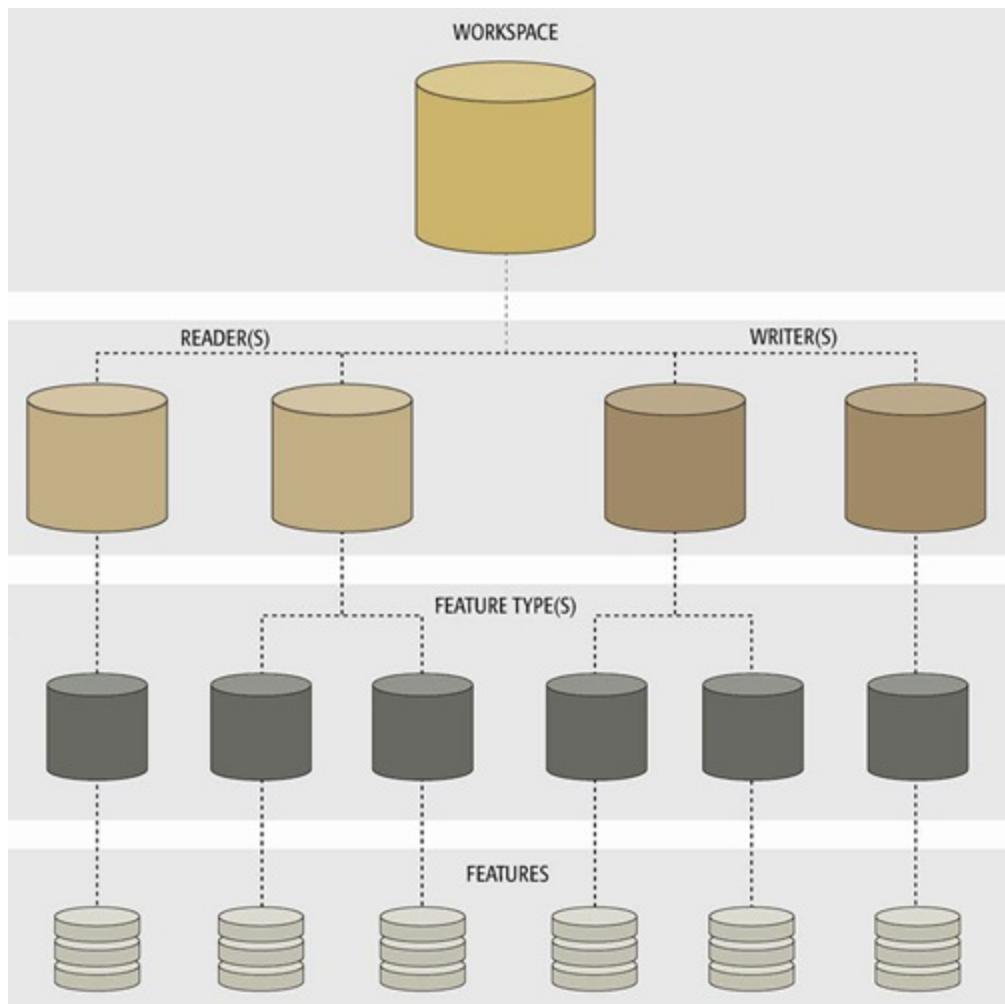
### Features

**Features** are the smallest single components of an FME translation.

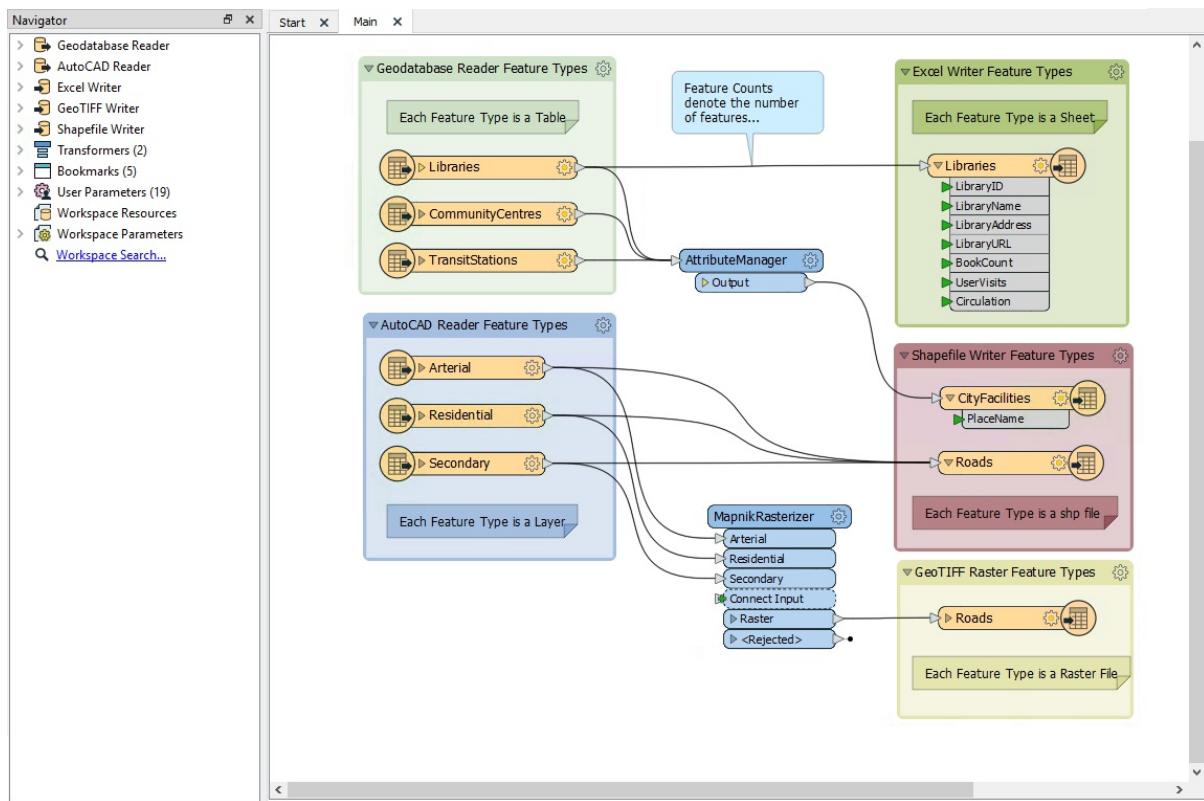
They aren't individually represented within a workspace, except by the feature counts on a completed translation.

### Relationships

Each workspace can contain multiple readers and writers, each of which can have multiple feature types, with multiple features. They exist in a hierarchy that looks like this:



A workspace with multiple readers and writers might look like this:

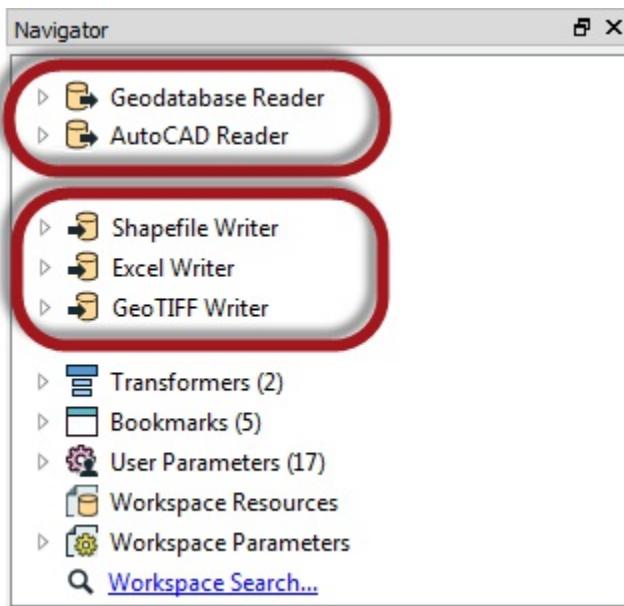


This workspace has two readers (each with three feature types), and three writers (with one, two, and one feature types). Each reader and writer is a different format, and each has a different name for its feature types.

## Multiple Readers and Writers

An FME workspace is not limited to any particular number of readers or writers; readers and writers can be added to a workspace at any time, any number of formats can be used, and there does not need to be an equal number of readers and writers.

For example, the Navigator window shows this workspace contains two readers and three writers, of different data types and formats!



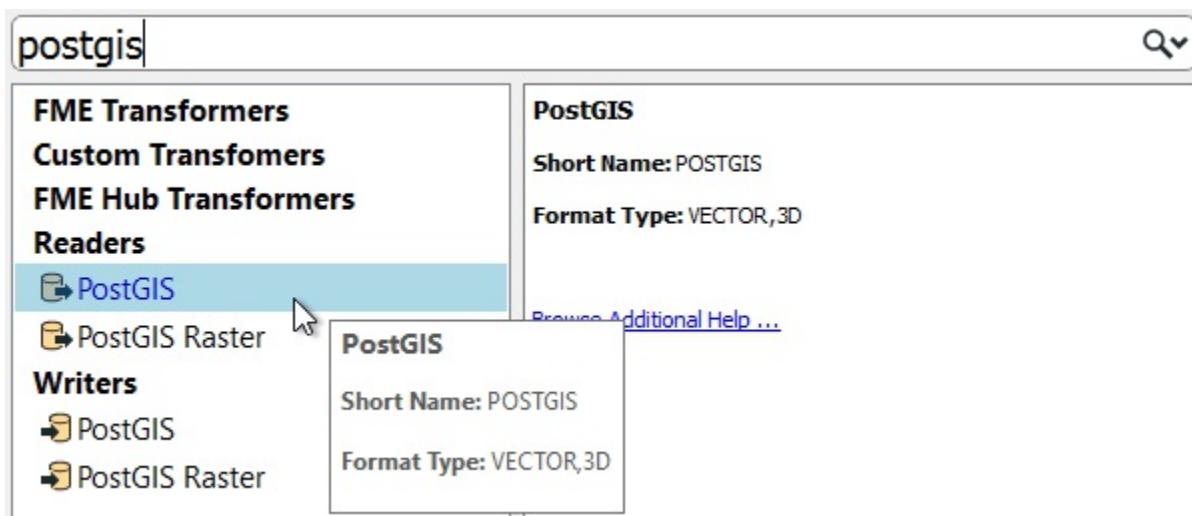
### FME Lizard says...

*It's important to note that readers and writers don't appear as objects on the Workbench canvas. Their feature types do, but readers and writers don't.*

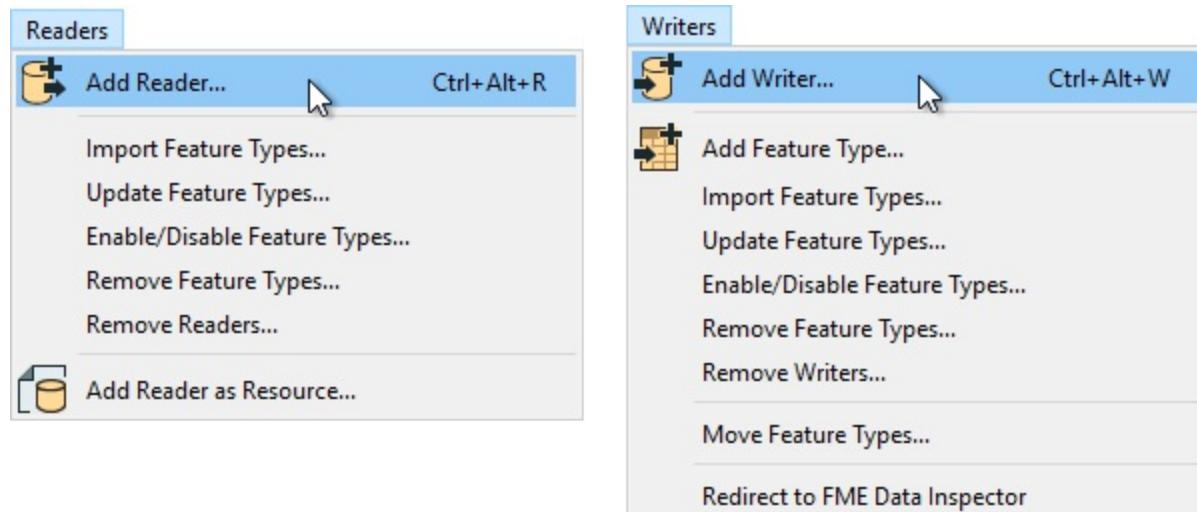
*Instead they are represented by entries in the Navigator window, as in the above screenshot.*

## Adding Readers and Writers

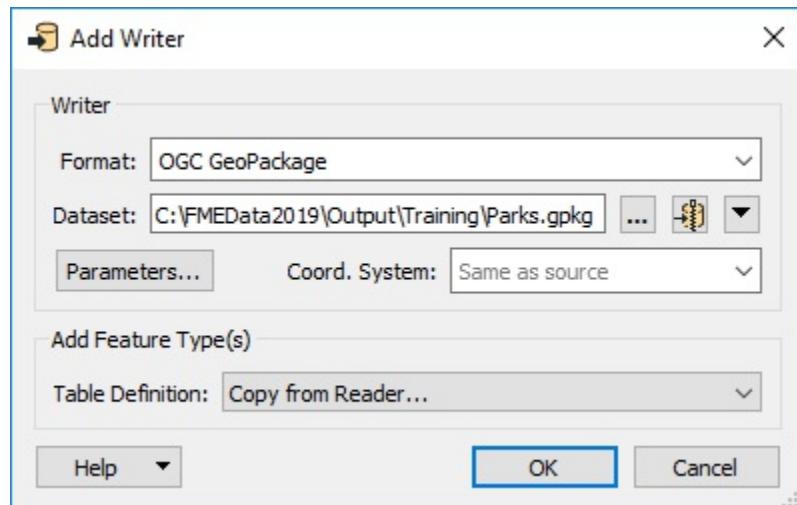
Additional readers or writers are added to a translation using the Quick Add menu:



...Or by selecting Readers > Add Reader (Writers > Add Writer) from the menu bar:



This action opens a dialog, similar to the Generate Workspace dialog, in which the parameters for the new reader or writer can be defined:



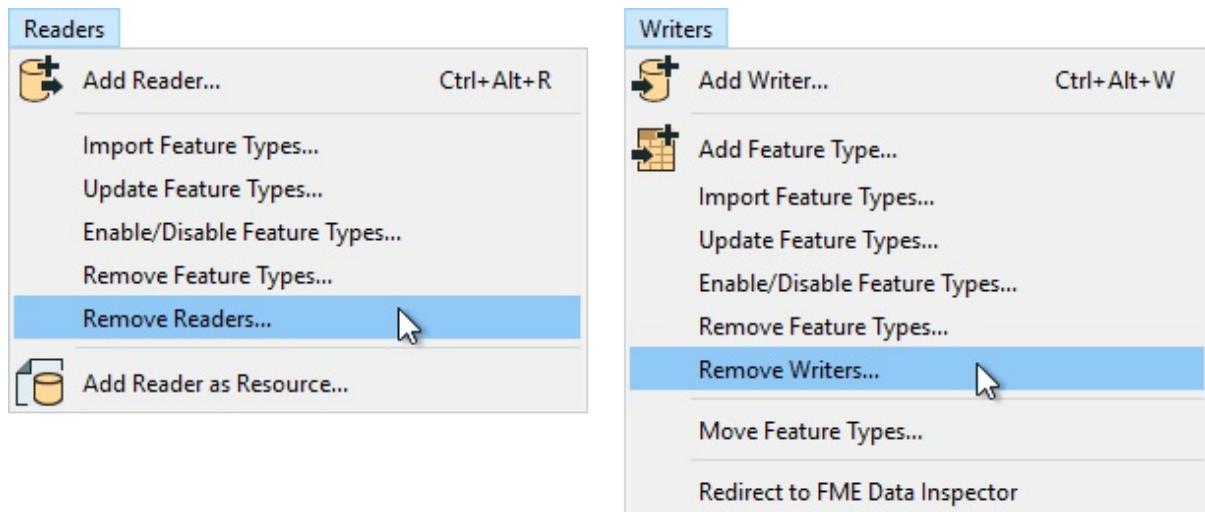
You can add as many readers and writers as you require in this way.

### TIP

*A reader can also be added by dragging a dataset from a filesystem explorer and dropping it onto the Workbench canvas.*

### Removing a Reader or Writer

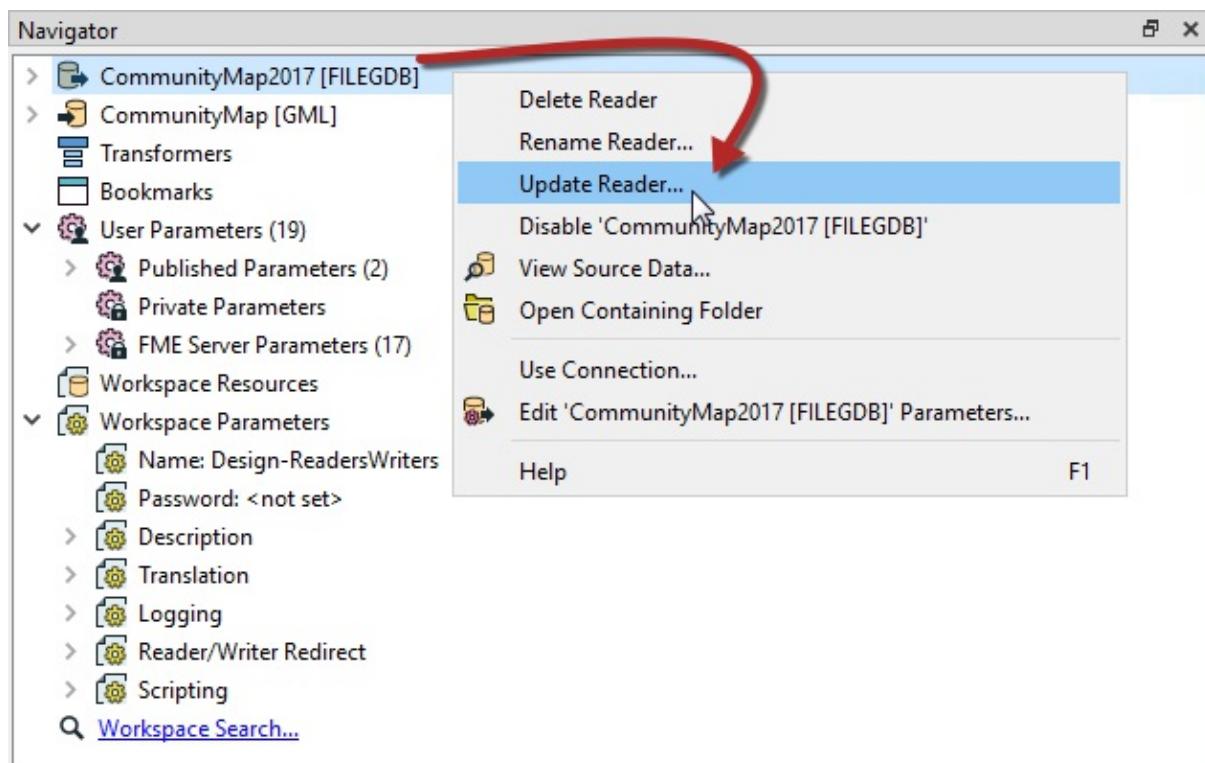
If a reader or writer is no longer required, then it can be removed very simply using options on the menu bar:



Alternatively, it's possible to right-click a reader/writer in the Navigator window and choose the Delete option.

## Updating a Reader or Writer

Readers and writers can be updated so that older workspaces have the speed and functionality available in a newer version of FME. You can update a reader/writer by right-clicking the reader/writer in the Navigator window and choosing the Update option:

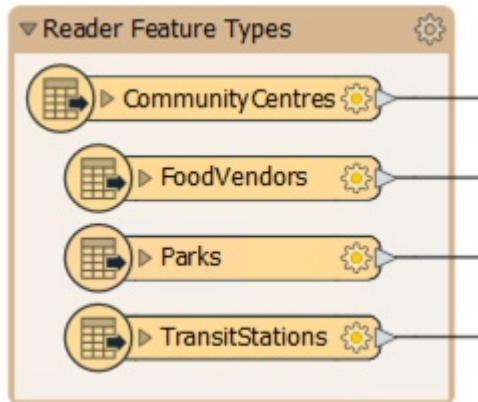


For readers, this tool provides the option to either update the reader or to also update the list of feature types being read. This way the workspace can be updated if the source data changes. Another way to update feature types is Reader > Update Feature Types on the menu bar.



## Reader Parameters

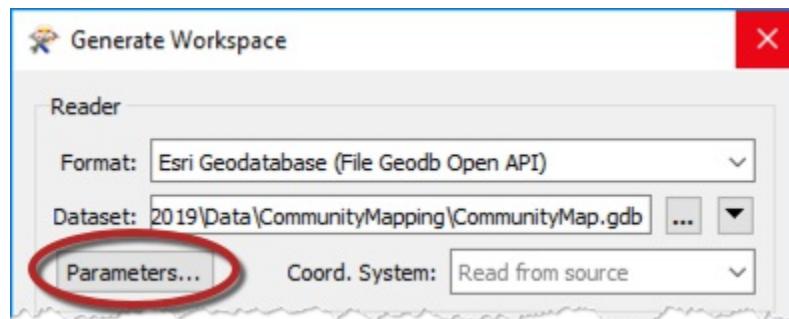
As we know, a workspace contains a reader to read a dataset, and each feature type in that dataset is shown in the workspace canvas:



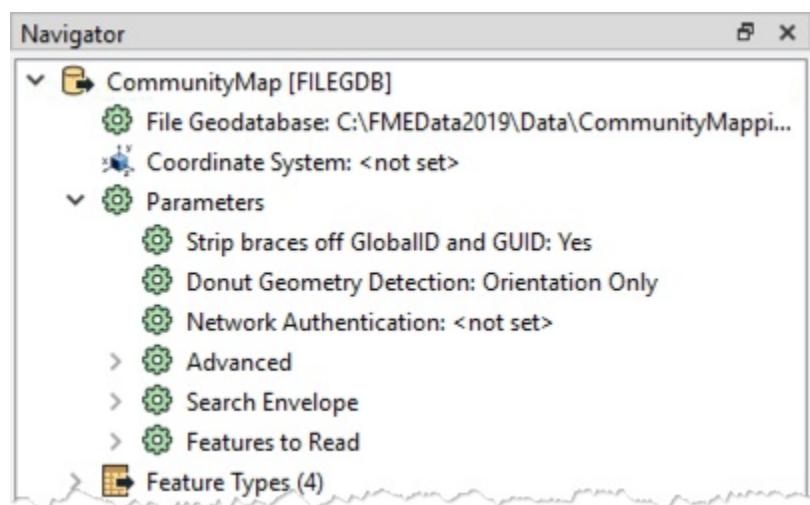
To control how that reader operates requires the use of **reader parameters**.

### Finding Reader Parameters

Reader parameters can be located - and set - by clicking Parameters when a new workspace is being generated:



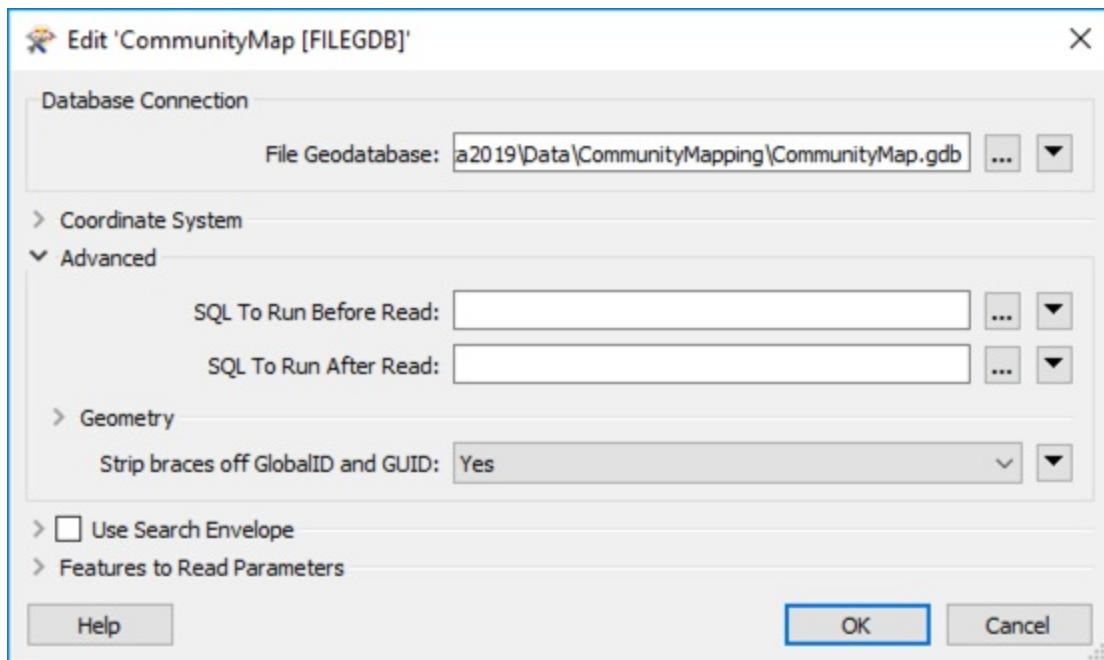
They can also be found in the Navigator window in Workbench:



Because parameters refer to specific components and characteristics of the related format, readers of different formats have a different set of control parameters.

### Setting Reader Parameters

To edit a parameter in the Navigator window, double-click on any of the parameters. Doing so opens up a dialog where the parameter's value may be set:

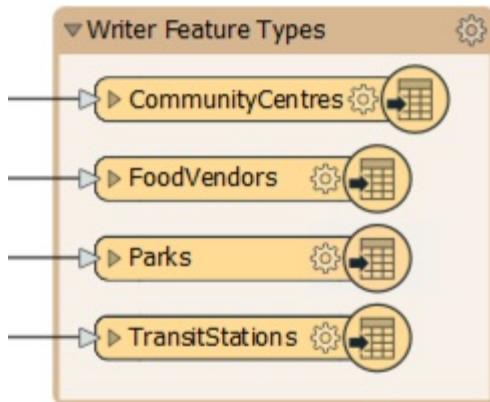


### FME Lizard says...

*Reader parameters control all feature types in the dataset. Think of it like brewing a pot of coffee. The strength control on the coffee machine affects all the cups that are poured.*

## Writer Parameters

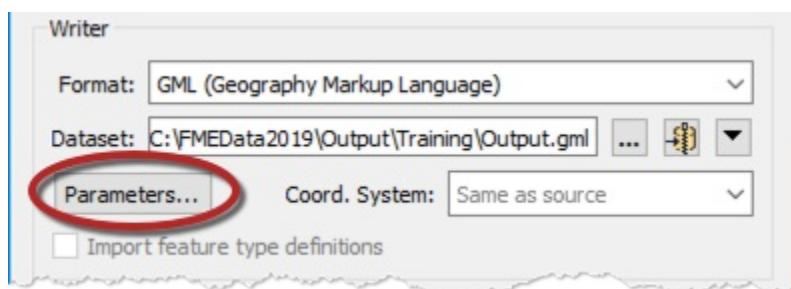
Like readers, we know a workspace contains a writer to write a dataset, and each feature type to be written is shown in the workspace canvas:



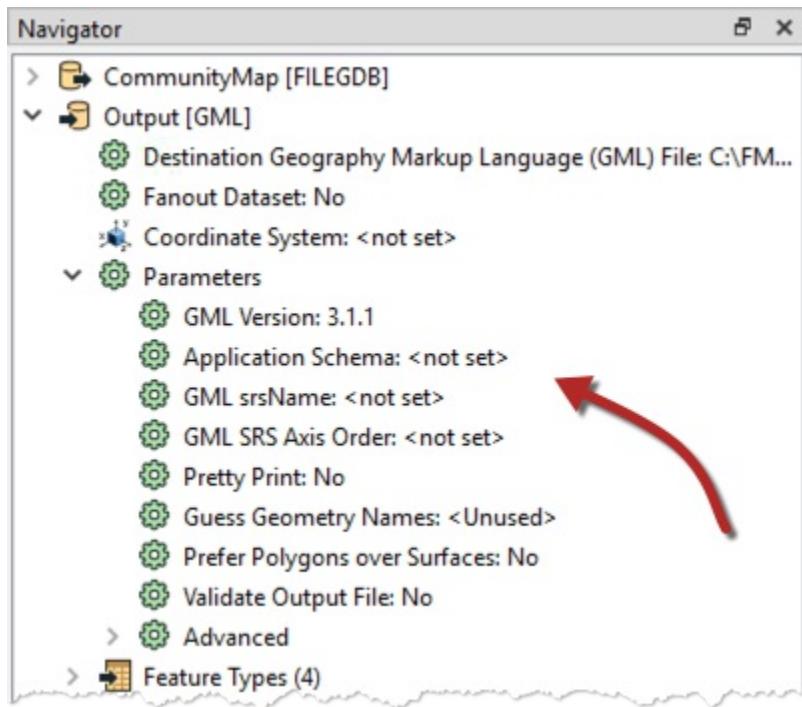
To control how that writer operates requires the use of **writer parameters**.

### Finding Writer Parameters

Writer parameters can be located - and set - by clicking Parameters when a new workspace is being generated:



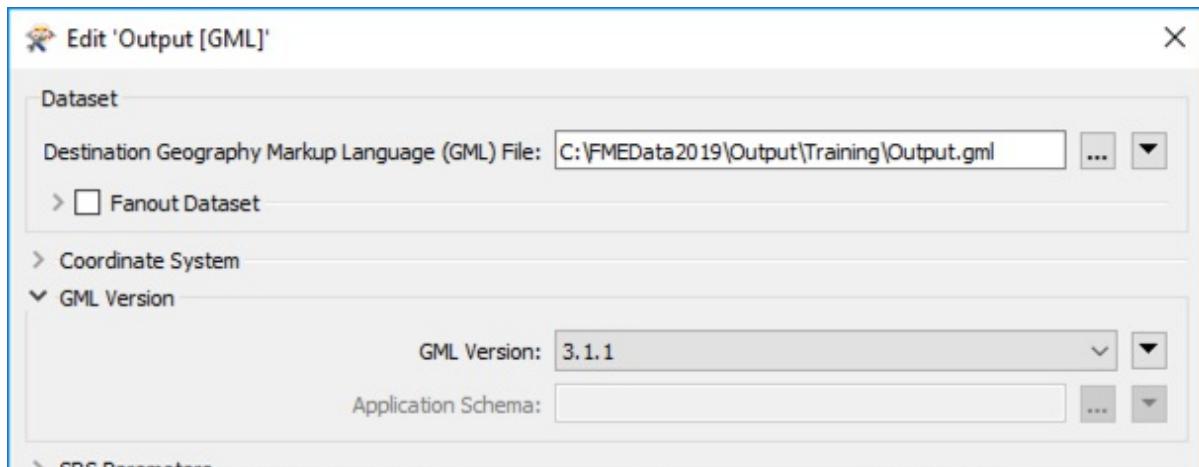
They too can also be found in the Navigator window in Workbench:



Because parameters refer to specific components and characteristics of the related format, writers of different formats have a different set of control parameters.

## Setting Writer Parameters

To edit a parameter in the Navigator window, double-click on any of the parameters. Doing so opens up a dialog where the parameter's value may be set:



### FME Lizard says...

*Like readers, writer parameters control all feature types in the dataset. For example, in the above screenshot, all feature types will be version 3.1.1.*

*But each reader and writer feature type does have its own settings (in the same way that each cup of coffee can be adjusted with cream and sugar). We'll find out about that in the next chapter.*



## Exercise 2

## Residential Garbage Collection Zones

<b>Data</b>	Addresses (Esri Geodatabase), Zones (MapInfo TAB)
<b>Overall Goal</b>	Create boundaries for residential garbage collection
<b>Demonstrates</b>	Readers and Writers
<b>Start Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\Design-Ex2-Begin.fmw
<b>End Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\Design-Ex2-Complete.fmw

Here we continue with a project to redefine garbage collection schedules.

In the first exercise, we used various transformers to divide addresses into five separate groups. Now the task is to refine that work by estimating the number of residents per address based on the zone type it falls within:

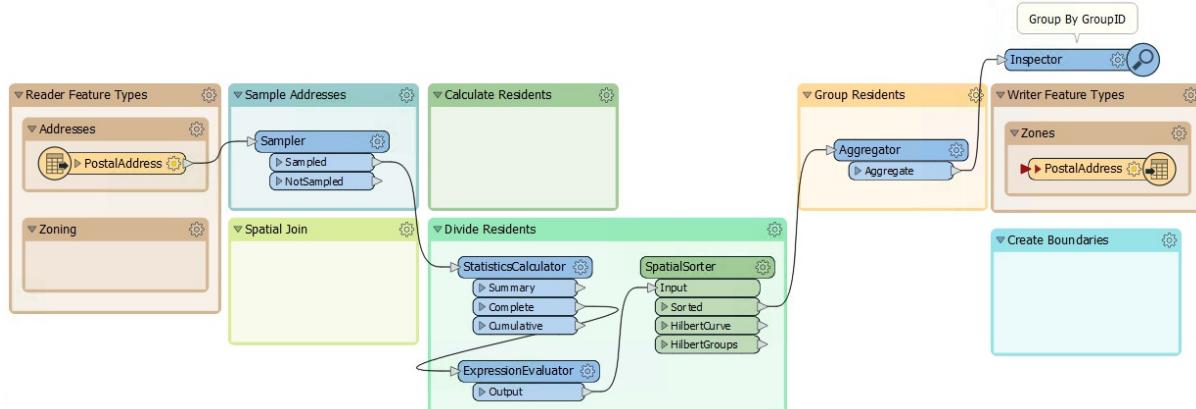
- Single-family residences: 2 adults
- Two-family residences: 4 adults
- Multi-family residences: 12 adults
- Comprehensive development zone: 8 adults
- Commercial properties: 1 adult

### 1) Open Workspace

Open your workspace from the previous exercise.

If you gave that workspace a version number in its name, then you should make a copy of the workspace with a new version number. For example, if you saved it to GarbageCollection-v1.fmw then make a copy named GarbageCollection-v2.fmw and open that for editing.

Alternatively you can open the workspace C:\FMEData2019\Workspaces\DesktopBasic\Design-Ex2-Begin.fmw:



The remaining transformers in the "Transformers" bookmark are these:

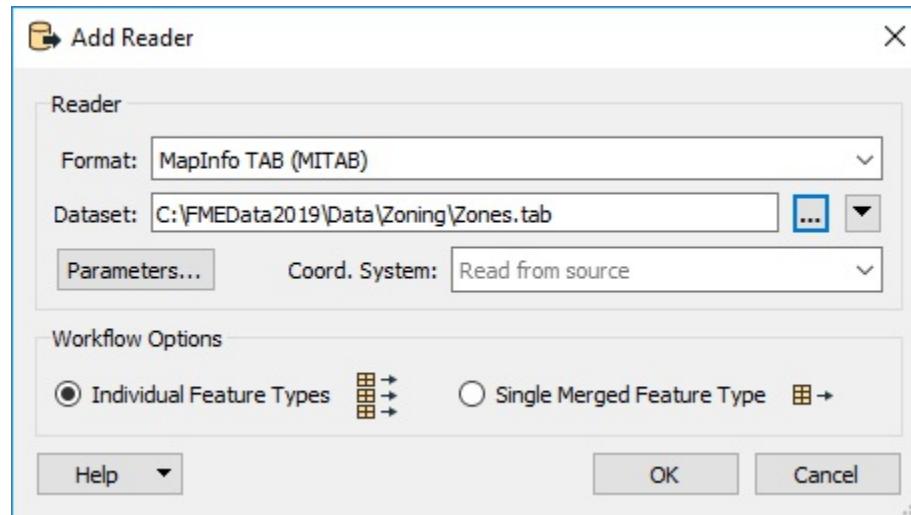


## 2) Add Reader

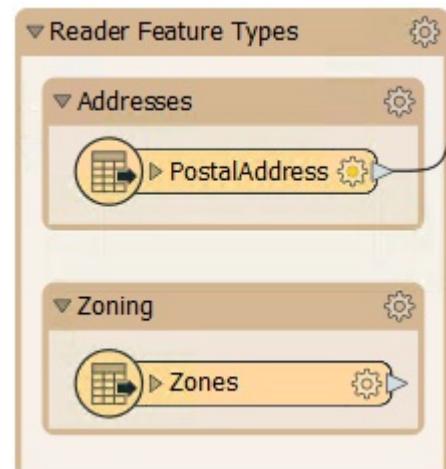
The first task here is to identify which planning zone each address falls inside. We need to read the zoning data and carry out a spatial join. To read a new dataset of data in a different format requires a new reader.

So, select Readers > Add Reader from the menu bar. When prompted enter the following parameters:

<b>Reader Format</b>	MapInfo TAB (MITAB)
<b>Reader Dataset</b>	C:\FMEData2019\Data\Zoning\Zones.tab



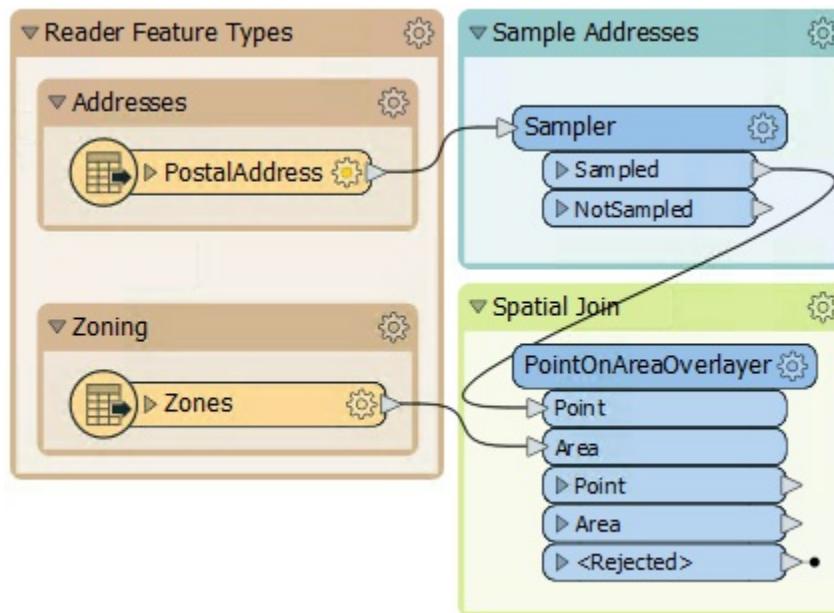
A reader is added to the Navigator window and a feature type to the canvas. Move the feature type into the Zoning bookmark:



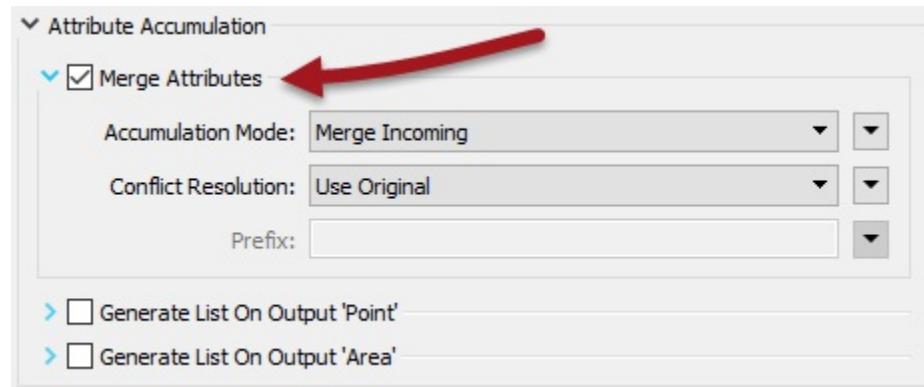
### 3) Create Spatial Join

To carry out a spatial join we'll use a PointOnAreaOverlayer transformer; this is a type of join called Point-in-Polygon.

So, move the PointOnAreaOverlayer transformer from the "Transformers" bookmark to the "Spatial Join" bookmark. Connect the newly added Zoning data to the Area port and the output from the Sampler to the Point port:



Inspect the PointOnAreaOverlayer parameters. Expand the Attribute Accumulation section and put a check mark against Merge Attributes:



This transformer is the first we've used that has a live <Rejected> port. For now, we'll leave it to stop the translation, since during testing we want to know about anything that causes a failure of the transformer.

Run the translation, ignore the Invalid Transformer Parameters dialog that pops up and click Run. This dialog pops up because we have previously run the translation to the Aggregator, but now we have broken that connection. We will fix it shortly.

Click on the PointOnAreaOverlayer:Point output port to view the data in the Visual Preview window. View both the Graphics and Table view. The overlay and attribute merging should cause each address to be given a zone name and category, click on any of the zones to confirm this.

### 4) Calculate Residents

The next step is to set how many residents live at a certain address according to its zoning type.

We know that:

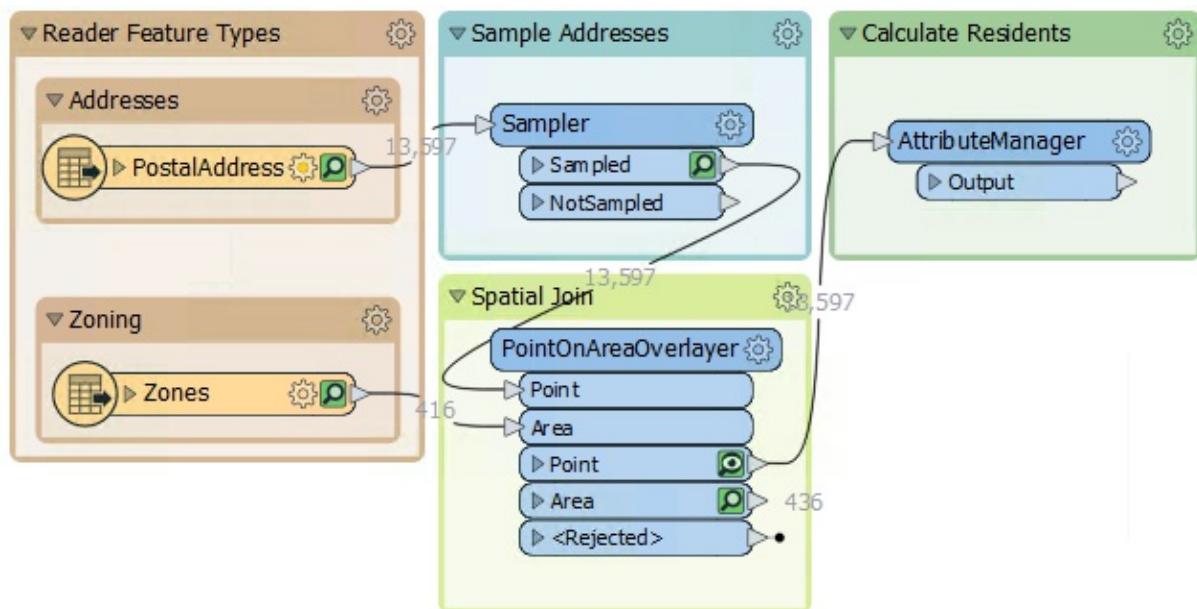
Zone Begins With	Zone Type	Residents

RS	Single Family	2
RT	Two Family	4
RM	Multiple Family	12
CD	Comprehensive	8
C	Commercial	1

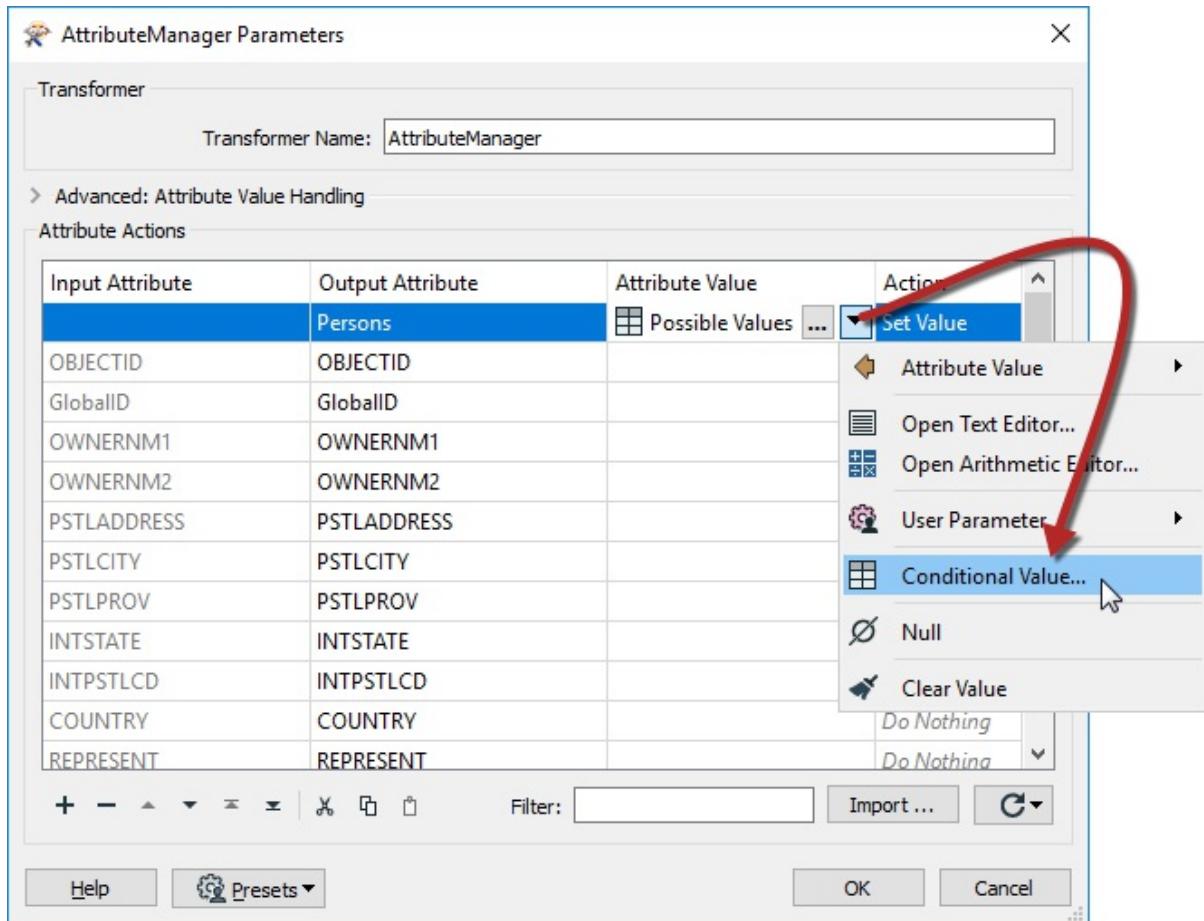
For example, zones RS-1, RS-2, RS-3 are all single-family zones, and we assume a total of two adults per address. This assumption makes it slightly more complicated because we need to match a zone type using a "begins with" string comparison.

This match can be done using an AttributeManager with **Conditional Values**.

This step is slightly complex, but luckily the AttributeManager inside the "Transformers" bookmark is already set up for this purpose. So move the AttributeManager into the "Calculate Residents" bookmark and connect it to the PointOnAreaOverlayer:Point output port:



If you are interested in what Conditional Values look like, open the parameters dialog for the AttributeManager and click the drop-down arrow in the Attribute Value field for the Persons attribute. Choose Conditional Value:



Doing so opens a Tester-like dialog with multiple conditions that test for each zone type, and an attribute value to set them to:

The 'Condition Statement' dialog box displays a table of conditions. The 'Test Condition' column contains various conditional statements like 'If @Value(ZoneName) BEGINS\_WITH RS', and the 'Attribute Value' column contains corresponding values like '2'. An 'Edit...' button is located at the bottom right.

Test Condition	Attribute Value
If @Value(ZoneName) BEGINS_WITH RS	<input type="checkbox"/> 2
Else If @Value(ZoneName) BEGINS_WITH RT	<input type="checkbox"/> 4
Else If @Value(ZoneName) BEGINS_WITH RM	<input type="checkbox"/> 12
Else If @Value(ZoneName) BEGINS_WITH CD	<input type="checkbox"/> 8
Else If @Value(ZoneName) BEGINS_WITH C	<input type="checkbox"/> 1
Else If	
Else <All Other Conditions>	<input type="checkbox"/> <No Action>

## TIP

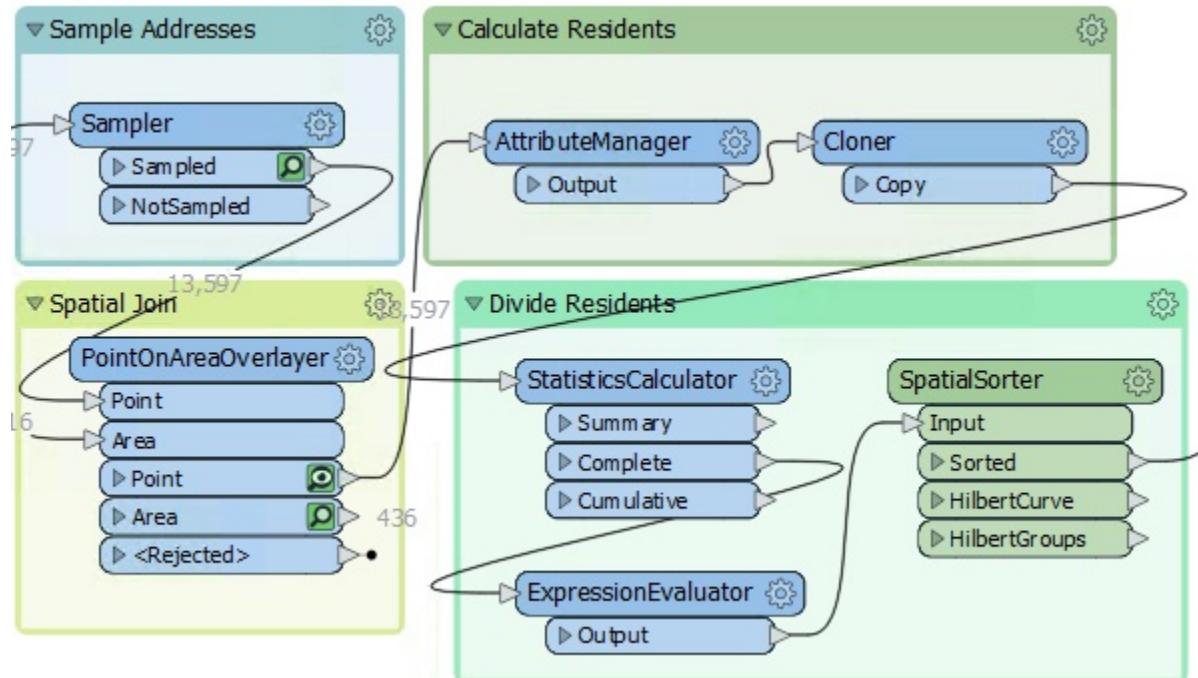
*If you want to challenge yourself, try adding a blank AttributeManager instead of moving the existing one. Then create the conditional statements to match the image above.*

## 5) Create Residents

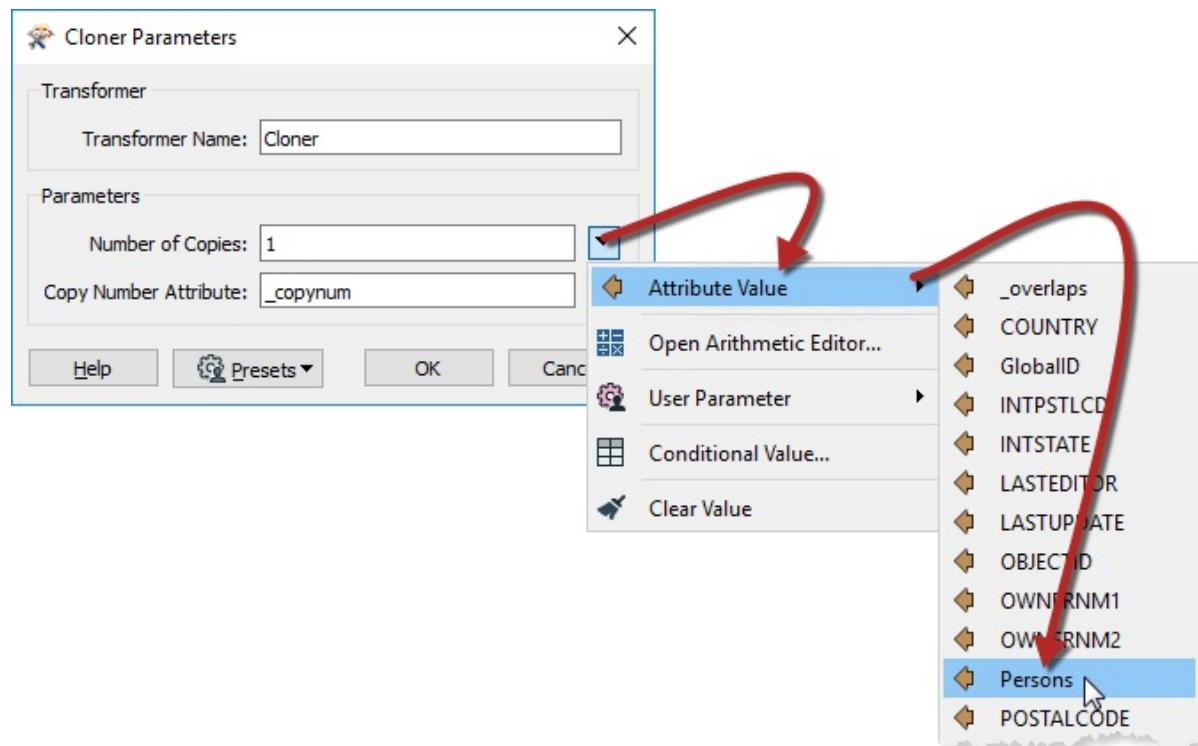
We now know (or have approximated) the number of residents per address. However, we must use that number in a way that will affect the output. The simplest way to do this is to create multiple copies of each address, one for each resident.

For example, for an address with eight residents, we'll create eight address points.

We can do this very simply with a Cloner transformer. So, move the Cloner transformer from the "Transformers" bookmark to the "Calculate Residents" bookmark. Connect the AttributeManager to the Cloner's input and its output to the StatisticsCalculator:



Inspect the Cloner parameters. For the Number of Copies parameter, click the drop-down arrow and choose Attribute Value > Persons:



Doing so will create `<Persons>` copies of the original addresses (note that the transformer doesn't output the original as well, so the output is `<Persons>` features, not `<Persons>+1`).

## 6) Run Translation

Make sure an Inspector is still attached to the Aggregator transformer and run the translation. The translation will fail with the error message:

**Cloner\_<Rejected>: Termination Message: 'Cloner output a <Rejected> feature.'**

The translation failed because addresses without a resident (for example, Industrial) have no Persons attribute and are being rejected by the Cloner transformer. The `<Rejected>` port is still set up to stop the translation, and so we get this error.

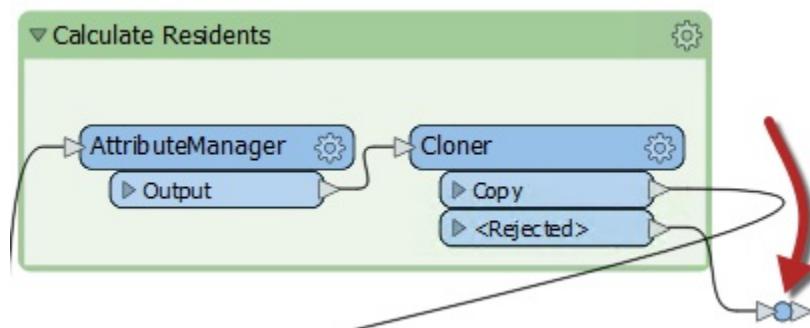
There are various choices to handle this. We could:

1. Change the Workspace parameter **Rejected Feature Handling** to *Continue Translation*
2. Add a transformer to handle the Cloner's rejected features
3. Set the Conditional Values to give a value of zero, instead of not including a value at all

Setting the Conditional Values would be the best solution to deal with the problem directly. But there might be other causes for rejected data, and we want to deal with that without having the translation stopped.

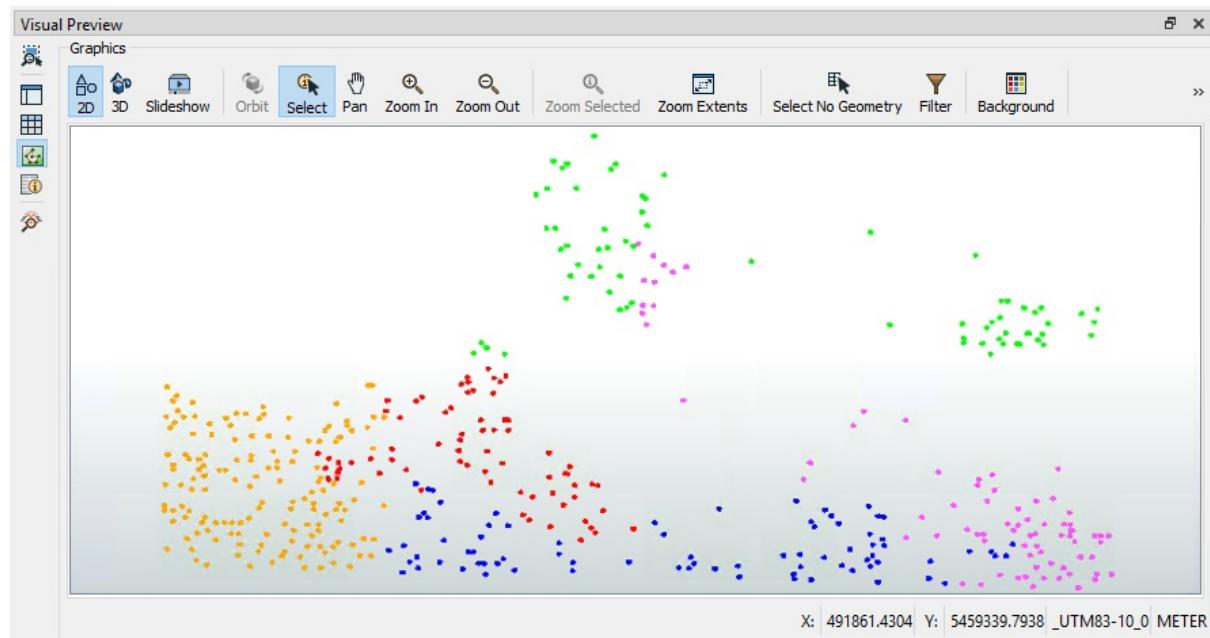
Setting the Rejected Feature Handling parameter means all `<Rejected>` ports would ignore rejected output. This setting might be useful in a production workspace, but in testing, we would probably want to stop the translation so that we can be aware of issues immediately.

So for us, the better solution is to add a transformer to the Cloner `<Rejected>` port. We don't need to inspect or log these features because we know that they will exist. So connect the `<Rejected>` port to a small transformer called a Junction:



This Junction will handle the rejected output, but quietly drop it without further fuss.

Re-run the translation. The output should be five groups of point feature again, but in a different pattern to the end of the previous exercise:



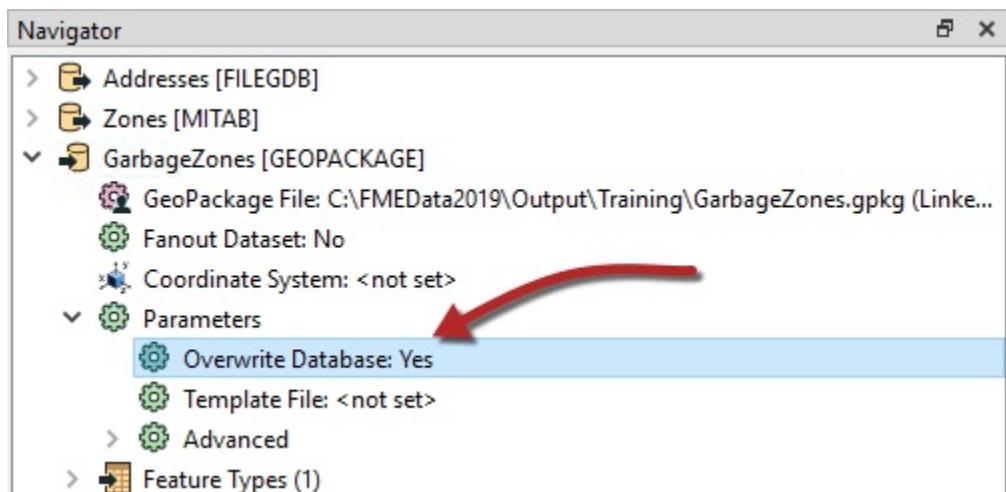
## 7) Write Output

Now to write some output. The simplest method is to connect the Aggregator output to the PostalAddress output feature type and re-run the workspace.

However, it would also be useful to rename the output feature type and remove all of its attributes, since they are from the reader dataset and don't apply here. So open the writer feature type parameters dialog. In the Parameters tab rename the feature type to GarbageZones. In the User Attributes tab, remove all of the attributes that are being written:



Also, we should change the GeoPackage writer parameter Overwrite Database to overwrite the database each time we run the workspace. To do this, find the GarbageZones [GEOPACKAGE] reader in the Navigator, expand the Parameters, double-click Overwrite Database, and then check the box and click Ok:



That way we don't accumulate more and more results in the same dataset.

---

## CONGRATULATIONS

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

- *Add a reader to a workspace*
- *Carry out a point-in-polygon spatial join*
- *Set conditional values in an AttributeManager transformer*
- *Use a Cloner transformer to create multiple copies of data*
- *Manage rejected features*

## **Workspace Testing Techniques**

When developing a workspace incrementally, or making edits to a structured workspace, a lot of testing will take place. Each time a change is made the author will wish to test the change, and this process of change/test repeats itself again and again.

To effectively and efficiently carry out this process, there are a number of tools available in FME Workbench.

The first set of tools are for inspecting data as it changes. The second set of tools are for defining which parts of the workspace need to be tested.

## Partial Runs

A partial run is when only one section of a workspace is carried out. One way to do this is to disable objects in the canvas to only run certain enabled sections. Another method is to use a tool called Partial Runs, which is represented by pop-up options when a workspace is run with caching turned on.

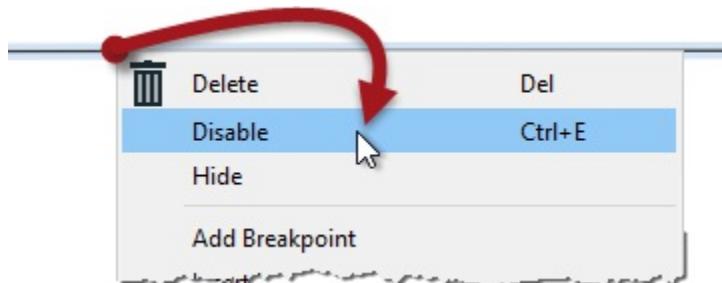
The technique you use will depend on how large the workspace is, and how much of it you need to run. You may use one technique or the other - or you may use both!

---

### Disabled Objects

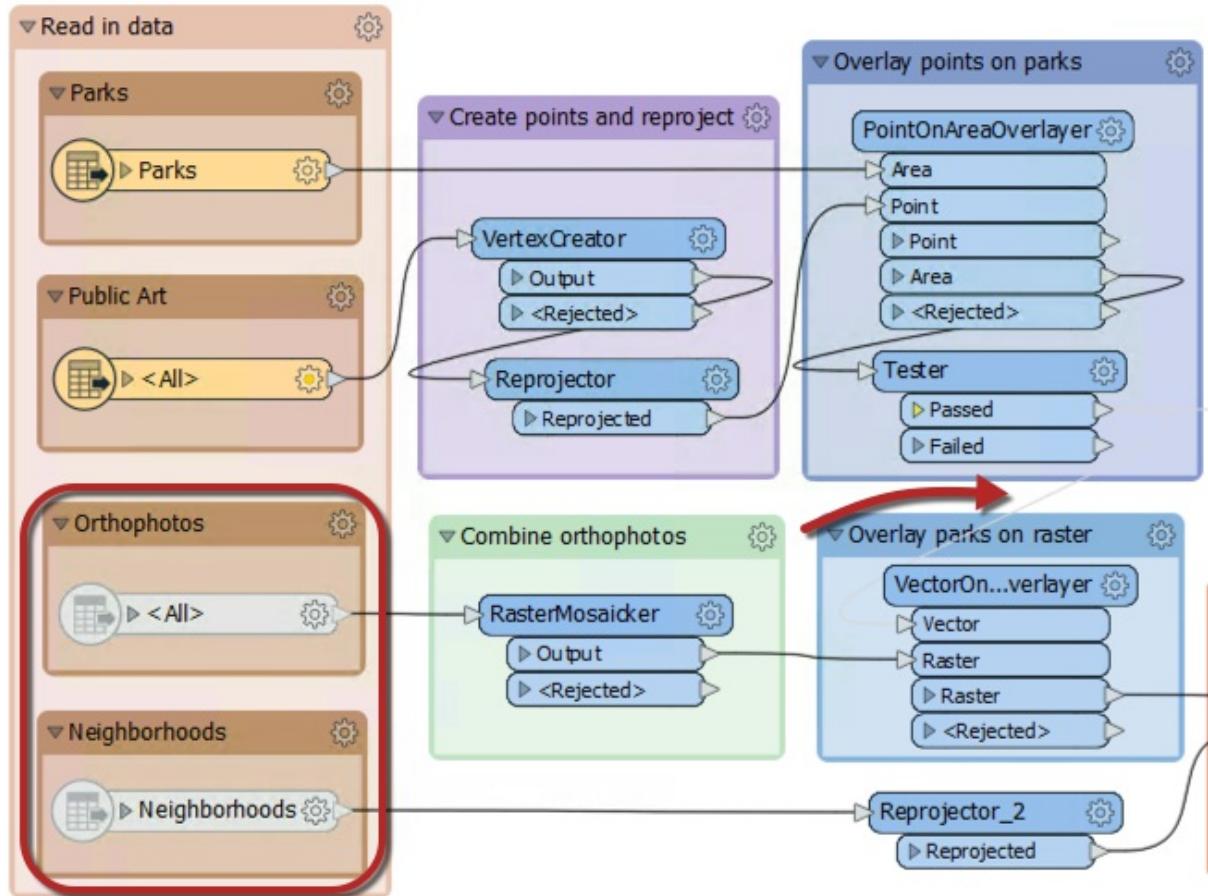
If designed correctly, a large workspace should be made up of small sections. Isolating a section (or part of a section) for testing is possible by disabling connections to all other components.

An object (connection or feature type) is disabled by right-clicking it and choosing the option to Disable (or selecting it and using the shortcut **Ctrl+E**):



A disabled connection is rendered inoperative in much the same way as if it had been deleted, and no features will pass through. The same disabling can be done to other canvas objects such as transformers and feature types. Even a reader/writer can be disabled through the Navigator window.

Here an author has disabled two connections (both from the Tester:Passed port) and two feature types:



With that setup the top part of the workspace will operate up until (and including) the Tester. The bottom portion will not run at all. No data will emerge from the disabled feature types, and no data is passed to it from the Tester.

With caching turned on, the author can inspect part of the workspace without having to run the entire translation. This feature is a significant advantage when (like here) the disabled section takes up most of the overall processing time.

## Partial Runs

The technical aspect of partial runs has already been covered. However, using partial runs is an important part of workspace testing. You should use partial runs to develop your workspace incrementally, testing each new section to ensure it works.

A partial run is particularly useful in avoiding re-reading data from its source; especially when the data comes from a slow, remote location such as a web service. Creating a cache can help increase the development of your workspace.

Finally, caches can be saved with the workspace when it is saved as a template (File > Save As Template, check Include Feature Caches). That means the workspace can be re-run using the caches from a previous session or even from another author!

### Exercise 3

### Residential Garbage Collection Zones

<b>Data</b>	Addresses (Esri Geodatabase), Zones (MapInfo TAB)
<b>Overall Goal</b>	Create boundaries for residential garbage collection
<b>Demonstrates</b>	Readers and Writers
<b>Start Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\Design-Ex3-Begin.fmw
<b>End Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\Design-Ex3-Complete.fmw

Here we continue with a project to redefine garbage collection schedules.

In the first two exercises, we used various transformers to divide addresses into five separate groups, according to zoning type. Then we wrote the data to Geopackage.

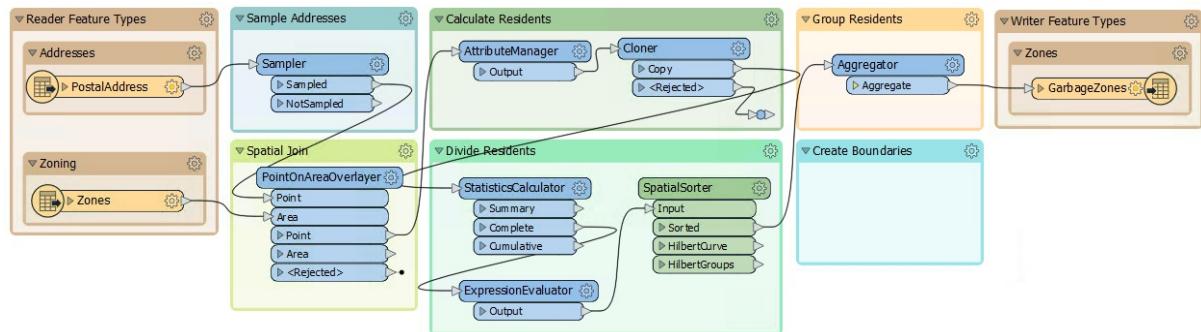
Now the task is to replace the groups of point features with a polygon boundary.

#### 1) Open Workspace

Open your workspace from the previous exercise.

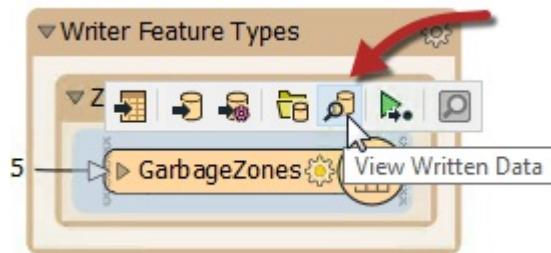
If you gave that workspace a version number in its name, then you should make a copy of the workspace with a new version number. For example, if you saved it to GarbageCollection-v2.fmw then make a copy named GarbageCollection-v3.fmw and open that for editing.

Alternatively you can open the workspace C:\FMEData2019\Workspaces\DesktopBasic\Design-Ex3-Begin.fmw:



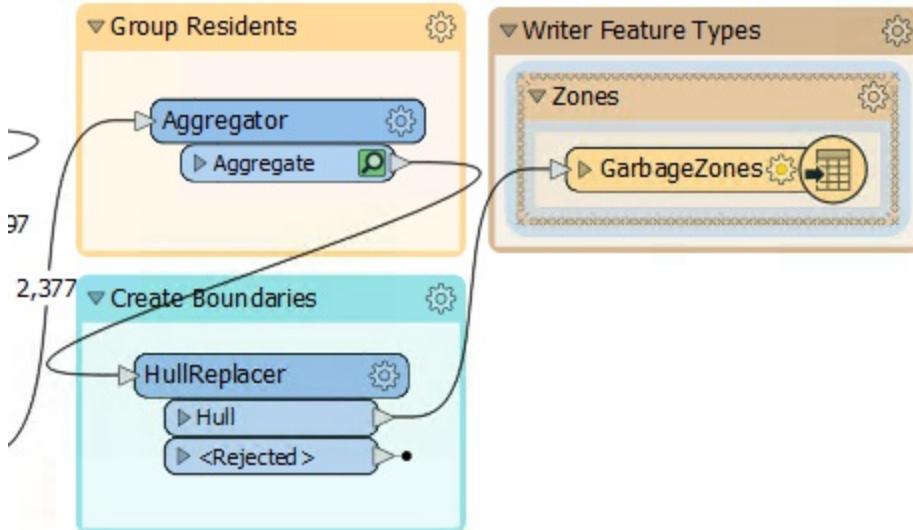
#### 2) Run the Workspace

Run the workspace to finish writing out the data. You can inspect the output dataset if you desire by clicking on the GarbageZones writer feature type and then clicking on the View Written Data button in the popup menu:



#### 3) Add a HullReplacer Transformer

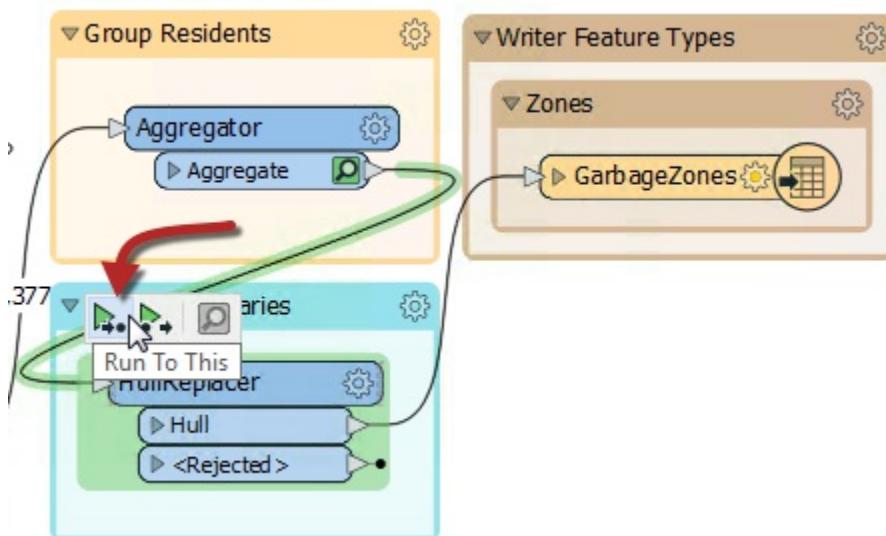
Move the HullReplacer transformer from the "Transformers" bookmark into the "Create Boundaries" bookmark. Connect it between the Aggregator and writer feature type:



Notice how the HullReplacer has no cache because it is newly placed.

#### 4) Re-Run the Workspace

Now let's re-run the workspace. But rather than re-write the output data, we can run just the new transformer we just added. Click on the HullReplacer transformer and on the icons that pop up, click Run To This:

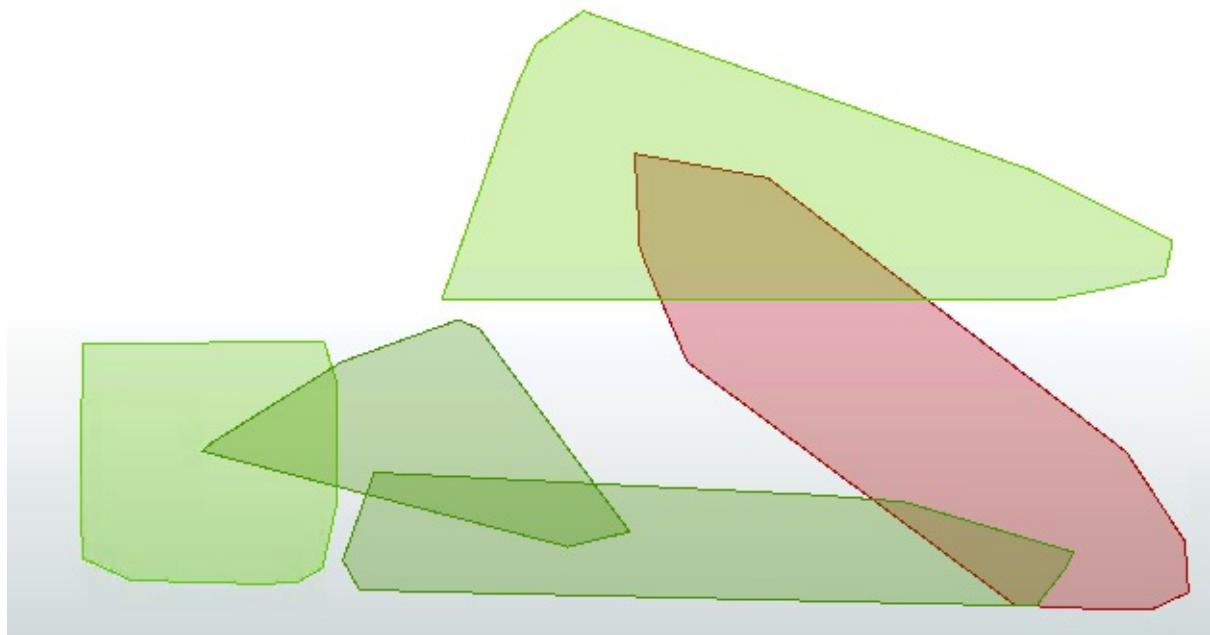


Notice how hovering shows what parts of the workspace will be run. Since we already have features cached up to the Aggregator (assuming you haven't closed the workspace since it was last run), only the section between the Aggregator and the HullReplacer will be run.

#### TIP

*Using the Run To This option is a good method to check your translation before writing the data out, especially if you are writing to a database or an online source.*

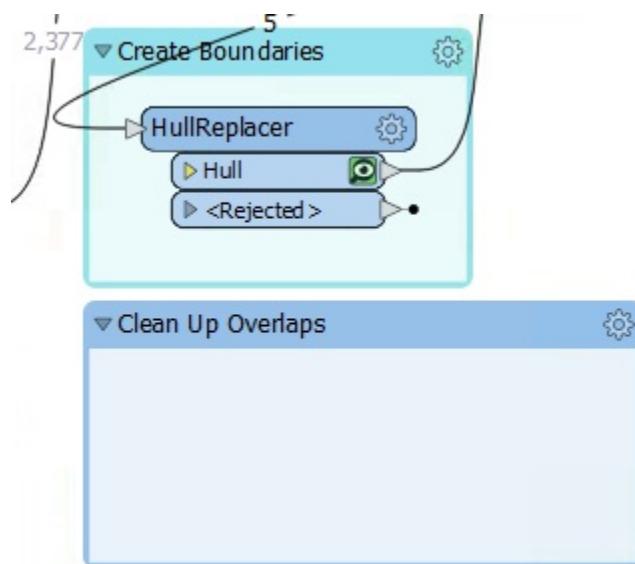
Click on the cached features on the HullReplacer:Hull output port to confirm the data. The output now includes polygons, to prove that the translation has functioned correctly:



## 5) Clean Up Overlaps

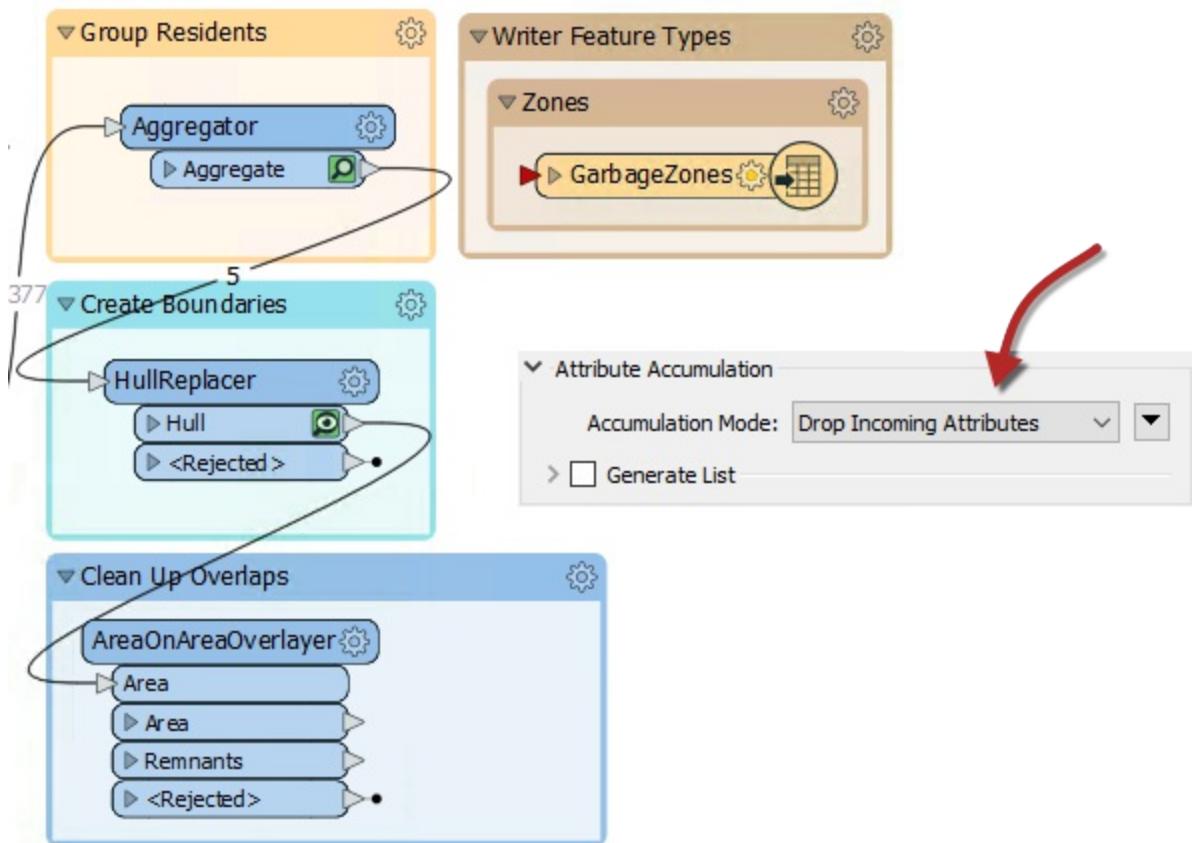
The problem with the output is that all of the polygons overlap to some extent. That needs to be fixed so that there are no overlaps. What's more, we should check which zone an overlap belongs to by seeing which group contains most of its addresses.

Because this is unexpected, we don't have an area of the workspace set aside yet. Add a new bookmark (or move the now-empty Transformers bookmark) and name it Clean Up Overlaps:



## 6) Add an AreaOnAreaOverlayer Transformer

Overlaps can be dissected using the AreaOnAreaOverlayer transformer, so add one of these to the new bookmark, connected to the HullReplacer transformer. Check the parameters and set the **Attribute Accumulation Mode** to *Drop Incoming Attributes*.



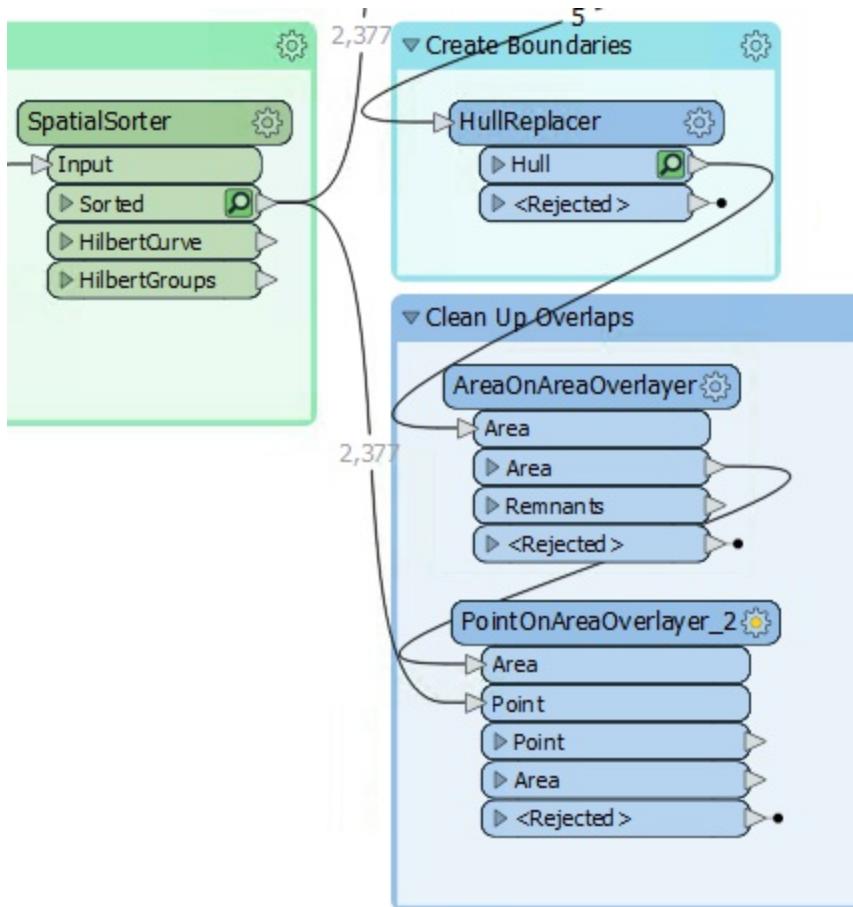
## 7) Add a PointOnAreaOverlayer Transformer

The overlaps are now separate features, but we do not yet know to which area they should be assigned. It should be the one with most addresses; for example, if an overlap contains 31 addresses from group one, and 52 addresses from group two, then it should be assigned to the group two polygon.

We can start on this by using a PointOnAreaOverlayer. This transformer will let us create a list of which addresses an overlap contains.

So add a PointOnAreaOverlayer transformer. The area features will be the output from the AreaOnAreaOverlayer.

The point features should be a copy of the addresses. The simplest way to get these is to make a second connection from the SpatialSorter:



Inspect the parameters. Under Attribute Accumulation, set the following parameters:

Merge Attributes	Yes
Generate List on Output 'Area'	Yes
'Area' List Name	PointList
Selected Attributes	GroupID

Attribute Accumulation

Merge Attributes

Accumulation Mode: Merge Incoming

Conflict Resolution: Use Original

Prefix: [ ]

Generate List On Output 'Point'

Generate List On Output 'Area'

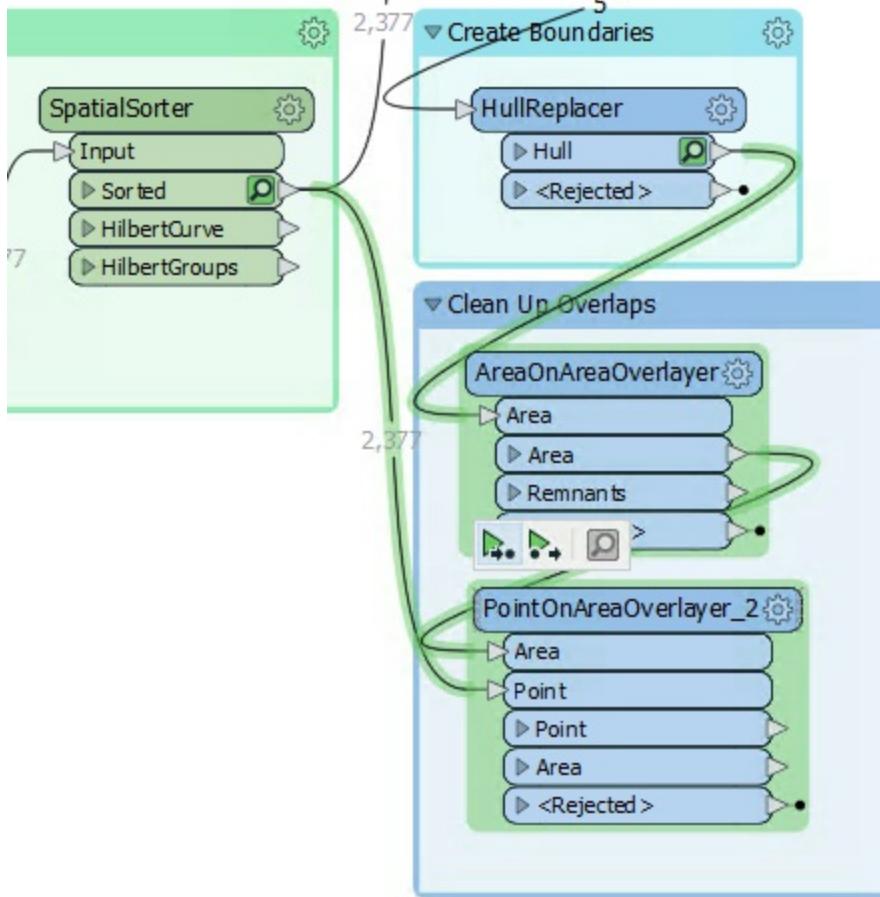
'Area' List Name: PointList

Add To 'Area' List: Selected Attributes

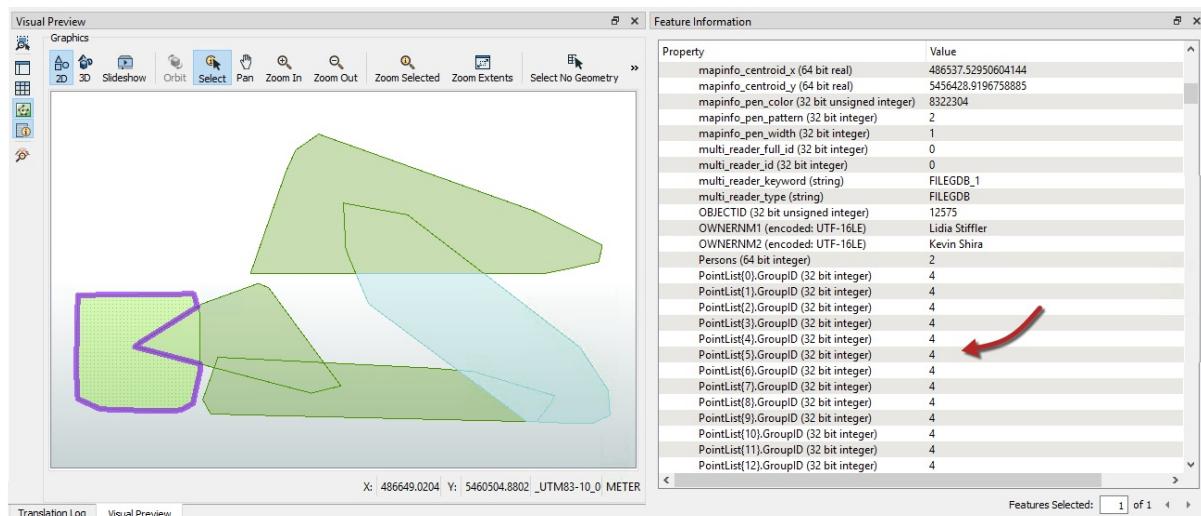
Selected Attributes: GroupID

Doing so will create an FME list attribute. A list attribute is an attribute with multiple values. Here the list will contain a record of the point features (and their GroupID values) that fall inside a polygon.

Confirm this works correctly by running the workspace at the new PointOnAreaOverlayer. Notice how the translation pulls data from two caches; the AreaOnAreaOverlayer and SpatialSorter transformers:



Inspect the PointOnAreaOverlayer:Area output port features. Selecting a feature will show (in the Feature Information window) the list attribute and all of its values:



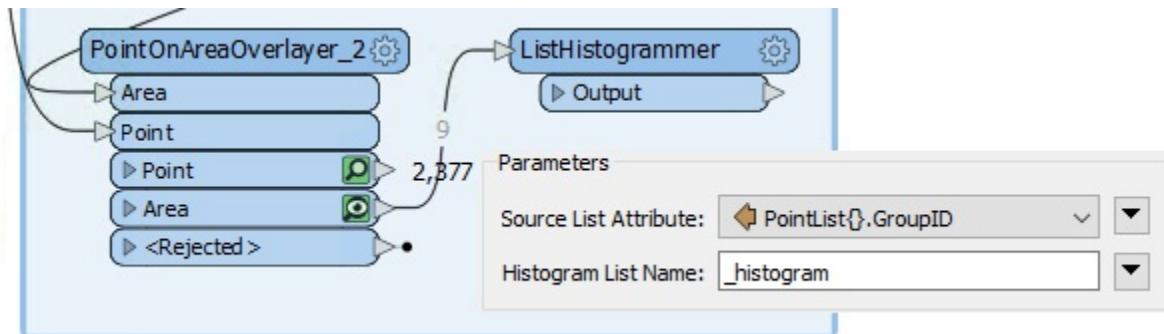
## 8) Add a ListHistogrammer Transformer

To count the most frequent GroupID for each list on an overlap's we'll use the ListHistogrammer transformer.

## TIP

This is not a commonly used transformer, so don't worry if you weren't aware of it, or if you are concerned about the large number of transformers available in FME. You will learn more about these transformers with practice. For now the ability to use partial runs is much more important.

Place a ListHistogrammer transformer connected to the PointOnAreaOverlayer:Area output port. Inspect the parameters and select PointList{}.GroupID as the source attribute:

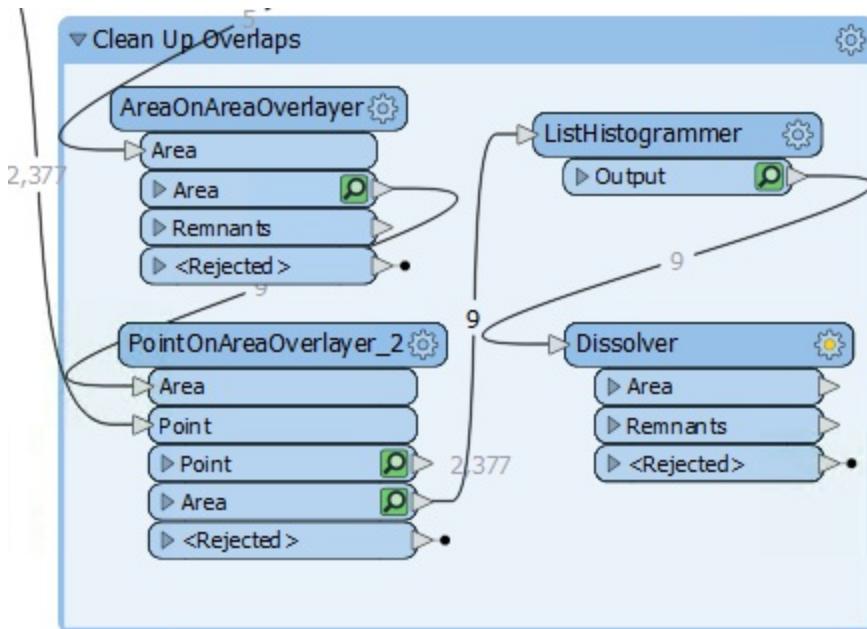


Re-run the workspace (from the ListHistogrammer) and inspect the results. Notice that a new list attribute is created; a list of the number of different GroupID values with the most frequent GroupID at the top of the list. So we merely need to use that GroupID to merge areas.

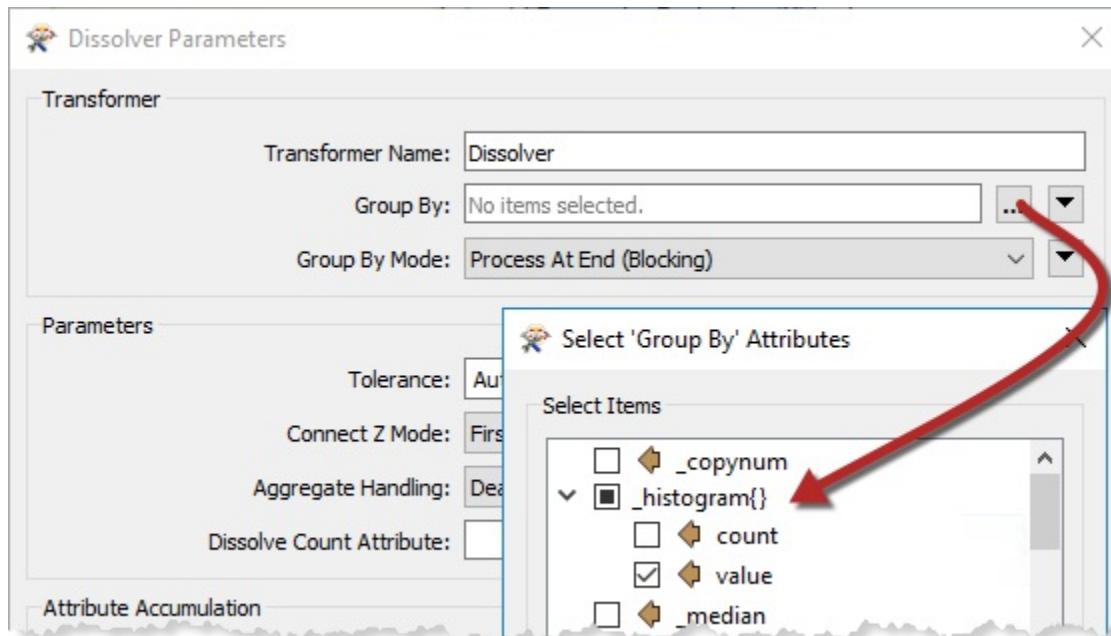
Feature Information	
Property	Value
Feature Type	ListHistogrammer_OUTPUT
Coordinate System	<a href="#">UTM83-10_0</a>
Dimension	2D
Number of Vertices	18
Min Extents	489056.53089999966, 5458141.3617
Max Extents	494185.6783999996, 5460171.461100001
<b>Attributes (564)</b>	
_copynum (32 bit unsigned integer)	2
_histogram{0}.count (encoded: UTF-8)	476
_histogram{0}.value (encoded: UTF-8)	1
_histogram{1}.count (encoded: UTF-8)	36
_histogram{1}.value (encoded: UTF-8)	2
_median (encoded: UTF-8)	Canada

### 9) Add a Dissolver Transformer

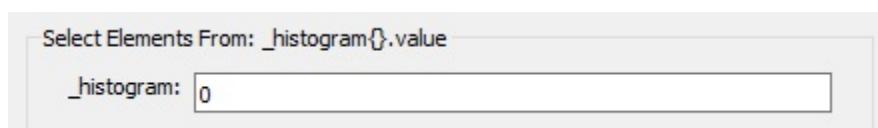
Finally add a Dissolver transformer to merge the features together. Connect the Dissolver to the ListHistogrammer output port:



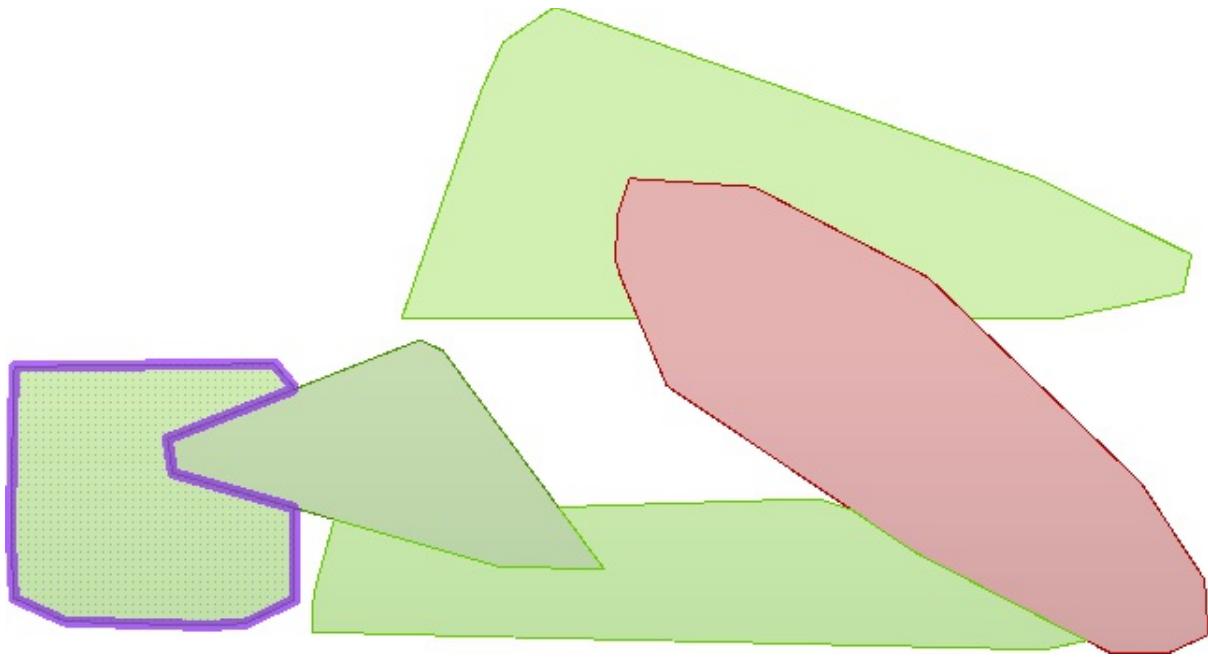
Inspect the parameters. Under Group By select the attribute `_histogram.value`:



You'll be prompted for a value; this is which item in the list do we want. We want the first element because it has the most values, so this field should be set to zero (which it will be by default):



Run the workspace to the Dissolver and inspect the Dissolver:Area output port:



We now have five polygon features to represent garbage collection areas, each with approximately the same number of residents. Connect the Dissolver:Area port to the writer feature type and this workspace is nearly complete.

#### 10) Remove the Sampler Transformer

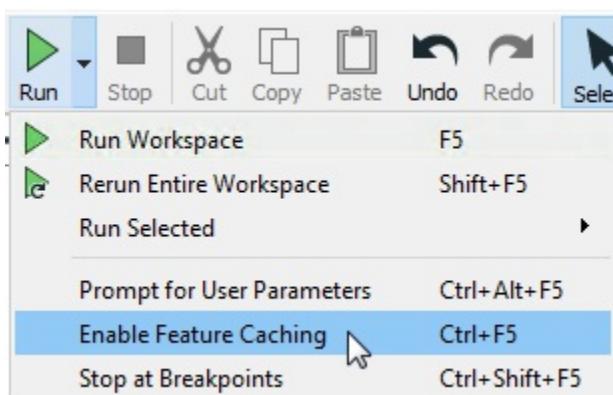
To complete the project let's run the workspace on the full dataset, but first let's get the workspace ready for production.

Delete the Sampler transformer, ensuring that PostalAddress and the PointOnAreaOverlayer:Point input port are connected.

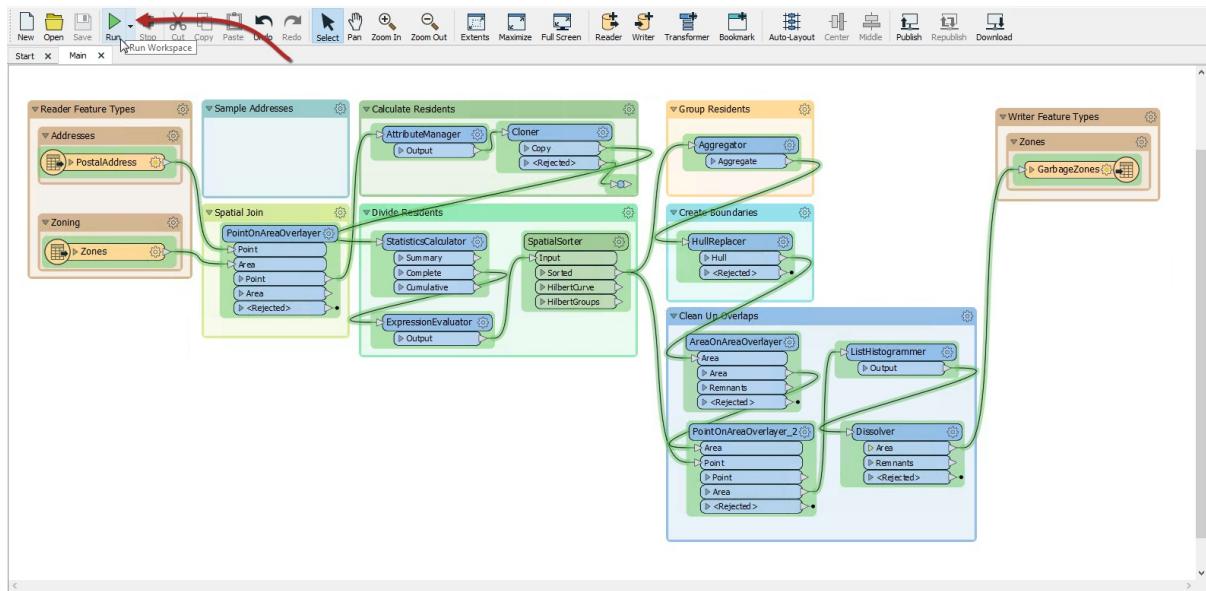
#### TIP

*Instead of deleting the Sampler, you can just disable it. Right-click on the Sampler and choose Disable, then connect the PostalAddress and PointOnAreaOverlayer just like the step above. This way if you need to come back and tweak something, the dataset can be sampled again easily.*

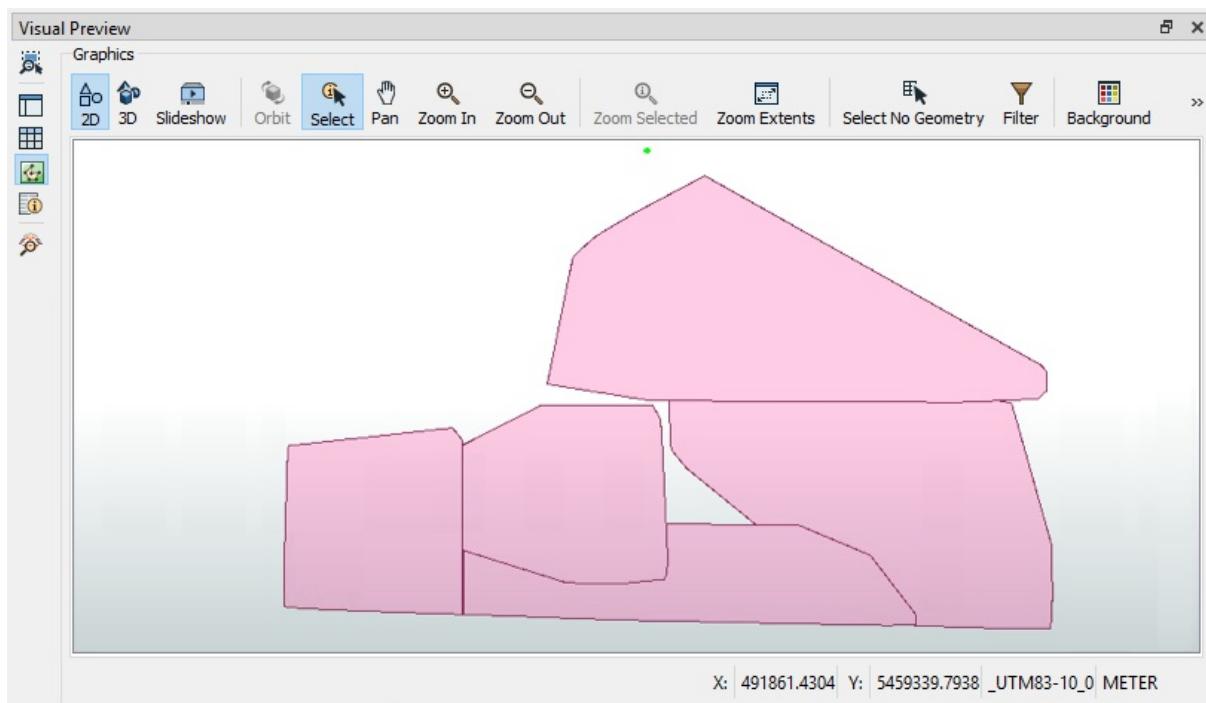
Before we run the translations, let's disable feature caching as the PostalAddress dataset is large. Click on the dropdown next to the Run button and click Enable Feature Cache to disable it:



Now with feature caching disabled, the Run button will run the entire workspace:



As expected, the result will look different, now that we're using the full dataset:



## CONGRATULATIONS

By completing this exercise you have learned how to:

- Use the Run To Here option for testing
- Disable Feature Caching
- Use new transformers: HullReplacer, ListHistogrammer, Dissolver
- Remove a transformer



## **Module Review**

This session was designed to help you understand how a translation is put together, the best ways of creating it, how to read data in different ways, and how to test different sections of a workspace.

### **What You Should Have Learned from this Module**

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

#### **Theory**

- A workspace is made up of readers, writers, and feature types. They exist in a hierarchical structure.
- Incremental development is a way of constructing a workspace, one section at a time.
- A workspace can consist of any number of readers and writers - or none at all
- Feature caching, partial runs, and disabling objects can be used to debug workspaces

#### **FME Skills**

- The ability to create a workspace one section at a time
- The ability to handle rejected data in a workspace
- The ability to add new readers and writers to a workspace

## **Questions**

There are no questions for this chapter.

---

## Practical Transformer Use

Transformers are at the very heart of FME's capabilities, like a workshop with a variety of machines capable of turning raw materials into a finished product. It's not for nothing that the main application is called FME Workbench!



To be able to use FME efficiently, being able to find the right transformer is an important skill, and it helps that many of the most-used transformers have a common theme.

For example, many transformers are related to being able to filter data within a workspace; once you understand the concept of filtering data, it becomes easier to find the transformer you need.

It's also important to understand the common aspects of transformers, to recognize that options and parameters in different transformers often operate in similar ways.

This chapter covers various searching methods for finding transformers, the common themes of the most popular transformers, and how parameter values can be constructed within a single transformer dialog.

## Locating Transformers

Even experienced FME users find the full list of transformers a daunting sight. In this section, you'll learn to stop worrying and love the transformer gallery.

The screenshot shows the FME Transformer Gallery interface. At the top, there's a search bar labeled "Search all transformers..." and filters for "Filter By All Categories" and "Sort By Most Used". Below this, a grid of transformer cards is displayed in three rows:

- Row 1:** Tester, AttributeCreator, AttributeManager, FeatureMerger
- Row 2:** Creator, Inspector, AttributeKeeper, TestFilter
- Row 3:** Clipper, Reprojector, AttributeRenamer, Aggregator

Each card contains a small icon, the transformer name, a brief description, and a gear icon for settings.

With an abundance of transformers, FME possesses a lot of functionality; probably a lot more than a new user realizes and much of which would be very useful to them. This section helps you find the transformer you need, even if you didn't know you needed it.

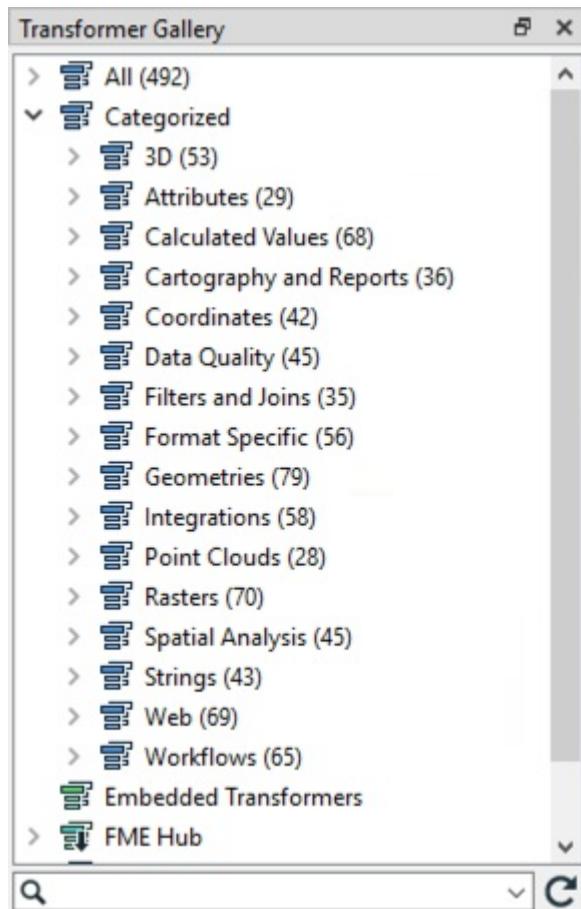
Although the transformer list can look a bit overwhelming, don't panic! The reality is that most users focus on 20-30 transformers that are relevant to their day-to-day workflow. You don't need to know every single transformer to use FME effectively.

## Transformer Gallery

The transformer gallery window is the obvious place to start looking for transformers. There are a number of ways in which transformers here can be located.

## Transformer Categories

Transformer categories are a good starting point from which to explore the transformer list. Transformers are grouped in categories to help find a transformer relevant to the problem at hand.



Although all of them are important, the most commonly used transformers are found in these categories:

- **Attributes:** Operations for attribute/list management
- **Calculated Values:** Operations that return a calculated value
- **Filters and Joins:** Operations for dividing and merging data flows
- **Geometries:** Operations that create geometry or transform it to a different geometry type
- **Spatial Analysis:** Operations that return the result of a spatial analysis
- **Strings:** Operations that manipulate string contents, including dates

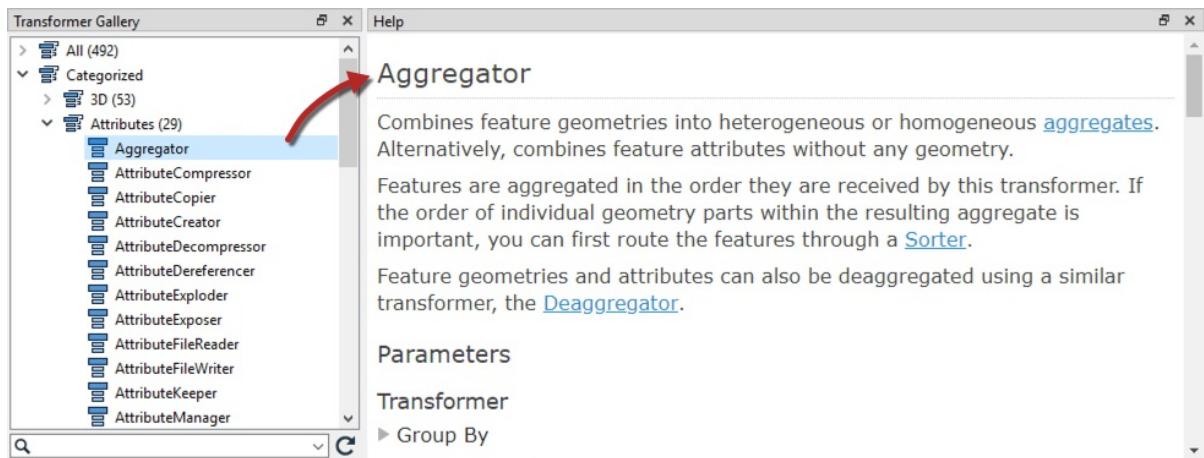
Simply click on the expand button to show all transformers within a particular category.

---

## Transformer Help

The FME Workbench Help tool displays information about transformers. Simply click on a transformer and press the F1 key to open the help dialog.

This tool is linked to FME Workbench so that a transformer selected (in the gallery or on the canvas) triggers content to display in the Help tool.



## TIP

*Another useful - and printable - piece of documentation is the [FME Transformer Reference Guide](#).*

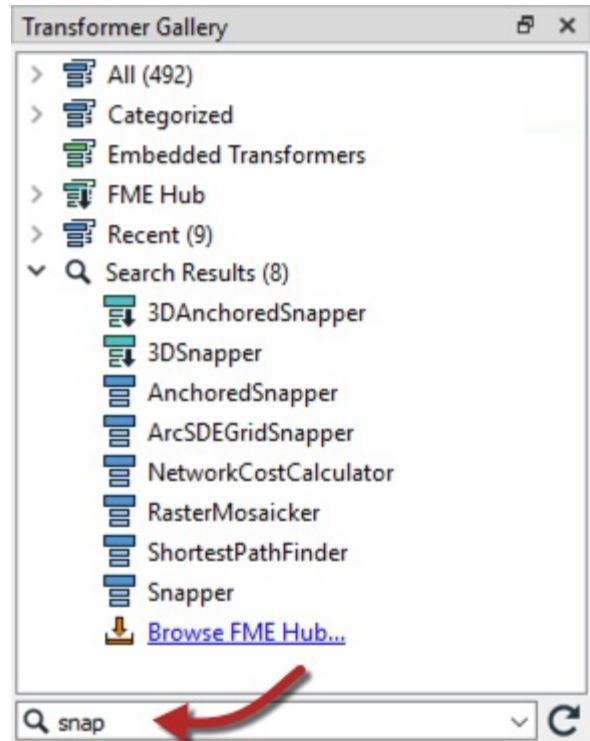
## Transformer Searching

There are search functions in both the transformer gallery and Quick Add dialog.

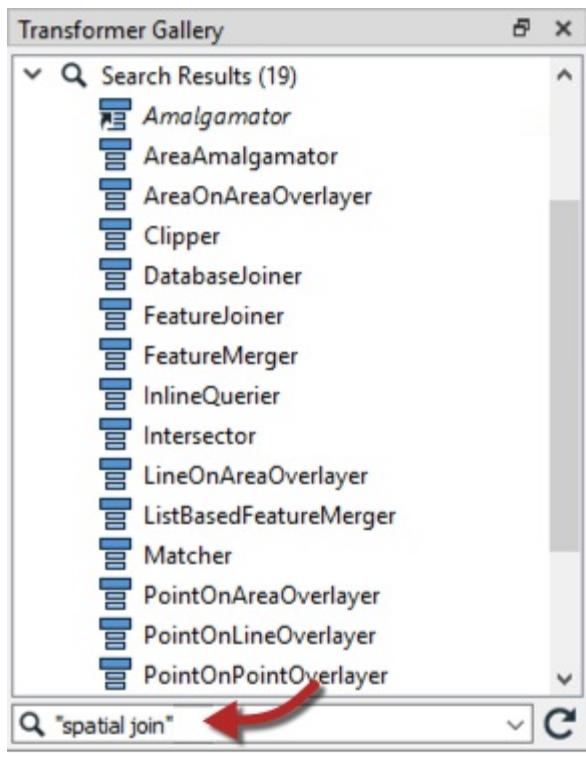
### Transformer Gallery Search

To perform a search in the transformer gallery, enter the search terms and either press the enter key or click the search icon (the binoculars icon).

The transformer gallery search searches in both name and description. Therefore a search term may be the exact name of a transformer, or it may be a general keyword referring to functionality in general:



Search terms can either be full or partial words and may consist of a number of keywords, including quote marks to enclose a single search reference:



## Quick Add Search

Quick Add search terms can also be full or partial words:

A screenshot of the FME Quick Add search interface. The search bar at the top contains the text "clip". Below the search bar, there are three main sections: "FME Transformers", "Custom Transformers", and "FME Hub Transformers". The "FME Transformers" section is expanded, showing a list of transformers: Clipper, MultiClipper, PointCloudZClipper, RasterClipOverlay, RasterGCPClipper, RasterGeoreferenceClipper, and WebMapTileClipper. The "Clipper" item is highlighted with a red arrow. To the right of the search results, there is a detailed description of the "Clipper" transformer, which states: "Performs a geometric clipping operation (sometimes called a cookie cutter). Most geometry types can be clipped by an area, and some may also be clipped by a solid. Attributes may be shared between objects (spatial join)". There is also a link to "Browse Additional Help ...".

By default, Quick Add does not look in transformer descriptions, so the search term must be the actual name of a transformer:

clipping



**FME Transformers**  
**Custom Transformers**  
**FME Hub Transformers**  
**Readers**  
**Writers**

No Results found. Try changing the [Search Mode](#)  
...or browse [FME Hub](#)

However, Quick Add will search in the transformer descriptions if you press the tab key:

clipping



**FME Transformers**  
Clipper  
PointCloudPropertyExtractor  
RasterBandAdder  
RasterResampler  
RasterSubsetter  
Recorder  
Tiler  
**Custom Transfomers**  
**FME Hub Transformers**  
PolygonSelfTouchRemover  
RasterGCPClipper  
SymbolFiller

### Clipper

Performs a geometric clipping operation (sometimes called a cookie cutter). Most geometry types can be clipped by an area, and some may also be clipped by a solid. Attributes may be shared between objects (spatial join).

[Browse Additional Help ...](#)

Quick Add results include aliases - for example, transformers that have an alternative name or which have been renamed - and also include transformers found in the FME Hub:

datecalculator



**FME Transformers**  
DateCalculator  
**Custom Transfomer**  
Alias for  
DateTimeCalculator  
**FME Hub Transform**  
ExcelDateCalculator  
RelativeDateCalculator  
**Readers**  
**Writers**

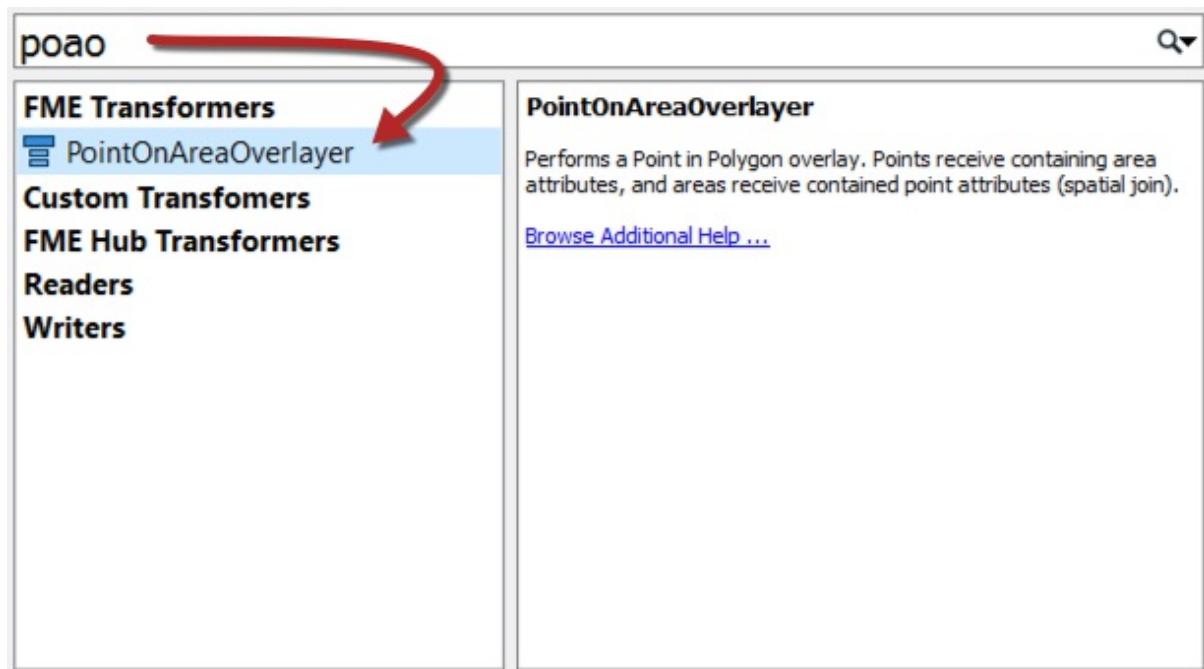
### DateTimeCalculator

Performs arithmetic on date, time, datetime, and interval values.

[Browse Additional Help ...](#)

## CamelCase

Quick Add also allows the use of CamelCase initials as a shortcut. CamelCase is where a single keyword is made up of several conjoined words, each of which retains an upper case initial; for example AttributeFileWriter (AFW) or ShortestPathFinder (SPF).



## FME Hub

FME Hub is an online facility for sharing FME functionality such as custom transformers, web connections, templates, formats and packages:



NEW TO FME HUB?

Contribute



# FME Hub

Discover new transformers, templates, web connections, and custom formats - and add them to your FME setup.

Filter (Clear All)



Search FME Hub...

Publisher

- Official
- Community

Item Type

- FME Desktop
  - Transformers
  - Formats
  - Templates
  - Web Connections
  - Packages

- FME Server
  - Projects

Include

- Deprecated Items

1 - 10 of 744 results
Newest

**Raster Object Detection Trainer**

Package created by safe | ★ 0 | ↓ 154 | Updated on Mar 20, 2019

Transformers to aid in creation of custom Raster Object Detection Models

**ScheduleFilter**

Transformer created by gavlepete | ★ 0 | ↓ 5 | Updated on Mar 17, 2019

Filters features based on a schedule. Let all features pass at a specified start date and repeat interval by filling in the tra...

**IMDFValidator**

Transformer created by safe-lab | ★ 1 | ↓ 169 | Updated on Mar 14, 2019

Validates features in IMDF files to see if they conform to the Apple specifications. Based on IMDF specification version 1....

Transformers from FME Hub are shown in Quick Add with a small, downwards-pointing arrow to denote that they will be downloaded if selected:

🔍

**FME Transformers**

**Custom Transfomers**

**FME Hub Transformers**

**MessageLogger**

**Readers**

**Writers**

**MessageLogger**

**Downloads: 2902**

The MessageLogger is designed to give some extra options beyond those available in the standard Logger transformer. The main capability is that it allows a Blue standard FME "WARN" message to be issued into the log window that can be easily spotted by a user. Most importantly though this warning will be summarised at the bottom of the workspace log like this:

**0.0 /INFORM /Translation was SUCCESSFUL with 1 warning(s) (0 feature(s) output)**

This capability allows features that arrive at the MessageLogger to issue a visible and summarised message.

**Input Ports**

**INPUT**

This transformer accepts any feature.

**Output Ports**

**OUTPUT**

## NEW

*Packages from FME Hub are new for 2019.0. They are a way to ship fully functional components outside of the main FME releases. Packages are fully portable, containing all required dependencies that they need in order to run. Python backed transformers, as well as Workbench created items, are supported.*

*Packages can be searched and downloaded the same way as custom transformers. They are denoted in Quick Add by a little box icon:*



## Most Valuable Transformers

If you have a thorough understanding of the most common transformers, then you have a good chance of being a very efficient user of FME Workbench.

Anyone can be proficient in FME using only a handful of transformers if they are the right ones!

### The Top 30

The [list of transformers](#) on the Safe Software website is ordered by most-used, calculated from user feedback. Having this information tells us where to direct our development efforts in making improvements, but it also gives users a head-start on knowing which of the (500+) FME transformers they're most likely to need in their work.

The following table (last updated April 2019) provides the list of the most commonly used 30 transformers. The Tester transformer is consistently number one in the list every year, highlighting its importance.

Rank	Transformer	Rank	Transformer
1	Tester	16	Counter
2	AttributeCreator	17	FeatureReader
3	AttributeManager	18	StringReplacer
4	FeatureMerger	19	StatisticsCalculator
5	Inspector	20	VertexCreator
6	AttributeKeeper	21	SpatialFilter
7	TestFilter	22	Sorter
8	Creator	23	AttributeExposer
9	AttributeRenamer	24	Bufferer
10	Reprojector	25	GeometryFilter
11	Aggregator	26	Dissolver
12	AttributeRemover	27	ListExploder
13	AttributeFilter	28	FeatureJoiner
14	Clipper	29	AttributeSplitter
15	DuplicateFilter	30	CoordinateExtractor

Besides the obvious transformers for transforming geometry (Clipper, Bufferer, Dissolver) and the obvious transformers for transforming attribute values (StringReplacer, Counter) there are some other distinct groups of transformers.

---

### Workflows

Workflow transformers - for example, the FeatureReader - can be found in the Workflows category in the Navigator window. They are for managing the flow of data through a workspace. Sometimes they are used to read data, to write data, or to connect to particular web services. They also integrate with FME Server to submit jobs and notifications.

### Managing Attributes

These transformers - mostly named the *Attribute<Something>* - are primarily for managing attributes (creating, renaming, and deleting) for schema mapping purposes. However, they can also be used to set new attribute values or update existing ones.

## Filtering

These transformers - mostly named the *<Something>Filter* - subdivide data as it flows through a workspace. Commonly the filter is a conditional filter, where the decision about which features are output to which connection is decided by some form of test or condition.

## Data Joins

Joins are the opposite action to filtering; they are when separate streams of data are combined as they flow through a workspace. Like filtering, there is a condition to be met - in this case matching key values - that determine how and where the join takes place.

## Exercise 1

## Transformer Selection Practice

<b>Data</b>	None
<b>Overall Goal</b>	Practice searching for and selecting transformers
<b>Start Workspace</b>	None
<b>End Workspace</b>	None

You've applied to become an **FME Certified Professional**. To help you practice for the exam, your colleagues have given you some questions to test your knowledge of searching for and selecting transformers.

Can you answer them?

### 1) Most-Used Web Transformers

1) The first thing your colleagues want to know is, what are the top 5, most-used transformers in the Web category?

- A. HTTPCaller
- B. JSONFlattener
- C. AttributeManager
- D. ParameterFetcher
- E. SalesforceConnector
- F. Clipper
- G. Generalizer
- H. JSONFragmenter
- I. SNSSender

### 2) Quick Add

Here is what appears to be a list of random characters, which transformers are returned by Quick Add for each string of random characters?:

1) lineco

- A. LineCompletion
- B. LightNewContainer
- C. LineCombiner
- D. LineCloser

2) reac

- A. AreaCalculator
- B. Reacher
- C. ReadEveryAreaCircle
- D. Recorder

3) rbic

- A. RasterBlenderIcon
- B. RasterBandInterpretationCoercer
- C. PointerBinTrainer
- D. RandomBlockImageContainer

4) drape

- A. SurfaceDrapier
- B. DissolveRasterAreaPolygonEvaluator
- C. Draper
- D. DrawPen

5) attributeexpl

- A. AttributeExporter
- B. AllTwentyTwoRasterImageBinsUserTrialEndEXporterPolygonLayer

- C. AttributeExploder
- D. AttributeExternal

**3) Transformer Searching**

Thirdly, your colleagues have come up with a list of different scenarios, and want you to search for a transformer to carry them out. Remember, there are often many ways to carry out a task in FME:

**1)** We have some lines of text in a file and want to read that text and add it as an attribute.

- A. AttributeFileReader
- B. CoordinateSwapper
- C. PointCloudThinner
- D. StringLengthCalculator
- E. NeighborPairFinder

**2)** We have a set of vector contours and want to create a cross-section by transposing the X and Z coordinates.

- A. AttributeFileReader
- B. CoordinateSwapper
- C. PointCloudThinner
- D. StringLengthCalculator
- E. NeighborPairFinder

**3)** We have a point cloud dataset and want to reduce its size by resampling it to remove excess points.

- A. AttributeFileReader
- B. CoordinateSwapper
- C. PointCloudThinner
- D. StringLengthCalculator
- E. NeighborPairFinder

**4)** We have a text string and want to find out how many characters the string contains.

- A. AttributeFileReader
- B. CoordinateSwapper
- C. PointCloudThinner
- D. StringLengthCalculator
- E. NeighborPairFinder

**5)** We have a set of addresses and for each address want to find the closest two libraries.

- A. AttributeFileReader
- B. CoordinateSwapper
- C. PointCloudThinner
- D. StringLengthCalculator
- E. NeighborPairFinder

## Workflow Transformers

Although data is usually read and written using readers and writers, it is also possible to use transformers to do the same tasks. These transformers appear in the Workflows category of the Transformer Gallery, along with transformers that are capable of handling data in web services.

---

### Key Workflow Transformers

Some of the key workflow transformers, and the tasks that they carry out, are as follows:

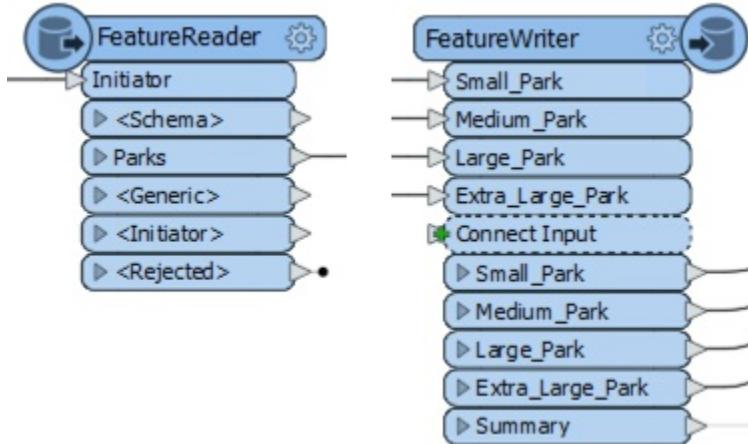
Task	Transformers
<b>Read and Write FME Formats</b>	FeatureReader, FeatureWriter
<b>Read and Write Web Services</b>	DropboxConnector, GoogleDriveConnector, SalesforceConnector, SlackConnector
<b>Read and Write Attributes</b>	AttributeFileReader, AttributeFileWriter
<b>Receive/Send Data Packets</b>	JMSSender/Receiver, SQSReceiver/Sender, TCPIPReceiver/Sender

These are part of a workflow because they don't necessarily read or write data at the start or end of a translation; they can be set up to read or write data mid-translation. This provides certain advantages over fixed-position readers and writers.

The FeatureReader is the 17th most-used transformer in FME, while the FeatureWriter just falls outside our most-valuable list, at number 31.

## FeatureReader and FeatureWriter

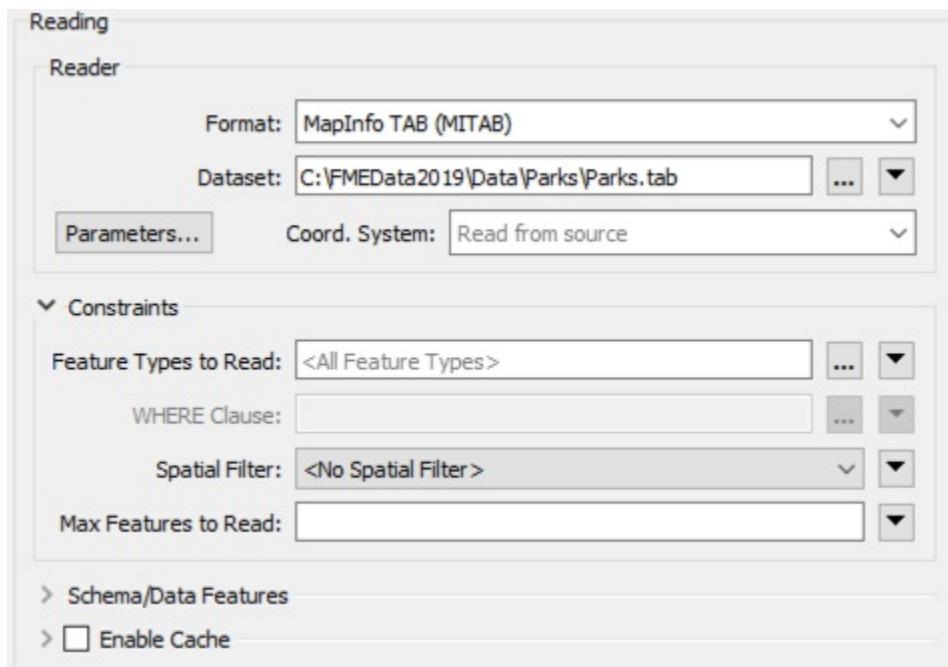
Besides being able to read data with a reader and writer, FME has transformers specifically designed to read and write data. These are the FeatureReader and FeatureWriter transformers.



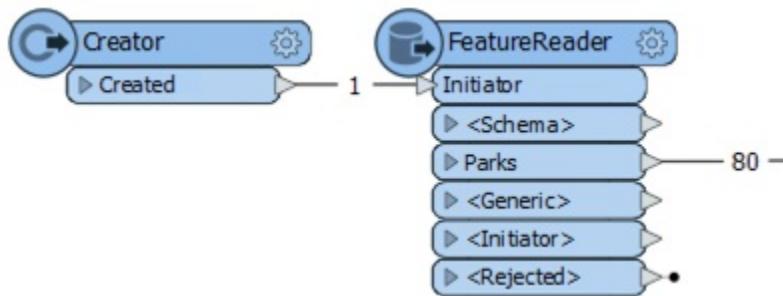
The advantage of these transformers is that they have an input port (FeatureReader) and output ports (FeatureWriter). So where a reader is always the first action in a workspace, and a writer is always the last, a FeatureReader and FeatureWriter can read and write data at any point in a translation.

### FeatureReader

The FeatureReader is set up with parameters to read a specific dataset:



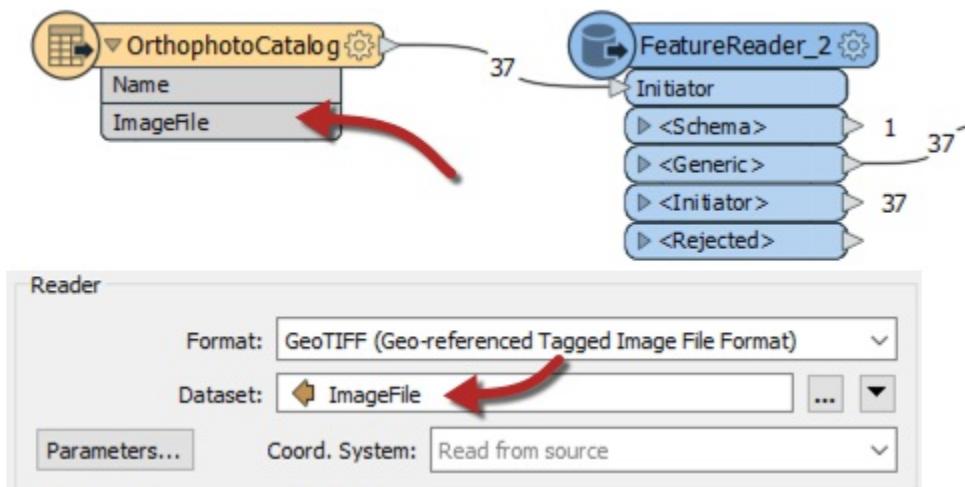
Any feature that enters the Initiator input port will cause the data to be read, like here where a Creator supplies a null feature to trigger reading:



The Creator creates a single feature that triggers the FeatureReader to read a dataset of park features. If for some reason the Creator created ten features, then the data would be read ten times, resulting in 800 output features!

## Dataset from an Attribute

A common case with the FeatureReader is to supply the dataset to read as an attribute:



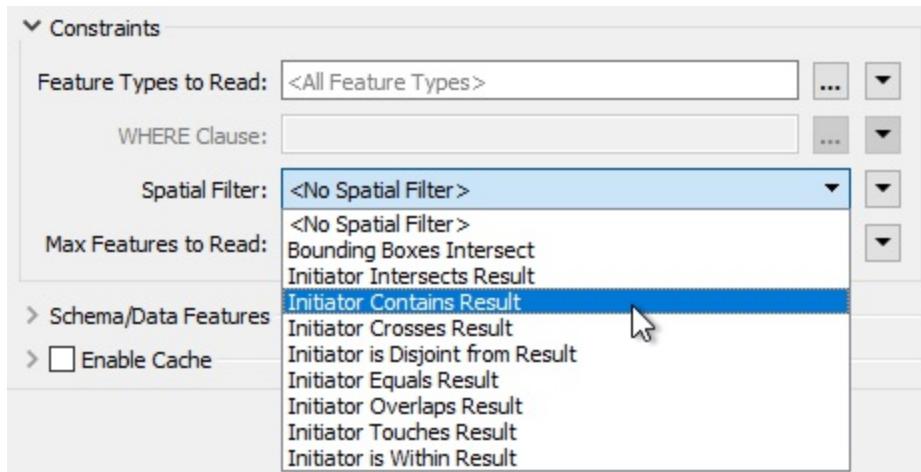
This example includes both reader and FeatureReader. The workspace reads a Shapefile dataset containing an index of orthophoto datasets. Each feature in the Shapefile is a polygon representing the orthophoto boundary with an attribute that points to a GeoTIFF file containing that orthophoto.

The FeatureReader is set up to use the attribute as the filename to read. The result is that 37 features are read from the Shapefile, and the equivalent 37 GeoTIFF images exit the FeatureReader.

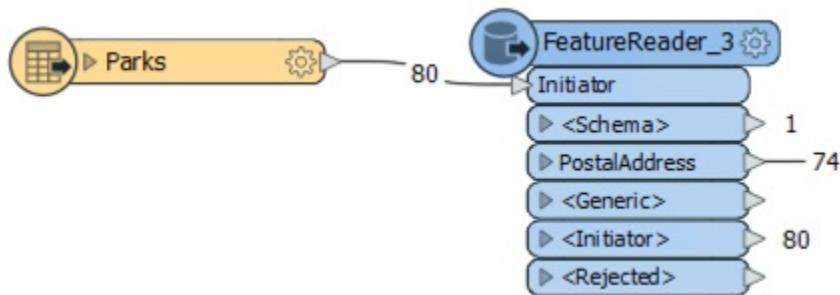
The source features do not need to be spatial. For example, an Excel spreadsheet with a list of files to read is just as valid.

## Spatial Filters

A key parameter in the FeatureReader sets a spatial filter on the data being read:



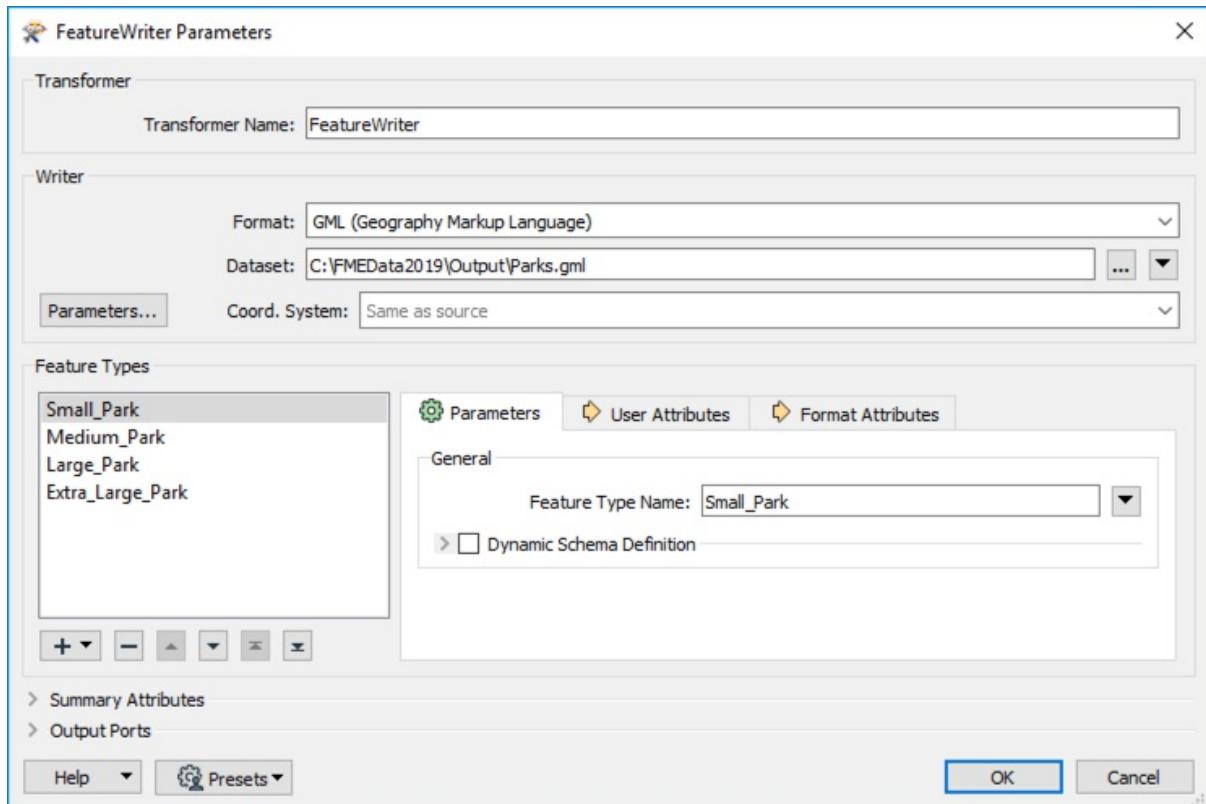
The Initiator Contains Result filter (for example) means that features output the FeatureReader if their geometry falls inside the geometry of the initiator feature. For example here:



A dataset of parks supplies input features that trigger reading from a database address table. A spatial filter is applied so that the only addresses to emerge are ones that fall inside a park.

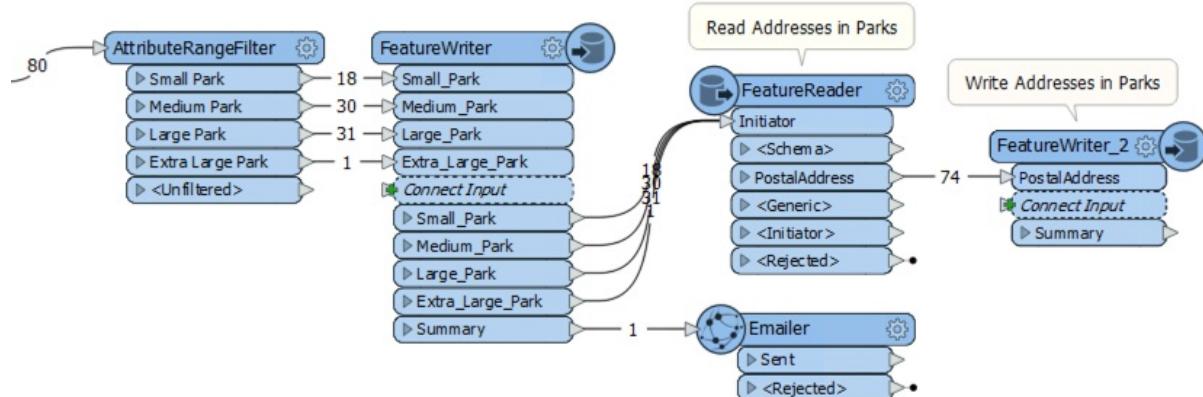
## FeatureWriter

The FeatureWriter is set up with parameters to write a specific dataset:



The dialog allows the definition of the format and dataset to write, plus the feature types that are to be written and their attributes. In short, all of the parameters, settings, and schema definition required for a writer, appear in this single dialog.

Feature types can be manually defined within the dialog itself, or can be added automatically by connecting to the *Connect Input* input port:

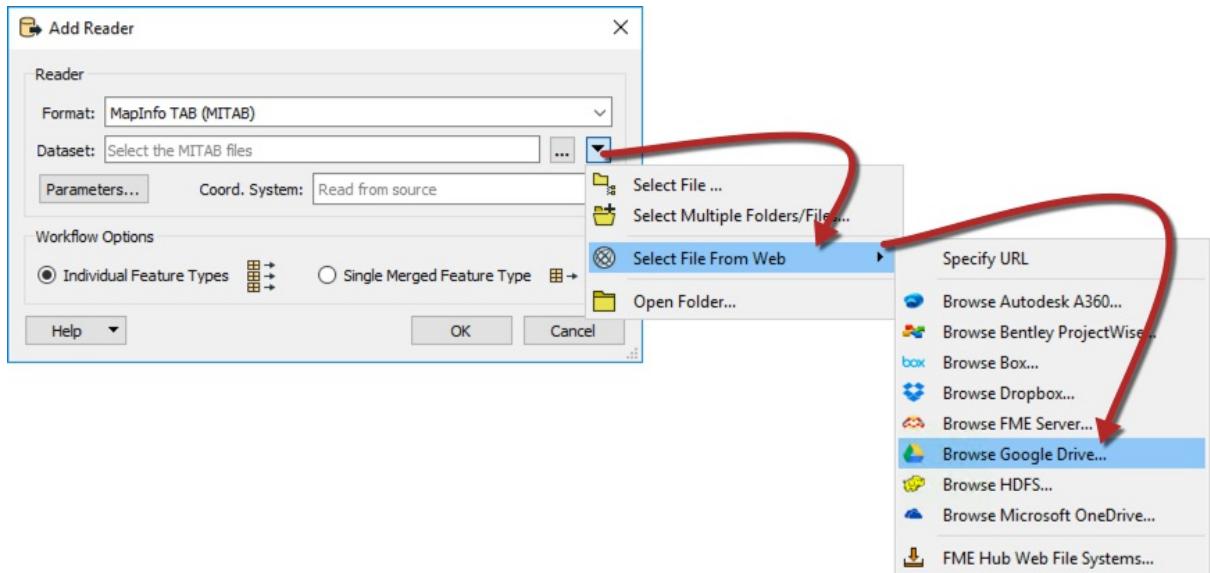


Notice also that an important part of the FeatureWriter is that its exit ports can be connected to other transformers for further processing. In the above screenshot parks data is written, and a single summary feature used to trigger an Emailer transformer. The Emailer sends a copy of the data to a user.

The data is also then used as the input to a FeatureReader to read all addresses that fall inside a park, the results of which are then also written with a FeatureWriter.

## Read/Write with Integration Transformers

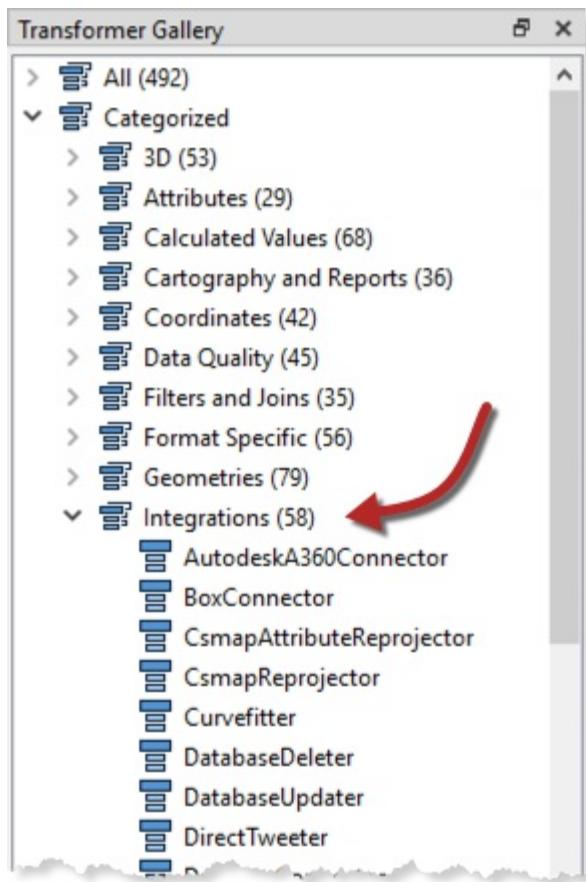
Readers and FeatureReaders can both read data locally, with a database, or using a web service:



However, sometimes it's useful to be able to use a transformer to read (or write) to a web or other integrated service. There are various reasons, but a key one is to be able to read a file itself, rather than reading features from it. For example, being able to extract a JPEG file from Dropbox and store it as an attribute (rather than reading it as an actual raster feature).

### Integration Transformers

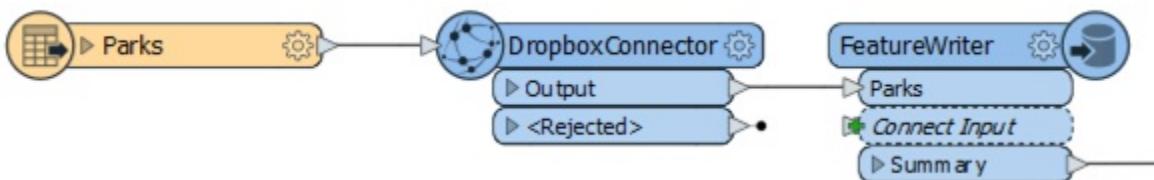
There are multiple transformers in the Integrations category of the Transformer Gallery:



Some of these integration transformers are for reading, writing, or copying files. For example:

- AutodeskA360Connector
- BoxConnector
- DropboxConnector
- FMEServerResourceConnector
- GoogleDriveConnector
- OneDriveConnector
- S3Connector

In this example the author is reading a dataset of parks and passing the features to a DropboxConnector:



Each park feature has the name of a JPEG image that is stored in Dropbox. The DropboxConnector transformer is set up to read that image and store it as an attribute (ParkImage) on the feature.

The features are then sent to a PostGIS database - using a FeatureWriter transformer - where the ParkImage attribute is written to a binary (bytea) column.

### TIP

*As well as fetching files from Dropbox, the DropboxConnector - as with most Connector transformers - is also capable of uploading content as a file.*



## Managing Attributes

A high proportion of the top 30 transformers are support transformers for managing attributes. These create new attributes, rename them, set values, and delete them.

A key use for these transformers is to rename attributes for schema mapping.

## Attribute Managing Transformers

The main attribute-management tasks and the transformers that can be used are as follows:

Task	Transformers
Create Attributes	AttributeCreator, AttributeManager
Set Attribute Values	AttributeCreator, AttributeManager
Remove Attributes	AttributeKeeper, AttributeManager, AttributeRemover, BulkAttributeRemover
Rename Attributes	AttributeManager, AttributeRenamer, BulkAttributeRenamer
Copy Attributes	AttributeCopier, AttributeCreator, AttributeManager
Sort Attributes	AttributeManager
Change Attribute Case	BulkAttributeRenamer
Add Prefixes/Suffixes	BulkAttributeRenamer

Many of these transformers can carry out similar operations, and you can see that the AttributeManager does so many tasks you can use it almost exclusively.

### WARNING

*Don't misunderstand the BulkAttributeRenamer. It changes the case - or adds suffixes/prefixes - to the attribute **name**, not the attribute value.*

## Lists

A **list** in FME is a mechanism that allows multiple values per attribute. So, where the attribute *myAttribute* can only contain a single value, the list attribute *myList{0}.myAttribute* can contain multiple values as:

```
myList{0}.myAttribute  
myList{1}.myAttribute  
myList{2}.myAttribute
```

For example, a single polygon representing a forested area, might have a list attribute to record the tree species contained in that area:

```
TreeList{0}.Species = Oak  
TreeList{1}.Species = Chestnut  
TreeList{2}.Species = Ash
```

Various transformers can create lists, others include support for lists, and several are designed to operate specifically on list attributes (for example the ListSorter).

---

For further reading check out [this article on Attribute Management](#) on the Safe Software blog.

## Creating and Setting Attributes

Creating attributes and setting a value are probably the primary attribute functions used within FME. When an attribute is created, its value can be set in a number of ways.

The transformers capable of creating an attribute - and setting its value - are:

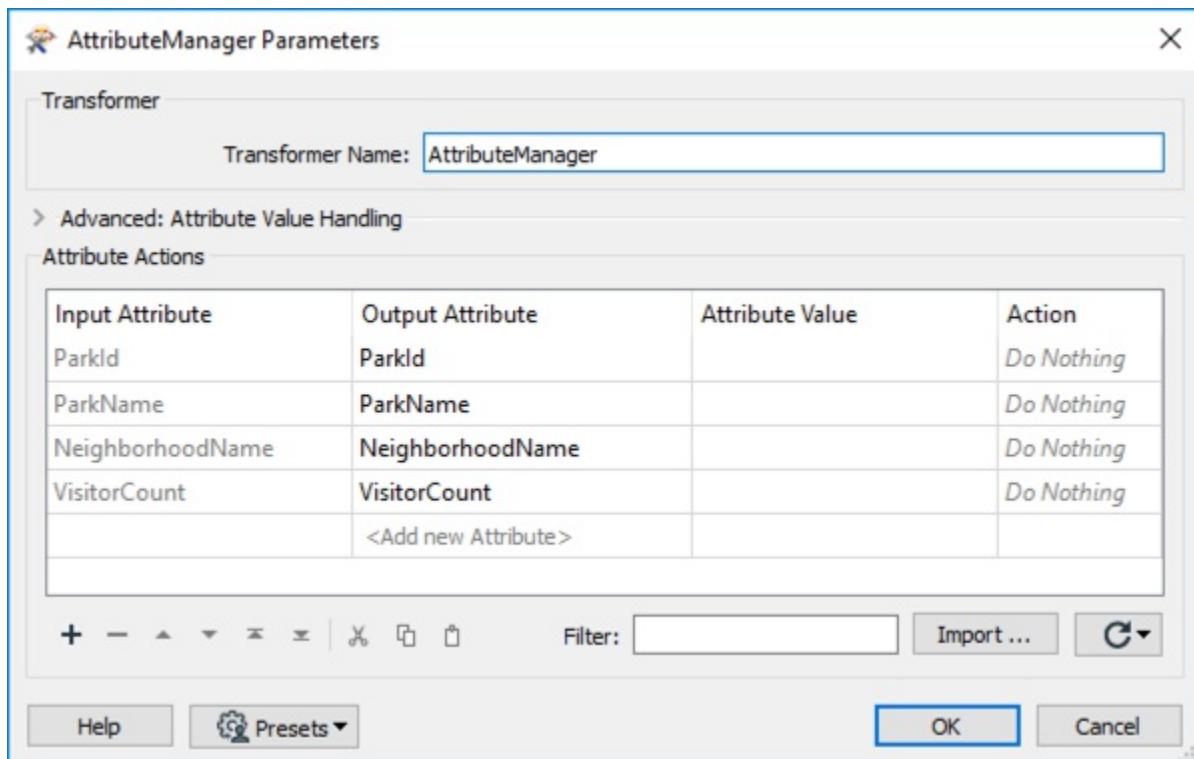
- AttributeCreator
- AttributeManager

**Note:** The *AttributeCopier* and *AttributeRenamer* transformers can set an attribute value, but only where the attribute doesn't already exist.

### The AttributeManager

For most operations we'll concentrate on the AttributeManager, so here is a quick overview of that transformer.

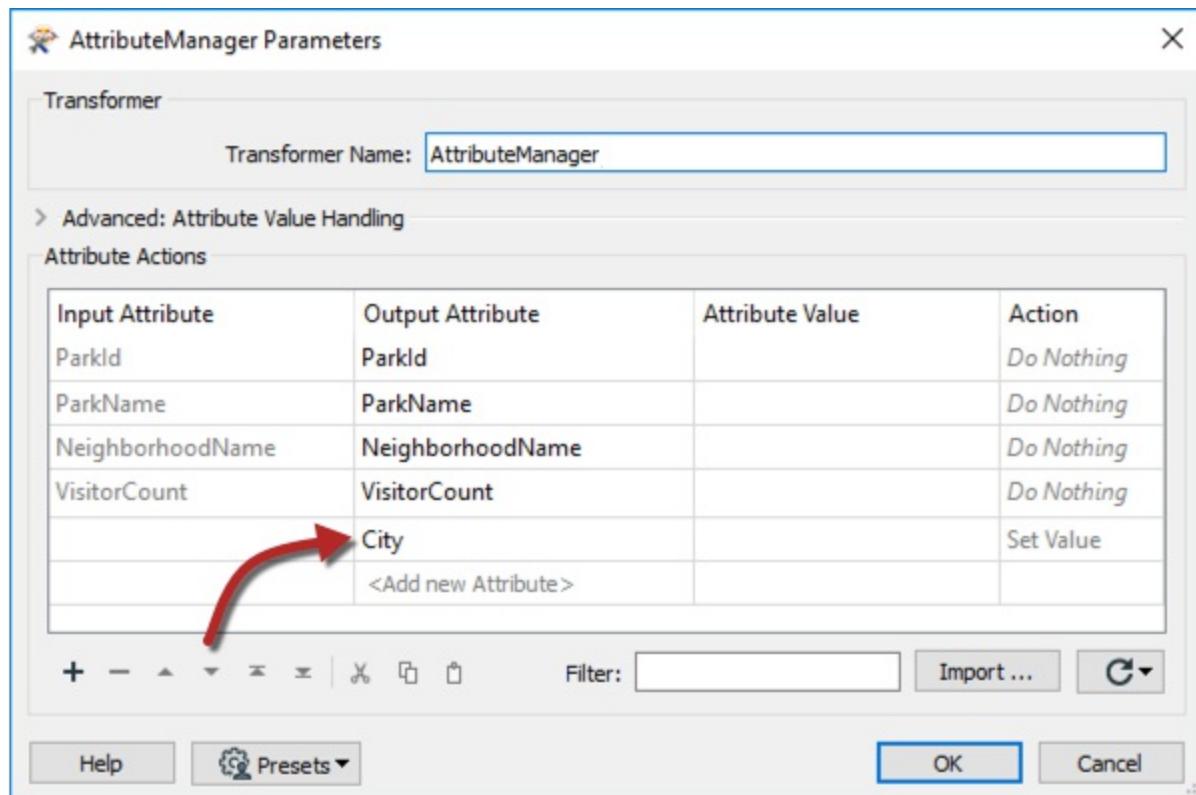
The AttributeManager parameters dialog has a number of fields: Input Attribute, Output Attribute, Attribute Value, and Action. Uniquely among attribute-handling transformers, it is automatically filled in with the details of the attributes connected to it:



The action field can be set by the user but is also set automatically when a change is made to the other fields.

### Manually Create an Attribute

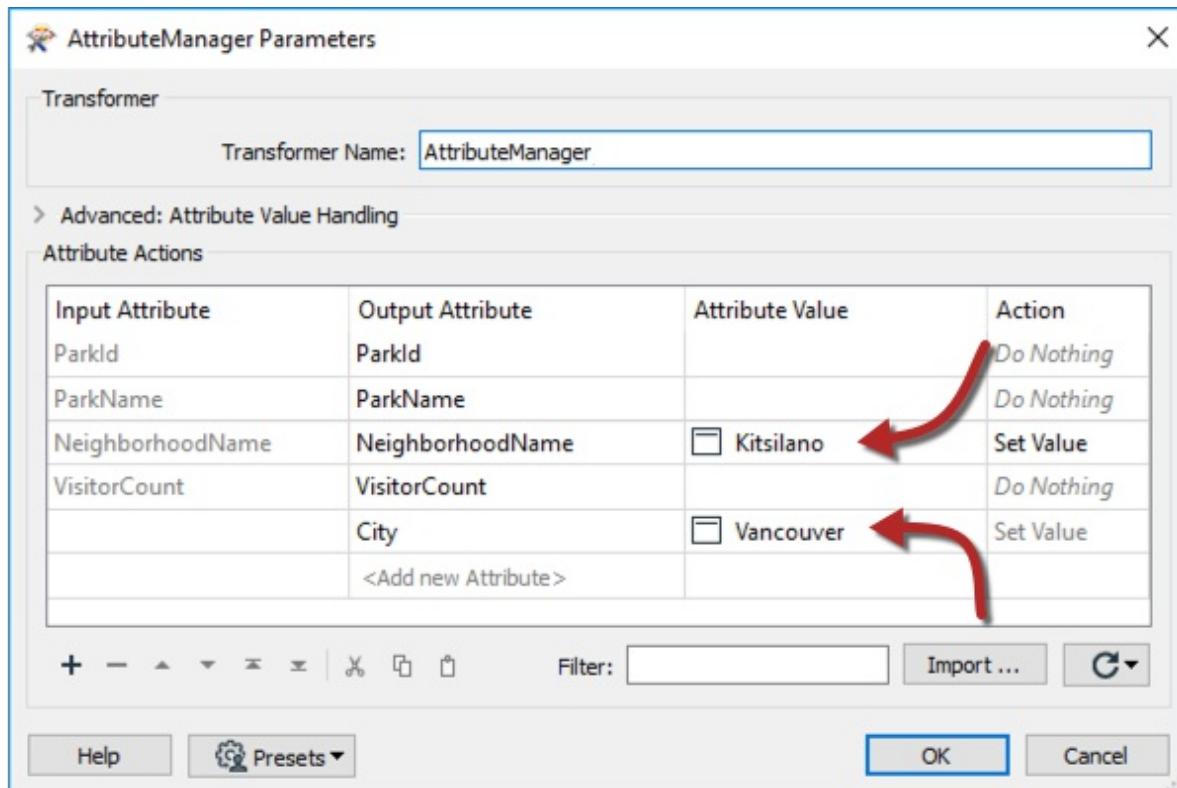
By entering a new attribute name into the Output Attribute field, it will be created in the output.



The text <Add new Attribute> highlights where a new attribute can be created. By default, when the Attribute Value field is empty, a new attribute has no value.

## Set a Fixed Attribute Value

A fixed (or *constant*) value for an attribute can be created by simply entering a value into the Attribute Value field:



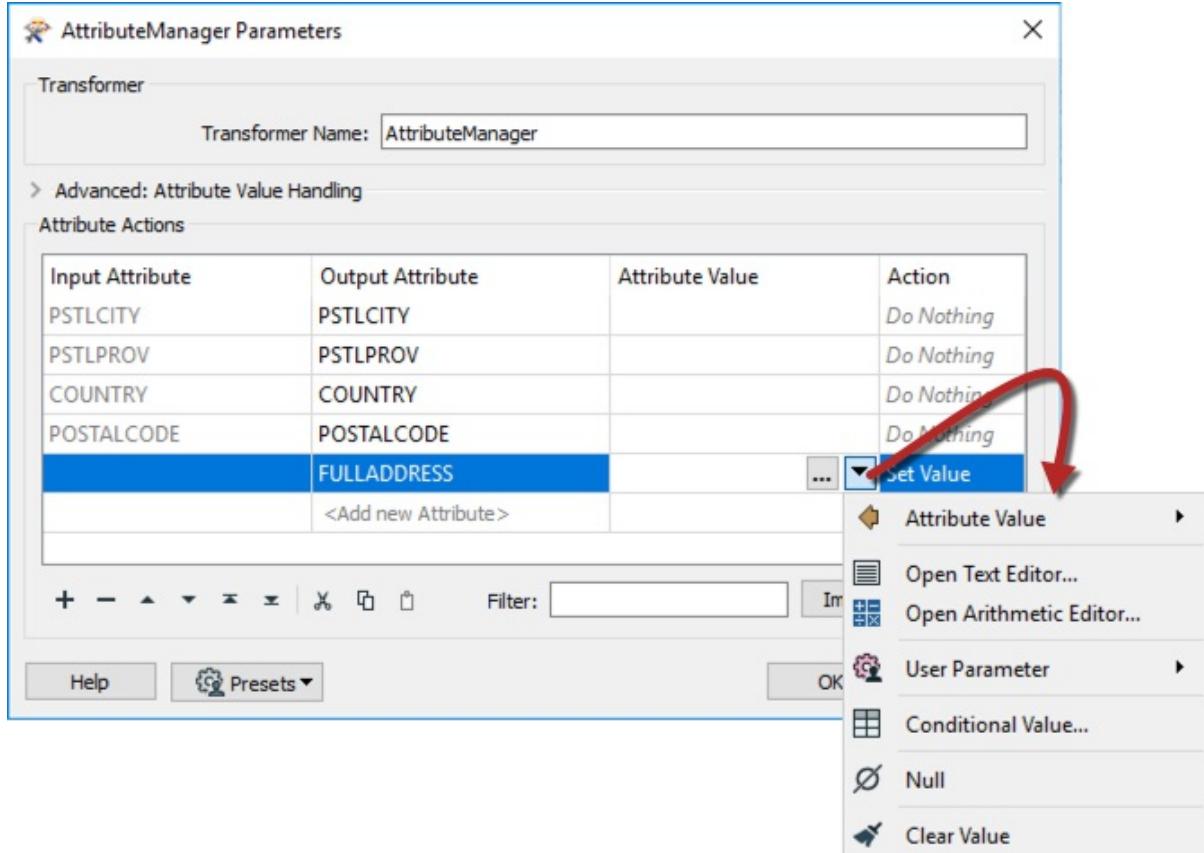
Here, for example, a new attribute called City is being given a fixed value of Vancouver.

However, also note that the existing attribute NeighborhoodName is also being assigned a fixed value. It has been given the value "Kitsilano." Notice how by entering a value into that field, the Action field has automatically changed from "*Do Nothing*" to "Set Value."

Besides entering set values like this, it's possible to construct an attribute value in a number of different ways...

## Constructing Attributes

Besides constant attribute values, FME also allows you to construct values using string manipulation and arithmetic calculations. This procedure is achieved by clicking on the arrow in the Attribute Value field and selecting either Open Text Editor or Open Arithmetic Editor:

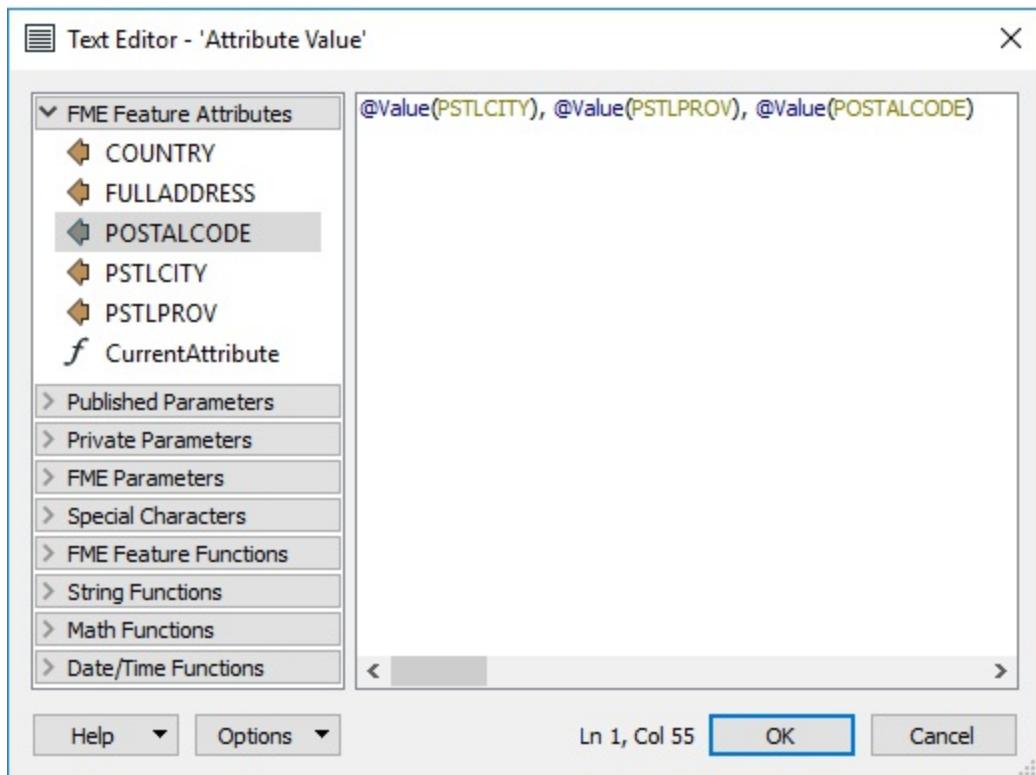


This method is advantageous because the attribute no longer needs to be a fixed value: it can be constructed from a mix of existing attributes, parameters, and constants.

## Text Editor

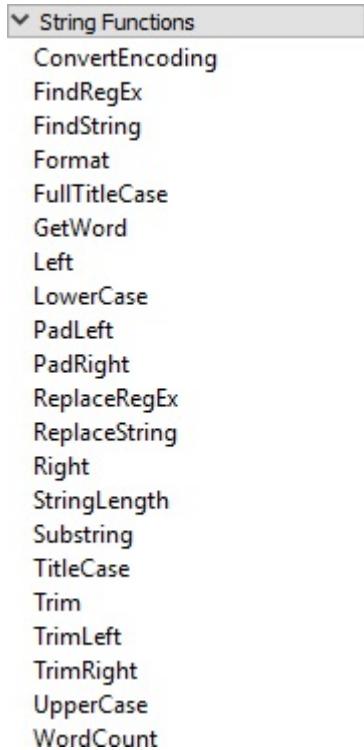
The text editor - as you would expect - allows you to construct a text value. It includes all the usual string-handling functionality you would need, such as concatenation, trimming, padding, and case-changing.

The text editor looks like this:



Here the user is constructing an address string by concatenating various existing attributes with some fixed characters (the commas).

Notice the menu on the left-hand side. Existing attributes are listed here and were added into the string by double-clicking them. Also, notice the other menu options. Maybe the most important for text are String Functions:



These are the functions that can be used to manipulate the strings being used. For example, here the user is making sure the attributes being used are trimmed when used:

```
@Trim(@Value(PSTLCITY)), @Trim(@Value(PSTLPROV)), @Trim(@Value(POSTALCODE))
```

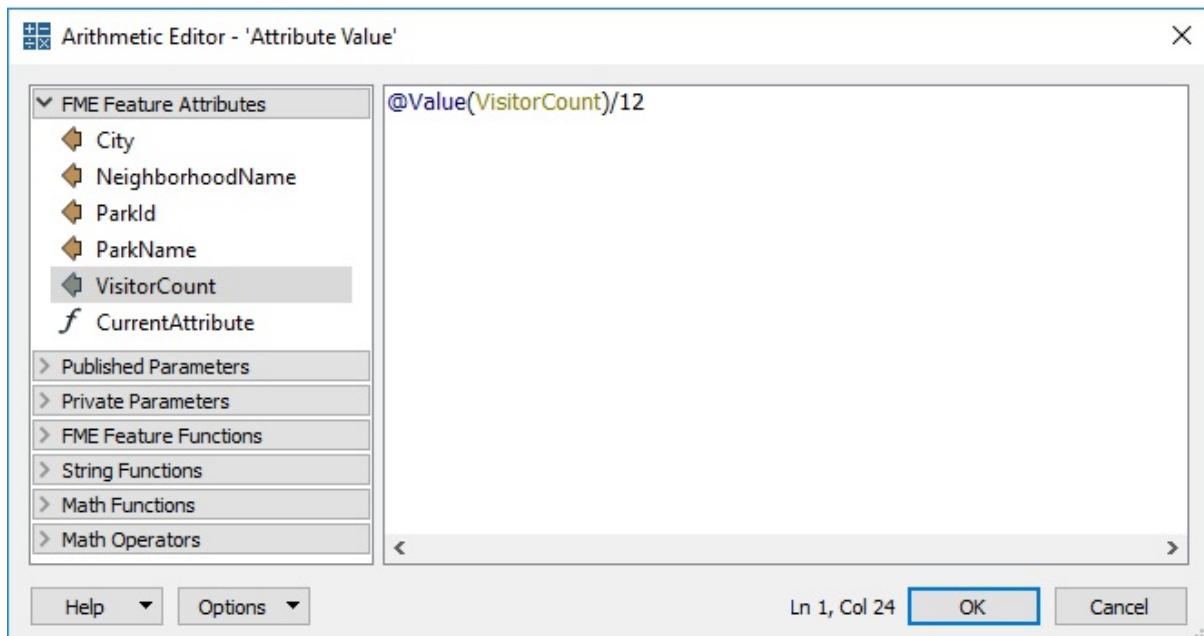


## TIP

Notice the Date/Time functions in the text editor, which can be used to manipulate dates, times, and datetime strings; including TimeZone components.

## Arithmetic Editor

The arithmetic editor is much the same as the text editor, except that whatever is entered into the dialog will be evaluated as an arithmetic expression, and a numeric result returned:



Here the user is calculating the monthly number of visitors to a park by dividing the annual number of visitors by 12 (twelve). As with the text editor, existing attributes and arithmetic functions were obtained from the menu on the left-hand side.

## WARNING

*The contents of the Arithmetic Editor **must** form an arithmetic expression that can be evaluated mathematically.*

## FME Feature Functions

One other item in the menu of both text and arithmetic editors is FME Feature Functions:

▼ FME Feature Functions	
Abort	
Area	Area
CoordSys	Count
Count	CurrentAttribute
CurrentAttribute	Dimension
Dimension	Evaluate
Evaluate	GeometryPartCount
GeometryPartCount	GeometryType
GeometryType	Length
Length	NumCoords
NumCoords	XValue
UUID	YValue
Value	ZValue
XValue	
YValue	
ZValue	

These are functions that reach into the very heart of FME's core functionality. They are the building blocks that transformers are built upon; basic functionality that can return values to the editor.

For example, the `@Area()` function returns the area of the current feature (assuming it is a polygon). `@CoordSys()` returns the coordinate system. They are the functional equivalent of the `AreaCalculator` and `CoordinateSystemExtractor` transformers.

Some functions return strings; others return numeric values. Therefore, the available functions vary depending on whether the text or arithmetic editor is being used. In the screenshot above, the text editor functions are on the left and the arithmetic editor functions on the right. The text editor can use either text or numeric values; the arithmetic editor can only ever accept numeric values.

FME Feature Functions are useful because they allow you to build processing directly into the `AttributeManager`, instead of using a separate transformer.

## Replacing Other Transformers

Integrated text and arithmetic editors provide a great benefit for workspace creation. They allow attribute-creating functions to be carried out directly in a single transformer.

For example, the `AttributeManager` text editor can be used as a direct replacement for the `StringConcatenator` and `ExpressionEvaluator` transformers.

The `AttributeManager` could also replace the `StringPadder` and `AttributeTrimmer` transformers, albeit with a little less user-friendliness. If FME Feature Functions are used inside the editor, this transformer could also technically replace transformers such as the `AreaCalculator`, `LengthCalculator`, `DateTimeStamper`, and many more.

This feature is usually a good thing. Workspaces will be more compact and well-defined when as many peripheral operations as possible are directly integrated into a single transformer. However, because it's possible for an `AttributeManager` to be carrying out many, many operations, it is also more important to use Best Practice and ensure it has the proper annotation.

If an `AttributeManager` is not properly annotated, it isn't possible to determine from looking at the Workbench canvas what action it is carrying out!

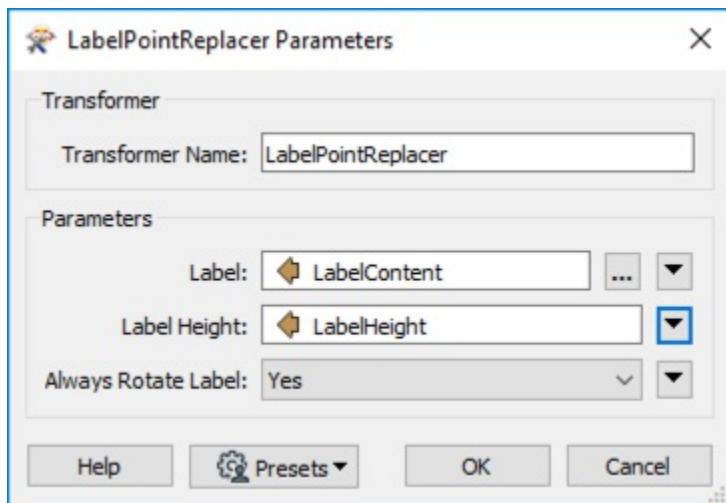
## Constructing Parameters

Let's just put attributes to one side for a moment and look at transformer parameters.

Transformer parameters are often set in a fixed way (hard-coded) or set to take on the value of a particular attribute. However, in the same way that attributes can be constructed, text or arithmetic editors can be used to build values for transformer parameters.

## Using Attributes for Parameters

As noted, most transformer parameters allow the user to select an attribute value instead of manually entering a fixed value. For example, the LabelPointReplacer can create a label whose contents and height are specified by attribute values:

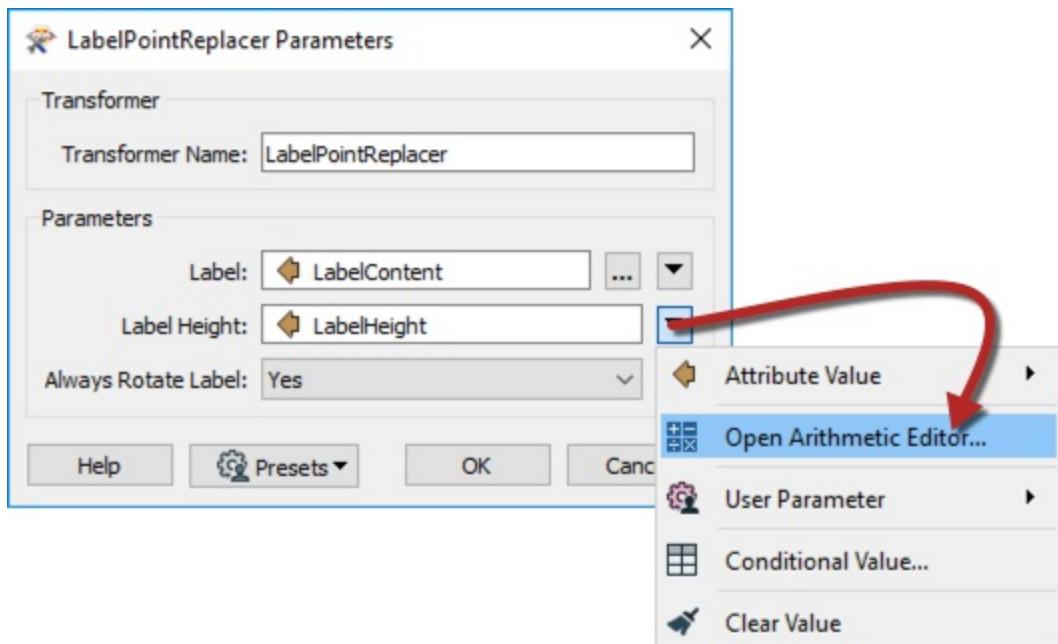


This feature is useful because it allows the parameters (for example label size) to get a different value for each feature. An attribute could be read from a source dataset, or calculated using an ExpressionEvaluator so that one feature creates a label ten units in height, another creates a label 15 units high, and so on. It is no longer a fixed value.

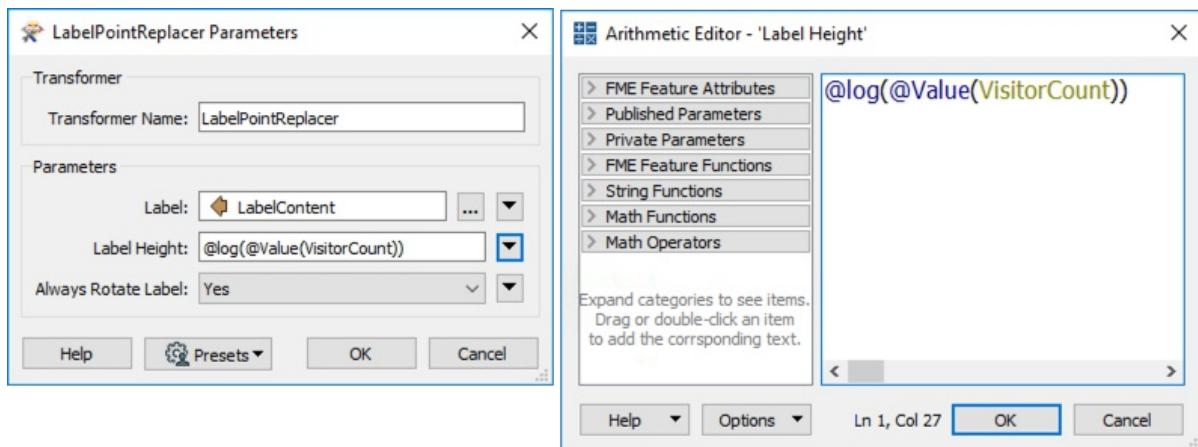
## Constructing Parameter Values

If a parameter value needs to be calculated or constructed, instead of using a separate transformer, FME has integrated string and numeric editors built into parameter dialogs.

For example, here the user is choosing to calculate label height using an arithmetic calculator:



The calculator allows the selection and use of FME attributes, other parameters, plus mathematical and string-based functions. For example, here the user has chosen to calculate the height of their labels using the logarithm of the visitor count for a park:



## FME Lizard says...

*It's a fixed rule that the editor dialogs available depend upon the type of parameter being set. For instance, the Label parameter in a LabelPointReplacer opens a text editor because the parameter requires a text value. The Label Height parameter opens an arithmetic editor because that parameter requires a numeric value.*

## Reducing Workspace Congestion

Like when attribute values are constructed, workspaces are more compact when as many peripheral operations as possible are directly integrated into a single transformer or parameter. However, as with attributes, it's important to add proper annotation, else it's difficult for a casual observer to understand what the workspace is meant to do.

Another drawback, specific to parameters, is that you don't also get the information as an attribute to use elsewhere. For example, if you construct a label string in the LabelPointReplacer, that string isn't available as an attribute elsewhere in the workspace.

## Renaming and Copying Attributes

Renaming and - to a lesser extent - copying attributes are also key attribute functions within FME. When an attribute is renamed it ceases to exist under its prior name; when it is copied, it exists both in its new and old names.

The transformers capable of renaming an attribute are:

Transformer	Capability
AttributeCopier	Copy
AttributeCreator	Copy
AttributeManager	Copy and Rename
AttributeRenamer	Rename

### Renaming

The fundamental purpose of renaming is to enter a new name for a selected attribute manually. The old attribute is removed and replaced with the newly named one:

Attribute Actions

Input Attribute	Output Attribute	Attribute Value	Action
GlobalID	GlobalID		Do Nothing
PSTLADDRESS	PostalAddress		Rename
PSTLCITY	PostalCity	<input checked="" type="checkbox"/> Vancouver	Rename
PSTLPROV	PostalProvince		Rename
COUNTRY	Country		Rename
	<Add new Attribute>		

+ - ▲ ▼ ⌂ ⌃ ⌄ Filter:  Import ... C

Here an AttributeManager is used to rename a number of fields by entering a different name for the Output Attribute. The Action is automatically set to Rename. Notice that the user is also entering a new constant value for the PSTLCITY/PostalCity attribute.

This type of behavior is obviously of use when the reader schema ('what we have') needs to be renamed to match the writer schema ('what we want').

### TIP

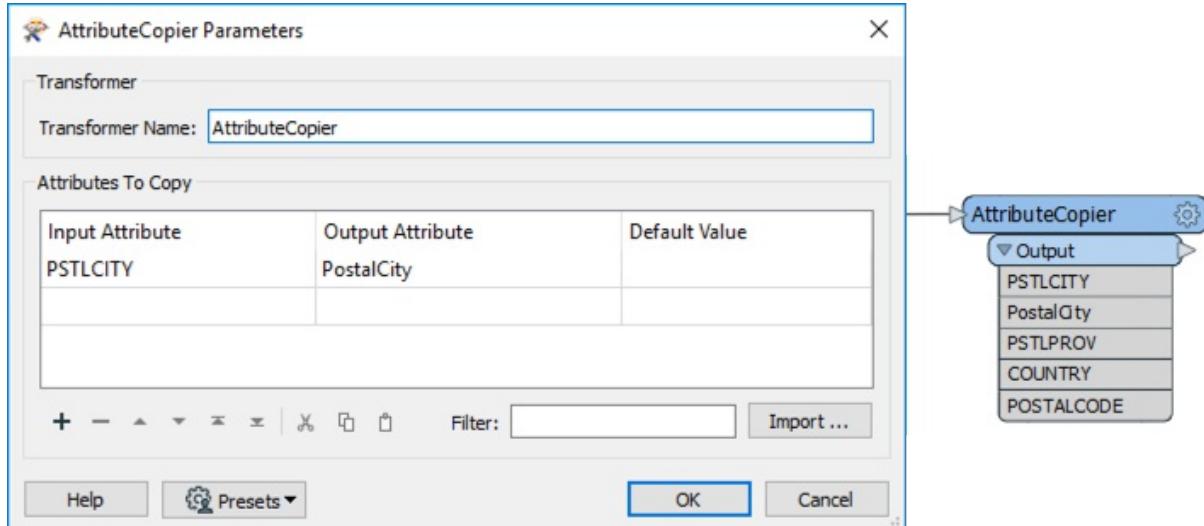
*Although you can manually type a new attribute name into the Output Attribute field, if the transformer is connected to a writer feature type with the correct attributes, its attribute names will be automatically available to select from:*

Attribute Actions			
Input Attribute	Output Attribute	Attribute Value	Action
GlobalID	GlobalID		Do Nothing
PSTLADDRESS	PostalAddress		Rename
PSTLCITY	PSTLCITY		Do Nothing
PSTLPROV	mapinfo_text_line_pen_pattern		Do Nothing
COUNTRY	mapinfo_text_line_pen_width		Do Nothing
	mapinfo_text_linetype		
	mapinfo_text_spacing		
	mapinfo_text_string		
	mapinfo_text_width		
	mapinfo_type		
	PostalAddress		
	PostalCity		
	PostalProvince		

+ - ▲ ▼ ⌂ ⌃ ⌄ Filter: Import ... C

## Copying

Depending on the transformer, copying an attribute can be one of two styles.



Here the AttributeCopier consists of selecting the existing attribute and entering a new name for it. Again, when connected to a writer feature type, its schema is available to use.

Note how both PSTLCITY and PostalCity exist on the output of the transformer, proving that it is copying the attribute rather than renaming it.

### TIP

*You can (as above) enter a constant attribute value in the AttributeCopier, but in reality it's hardly a copy operation in that case; it's more an attribute creation task.*

For other transformers, the setup style is reversed: a new attribute is created and given the value of an existing attribute:

Attribute Actions			
Input Attribute	Output Attribute	Attribute Value	Action
PSTLCITY	PSTLCITY		<i>Do Nothing</i>
PSTLPROV	PSTLPROV		<i>Do Nothing</i>
COUNTRY	COUNTRY		<i>Do Nothing</i>
POSTALCODE	POSTALCODE		<i>Do Nothing</i>
	PostalCity	↳ PSTLCITY	Set Value
	<Add new Attribute>		

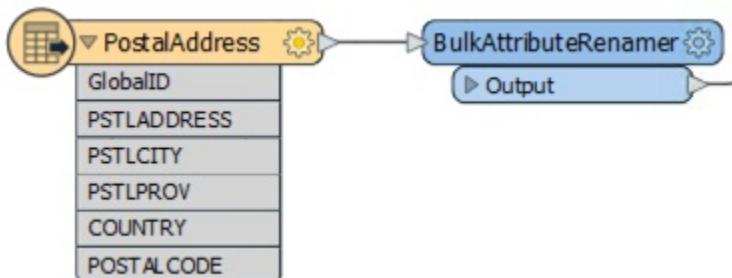
+ - ▲ ▼ ⌂ ⌃ | ⌂ ⌃ ⌄ Filter:  Import ... C ▾

In this AttributeManager transformer, the user creates a new attribute (PostalCity) and assigns it the value from another (PSTLCITY). In effect, they have made a copy of the original attribute.

## Bulk Attribute Renaming

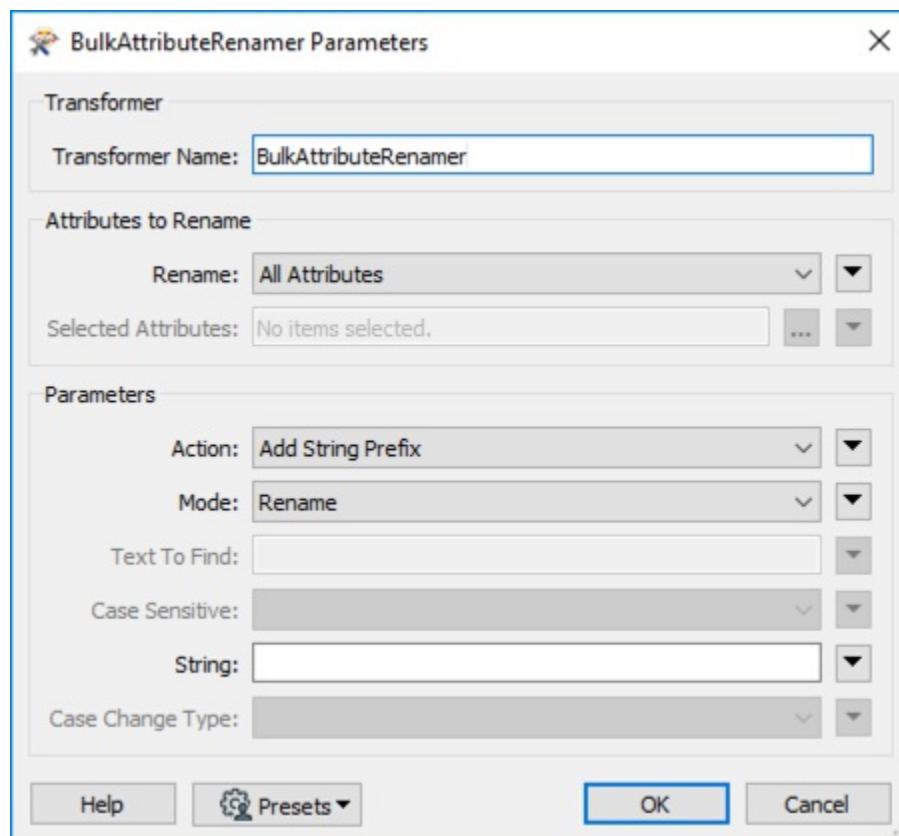
Usual attribute renaming involves selecting individual attributes to modify. However, in some cases, it's important to be able to carry out the same renaming operation on a large number of attributes.

This scenario is catered for by the BulkAttributeRenamer transformer.



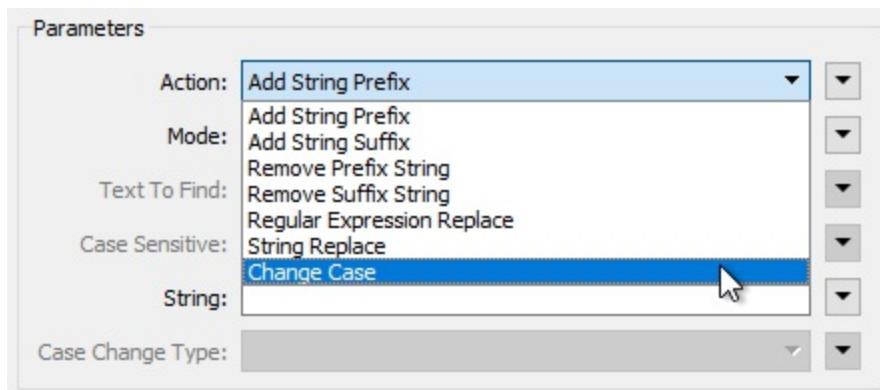
### BulkAttributeRenamer

The BulkAttributeRenamer carries out the core function of renaming attributes. But instead of manually specifying each attribute, this transformer lets the user select multiple attributes - or all of them:

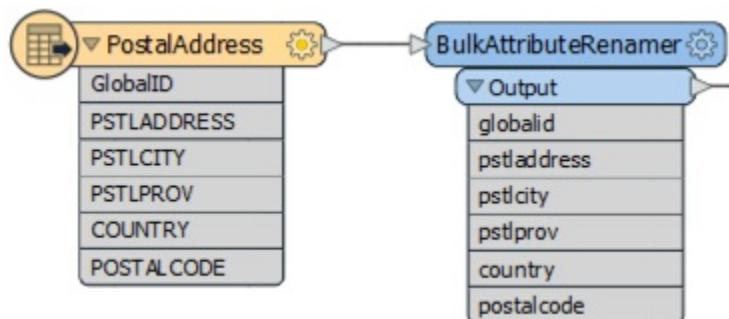


When multiple attributes are selected, the action must - of course - carry out the same renaming action on them all. These actions are:

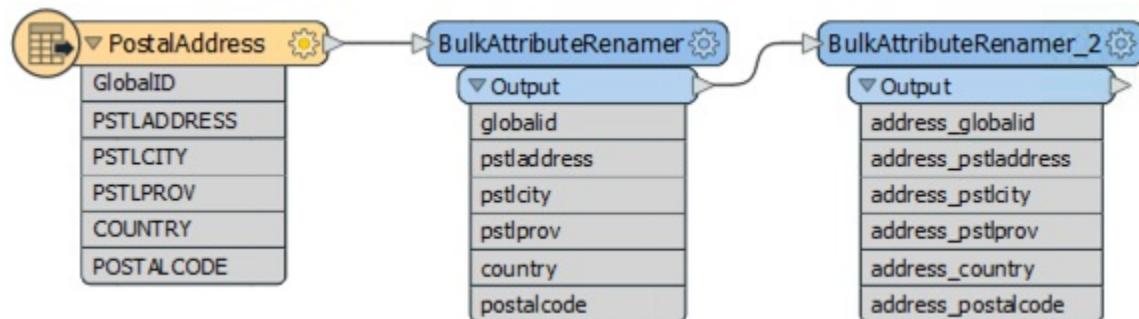
- Add String Prefix
- Add String Suffix
- Remove Prefix String
- Remove Suffix String
- Regular Expression Replace
- String Replace
- Change Case



The power of the transformer is also in its ability to manipulate multiple attributes at once, without having to select them all individually. Here, for example, the incoming attributes are all being renamed to lower case names to match a writer schema that does not support upper case:



Multiple transformers can be used to create a cumulative effect. Here, for example, the user has converted to lower case and then used a second transformer to add a prefix:



## Removing Attributes

Removing attributes is perhaps seen as a less important task in FME. That's because - for a manual attribute schema - only attributes defined in the writer are written to the output; extra attributes that are not required are just ignored.

However, removing attributes does carry useful benefits:

- Removing attributes that aren't required tidies up a workspace and makes it easier to understand
- A workspace is a complex network of objects and schemas. Removing attributes simplifies this network and makes the Workbench interface more responsive
- All data processing incurs costs of time and memory. Removing attributes means less data is being processed and so the FME engine performs faster

### TIP

*A reader feature type has parameters to hide attributes from the schema. This helps tidy a workspace, but does not help improve the translation performance. However, some formats (mostly databases) also have a setting for "Attributes to Read" and using this will help performance.*

Transformers that can remove attributes are:

- AttributeKeeper
- AttributeManager
- AttributeRemover
- BulkAttributeRemover

## Removing Attributes

The AttributeManager and AttributeRemover have the same technique; select an attribute to be removed:

Attribute Actions			
Input Attribute	Output Attribute	Attribute Value	Action
ParkId	ParkId		Do Nothing
RefParkId	RefParkId		Do Nothing
ParkName	ParkName		Do Nothing
NeighborhoodName	NeighborhoodName		Do Nothing
VisitorCount	VisitorCount		Do Nothing
TreeCount	TreeCount		Do Nothing
DogPark	DogPark		Remove
Washrooms	Washrooms		Remove
SpecialFeatures	SpecialFeatures		Remove
	<Add new Attribute>		

+ - ▲ ▼ ⌂ ⌂ | ⌂ >> Filter:  Import ... C ▾

Attributes can be removed in the AttributeManager by selecting it and clicking the - button. Alternatively, you can change the action field from *Do Nothing* to Remove.

Notice in the above that three attributes have been removed. The output attribute (when selected) shows the name struck out to signify that it is no longer present.

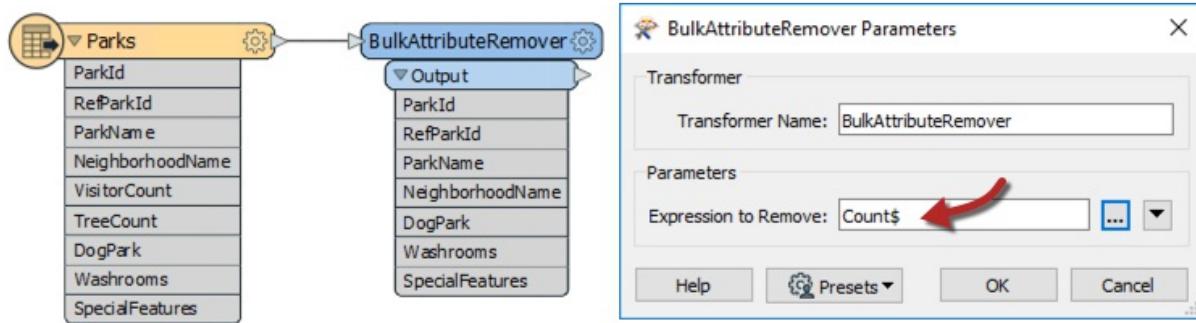
## Keeping Attributes

The AttributeKeeper transformer carries out the same function but approaches it from the opposite direction. It lets the user specify which attributes are **not** to be removed; in other words, this transformer lets the user specify which ones to keep.

So, the AttributeManager should be used where one or two attributes are to be removed, but the majority of them kept. The AttributeKeeper should be used when the majority of attributes are to be removed, and only one or two of them retained.

## Bulk Attribute Removal

The BulkAttributeRemover - like the BulkAttributeRenamer - lets the user carry out a process on multiple attributes. In this case, instead of being able to select all attributes, the user enters a string-matching expression to define which attributes to remove:



Here the user is removing all attribute whose name ends in the word "Count."

## Exercise 2

## Noise Control Laws Project (Addresses)

<b>Data</b>	Addresses (File Geodatabase)
<b>Overall Goal</b>	Convert a File Geodatabase to Microsoft Excel and map the schema
<b>Demonstrates</b>	Attribute Management for Schema Mapping
<b>Start Workspace</b>	None
<b>End Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\Transformers-Ex2-Complete.fmw

City councilors have voted to amend noise control laws and residents living in affected areas must be informed of these changes.

You have been recommended by your manager to take on the task of finding all affected addresses. There's a tight deadline, and at least three city councilors are standing watching you work. The pressure is on, and it's up to you to deliver!

This exercise is the first part of the project. You know that the address database for the city is stored in an Esri Geodatabase whose schema matches the Local Government Information Model PostalAddress table.

However, you are told that the software used to carry out automated bulk mailings requires addresses stored in an Excel spreadsheet using a completely different schema.

So, your first task is to create a workspace that converts addresses from Geodatabase to Excel, mapping the schema at the same time.

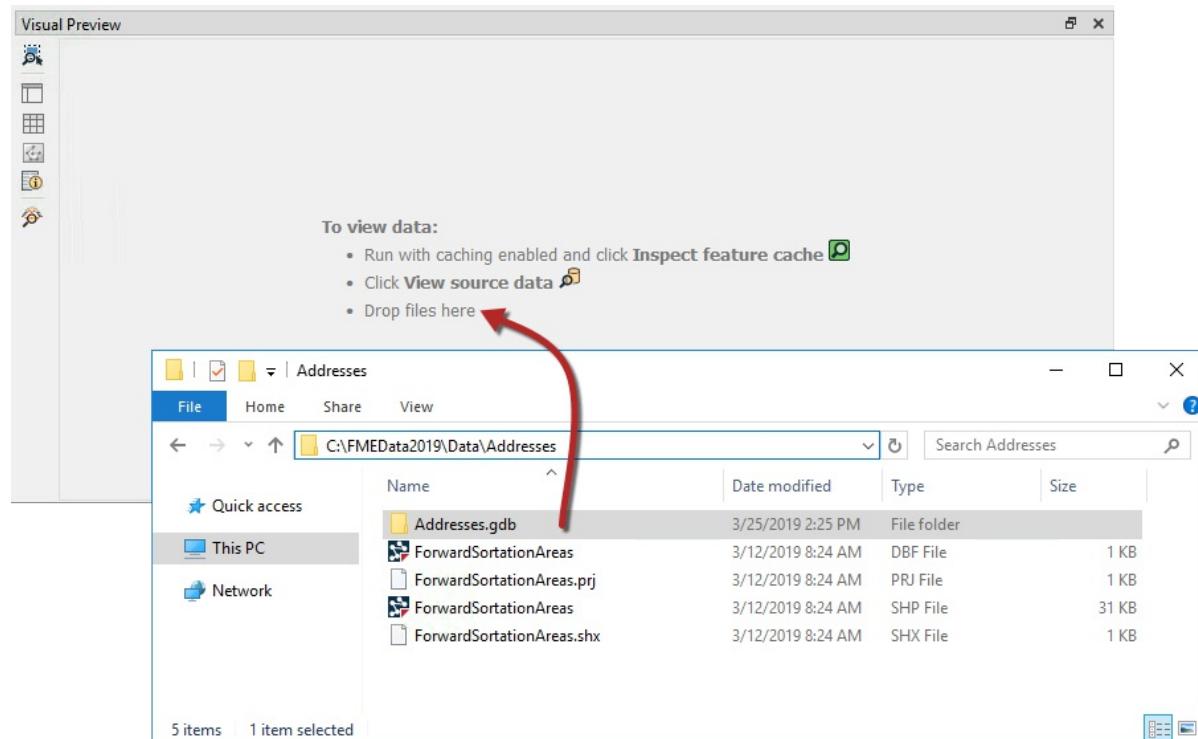
### 1) Open a blank workspace

As usual, the first task is to familiarize yourself with the data. To do this open a blank workspace and ensure that Visual Preview is open: View > Windows > Visual Preview.

Open your computer's file explorer and browse to the dataset:

```
C:\FMEData2019\Data\Addresses\Addresses.gdb
```

Then drag and drop the Addresses.gdb in the Visual Preview window:



Once the Geodatabase has been dropped onto Visual Preview, a Select Dataset to View dialog will appear. Ensure that the Format is Esri Geodatabase (File Geodb Open API) and then click OK. Then ensure that the Table View is open and the Table is showing PostalAddress. For more space you can close the Graphics View as we are only interested in the table:

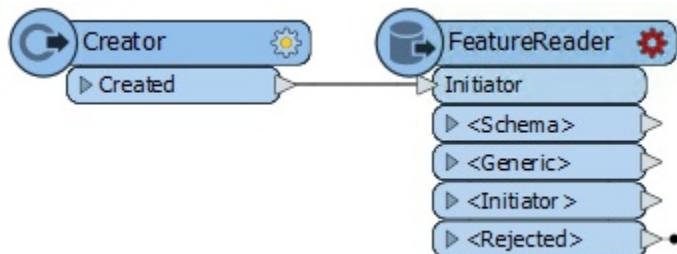
	OBJECTID	GlobalID	OWNERNM1	OWNERNM2	PSTLADDRESS	PSTLCITY	PSTLPROV	IN ^
1		1 1C55C207-5A3E...	Jake Warnock	<null>	1188 W Pender St	Vancouver	British Columbia	C^
2		2 8AFE5C37-E0A7...	Armand Augustyn	<null>	1661 Ontario St	Vancouver	British Columbia	C^
3		3 6963B7DB-653A...	Lieselotte Cota	<null>	535 Smithe St	Vancouver	British Columbia	C^
4		4 67133025-D2F6...	Lieselotte Cota	<null>	181 W 1st Av	Vancouver	British Columbia	C^
5		5 3E5D9415-BDD...	Ernest Ahlgren	<null>	141 W 1st Av	Vancouver	British Columbia	C^
6		6 E191FAAD-295B...	Jim Baskerville	<null>	808 Gore Av	Vancouver	British Columbia	C^
7		7 9E7BDFEB-2288...	Cassandra Bran...	<null>	266 E 15th Av	Vancouver	British Columbia	C^
8		8 9B11F75F-3443...	Caryl Chinn	<null>	36 Water St	Vancouver	British Columbia	C^
9		9 42946EE8-2B6A...	Gerald Woodburn	<null>	2762 W 3rd Av	Vancouver	British Columbia	C^
10		10 591FE3D2-95E0...	Many Purdy	<null>	989 Main St	Vancouver	British Columbia	C^

## 2) Add Creator and FeatureReader Transformers

Now that you are familiar with the source data, we can add the data to the workspace.

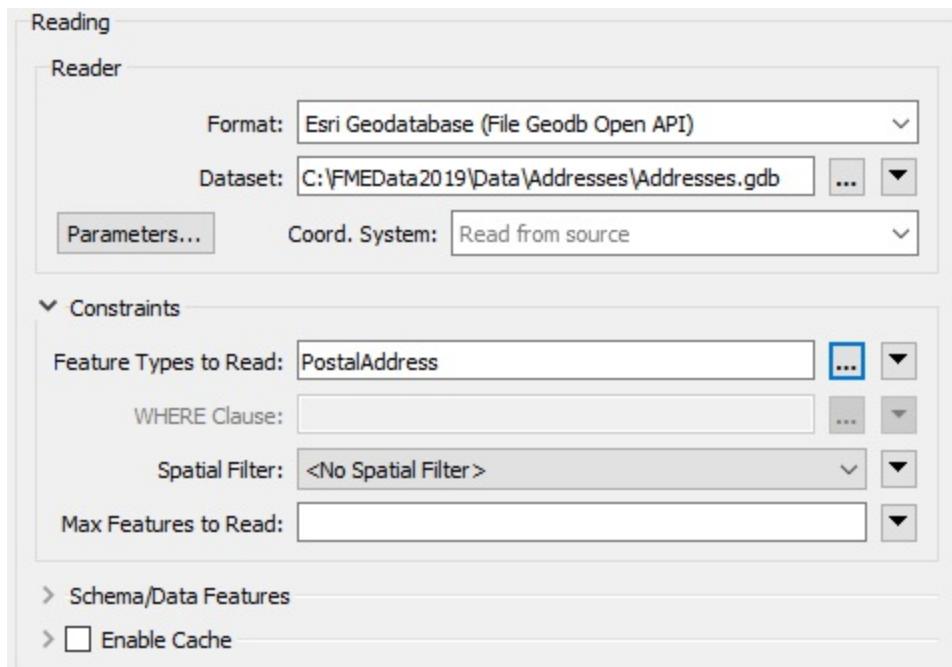
We can choose to read the source data using either a reader or a FeatureReader transformer. The FeatureReader will allow us to build in a spatial filter so - because we believe this project may need some filtering - we'll use a FeatureReader transformer with a Creator to create a feature to trigger it.

So place a Creator transformer and connect it to a FeatureReader:



Inspect the FeatureReader parameters and set up the parameters as follows:

<b>Reader Format</b>	Esri Geodatabase (File Geodb Open API)
<b>Reader Dataset</b>	C:\FMEData2019\Data\Addresses\Addresses.gdb
<b>Feature Types to Read</b>	PostalAddress



## TIP

*Just as with the Visual Preview window, datasets can be dragged/dropped into dataset fields in Workbench dialogs.*

*In this case the same .gdb folder could be dropped onto the Reader Dataset fields in the FeatureReader transformer, instead of using the browse for data button.*

### 3) Add an Excel Writer

Now let's add a writer to write the output data. There currently seems no benefit or need to use a FeatureWriter, so select Writers > Add Writer from the menu bar and use the following:

<b>Writer Format</b>	Microsoft Excel
<b>Writer Dataset</b>	C:\FMEData2019\Output\Training\AddressFile.xlsx
<b>Sheet Definition</b>	Import from Dataset



Setting 'Import from Dataset' will let us import an Excel spreadsheet to use as a guide. Click OK to add the writer.

#### 4) Import Feature Types

At this point you are prompted to select the dataset to import a schema definition from. The two fields should be set up with the same values as the writer. Set the Dataset parameter as follows:

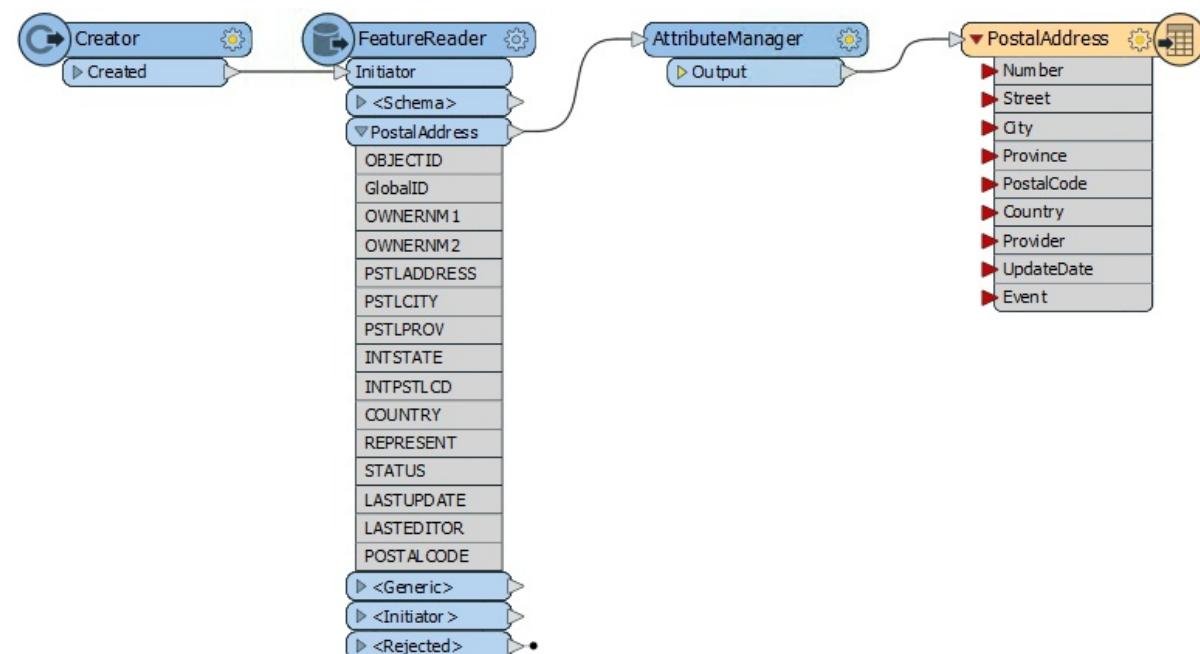
Reader Dataset	C:\FMEData2019\Resources\DesktopBasic\AddressSchema.xlsx
----------------	--

This file is our guide/template. Click OK to accept the values. The new feature type will be created to match the chosen Excel schema.

#### 5) Add an AttributeManager Transformer

Now we can start to map the schema from the reader (FeatureReader) to the writer. As you'll have noticed, the two do not currently match up very well.

So, place an AttributeManager connected between the FeatureReader:PostalAddress output port and the PostalAddress writer feature type.



Its parameters will look like this:

Attribute Actions			
Input Attribute	Output Attribute	Attribute Value	Action
OBJECTID	OBJECTID		Do Nothing
GlobalID	GlobalID		Do Nothing
OWNERNM1	OWNERNM1		Do Nothing
OWNERNM2	OWNERNM2		Do Nothing
PSTLADDRESS	PSTLADDRESS		Do Nothing
PSTLCITY	PSTLCITY		Do Nothing

Firstly let's clear up the reader schema by removing some of the unwanted attributes.

Click on the following attributes and press the - button to remove them:

- OBJECTID
- GlobalID
- OWNERNM1
- OWNERNM2
- INTSTATE

- INTPSTLCD
- REPRESENT
- STATUS
- LASTUPDATE
- LASTEDITOR

Attribute Actions

Input Attribute	Output Attribute	Attribute Value	Action
OBJECTID	OBJECTID		Remove
GlobalID			Remove
OWNERNM1			Remove
OWNERNM2			Remove
PSTLADDRESS	PSTLADDRESS		Do Nothing
PSTLCITY	PSTLCITY		Do Nothing
PSTLPROV	PSTLPROV		Do Nothing

Buttons: + - ▲ ▼ ↕ ↖ ↘ Filter: Import ... C Help Presets OK Cancel

## 6) Rename Attributes

Several source attributes can be written to the output as they are, but do need renaming first.

In the AttributeManager rename the following:

- PSTLCITY to City
- PSTLPROV to Province
- POSTALCODE to PostalCode
- COUNTRY to Country

PSTLADDRESS	PSTLADDRESS	Do Nothing
PSTLCITY	City	Rename
PSTLPROV	PSTLPROV	Do Nothing
INTSTATE	fme_text_string	Remove
INTPSTLCD	fme_type	Remove
COUNTRY	Number	Remove
REPRESENT	PostalCode	Rename
STATUS	Provider	Remove
LASTUPDATE	Province	Remove
LASTEDITOR	Street	Remove
	UpdateDate	Remove
	xlsx_col_id	Remove
	xlsx_raster_height	Remove

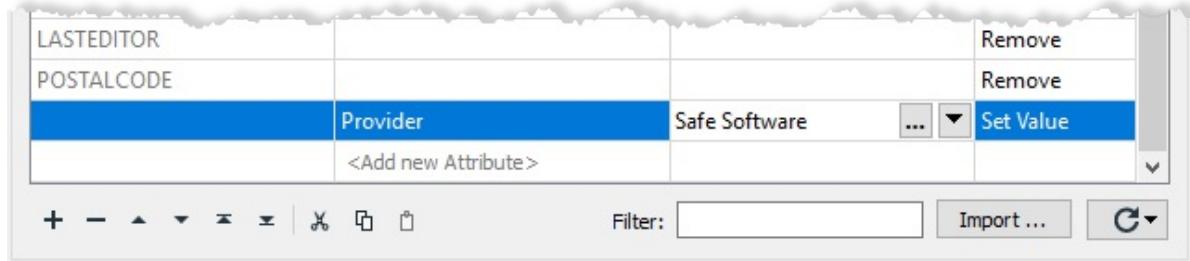
If the AttributeManager is connected to the writer feature type, then you should be able to select the Output Attribute field from a drop-down list instead of typing it in.

## 7) Create an Attribute (Provider)

Two attributes on the output (Provider and UpdateDate) are new and cannot be copied from the source data. They must be created.

In the AttributeManager create the new attribute "Provider." Because the attribute exists on the output schema, you can again select it from the drop-down list.

Set a fixed value such as your own organization name, "Safe Software," or "City of Interopolis."

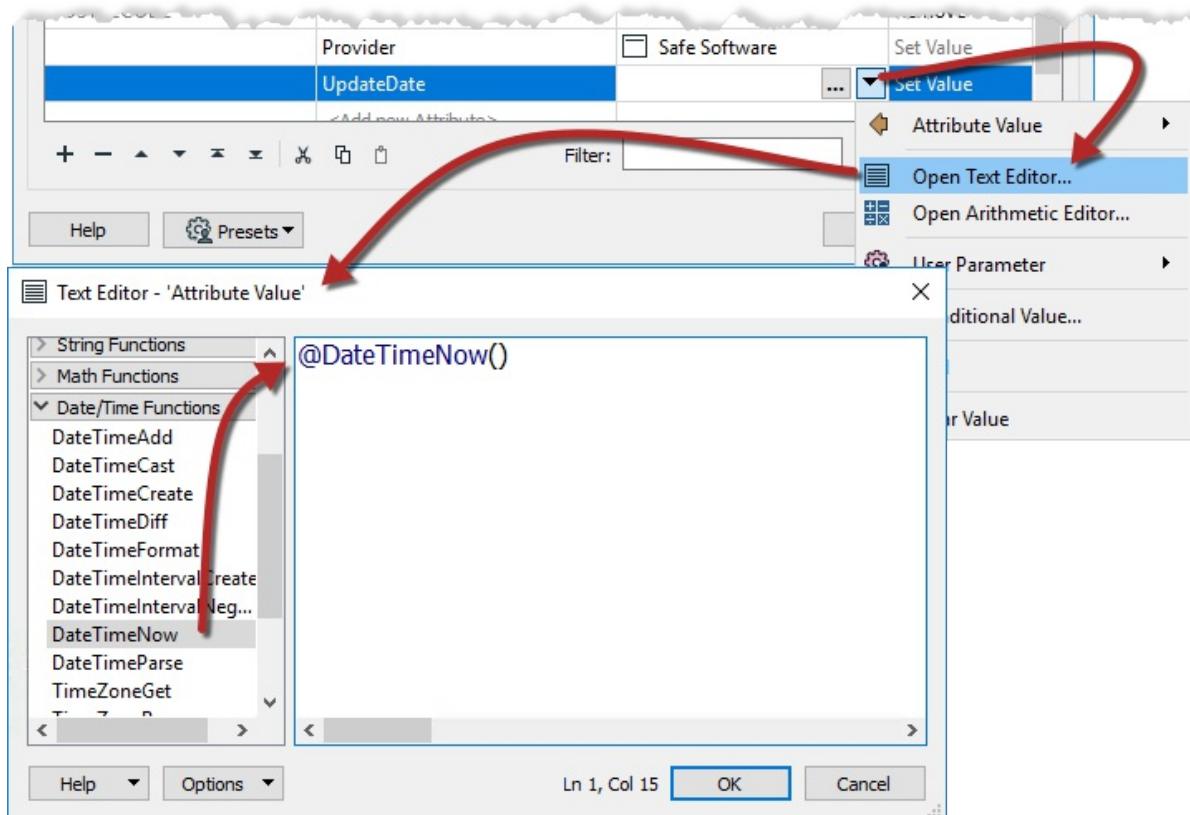


### 8) Create an Attribute (UpdateDate)

Now create the new attribute "UpdateDate". Rather than hard-coding a value, click on the drop-down arrow in the Attribute Value field and choose Open Text Editor.

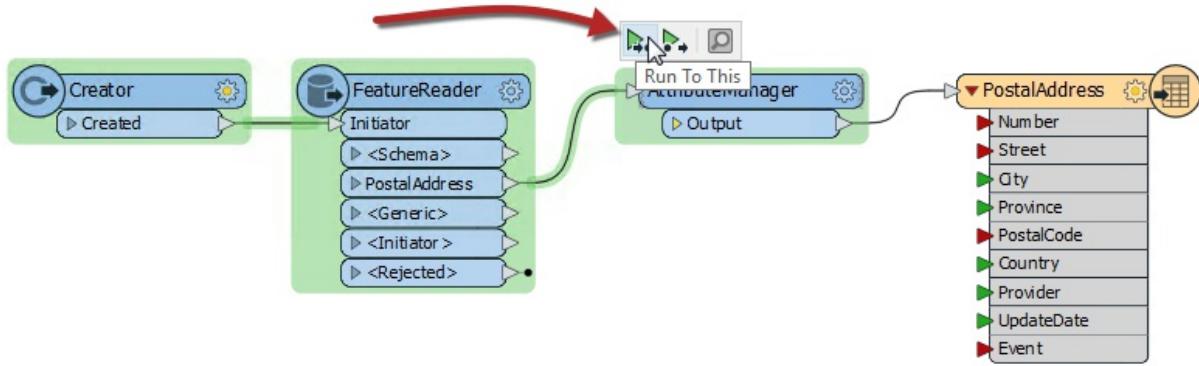
In the text editor locate the Date/Time Function called DateTimeNow and double-click to place it in the editor. By default it creates a datetime in ISO syntax, which is fine for us, so click OK to accept this.

Click OK again to close the AttributeManager dialog.



### 9) Run To the AttributeManager

Ensure feature caching is turned on, then run the workspace by clicking on the AttributeManager transformer and selecting Run to This:



This will run the translation up to this transformer, but not writing any output (which we don't need yet).

Inspect the AttributeManager:Output cache to confirm that the procedure worked as expected:

AttributeManager_Output							Columns...
	PSTLADDRESS	City	Province	Country	Provider	UpdateDate	
1	1188 W Pender St	Vancouver	British Columbia	Canada	Safe Software	20190326094001.9421382	
2	1661 Ontario St	Vancouver	British Columbia	Canada	Safe Software	20190326094001.9421382	
3	535 Smithe St	Vancouver	British Columbia	Canada	Safe Software	20190326094001.9421382	
4	181 W 1st Av	Vancouver	British Columbia	Canada	Safe Software	20190326094001.9421382	

## 10) Add an AttributeSplitter Transformer

Looking at the output schema there are two fields for Number and Street (for example "3305" and "W 10th Av"). However, the source schema condenses that information into one field with <space> characters separating the fields ("3305 W 10th Av"). Therefore we'll have to split the source attribute up to match the Writer schema.

Insert an AttributeSplitter transformer. Insert it **before** the AttributeManager - then if there are any actions to carry out on the split attributes we can use the same AttributeManager transformer.

View the AttributeSplitter parameters. Set PSTLADDRESS as the attribute to split and enter a space character into the Delimiter parameter. Notice that a list name is set in the List Name parameter (we'll use that list shortly):

Parameters

Attribute to Split:	PSTLADDRESS
Delimiter or Format String:	
Trim Whitespace:	Both
List Name:	_list
Drop Empty Parts:	No

Click OK to close the dialog. If you run the workspace now and inspect the cache, you'll see the address as a list attribute in the Feature Information window:

```
_list{0} (encoded: utf-8) 3305
_list{1} (encoded: utf-8) W
_list{2} (encoded: utf-8) 10th
_list{3} (encoded: utf-8) Av
```

Remember a list attribute is one that can store multiple values under a single name (\_list).

## 11) Rename Attribute

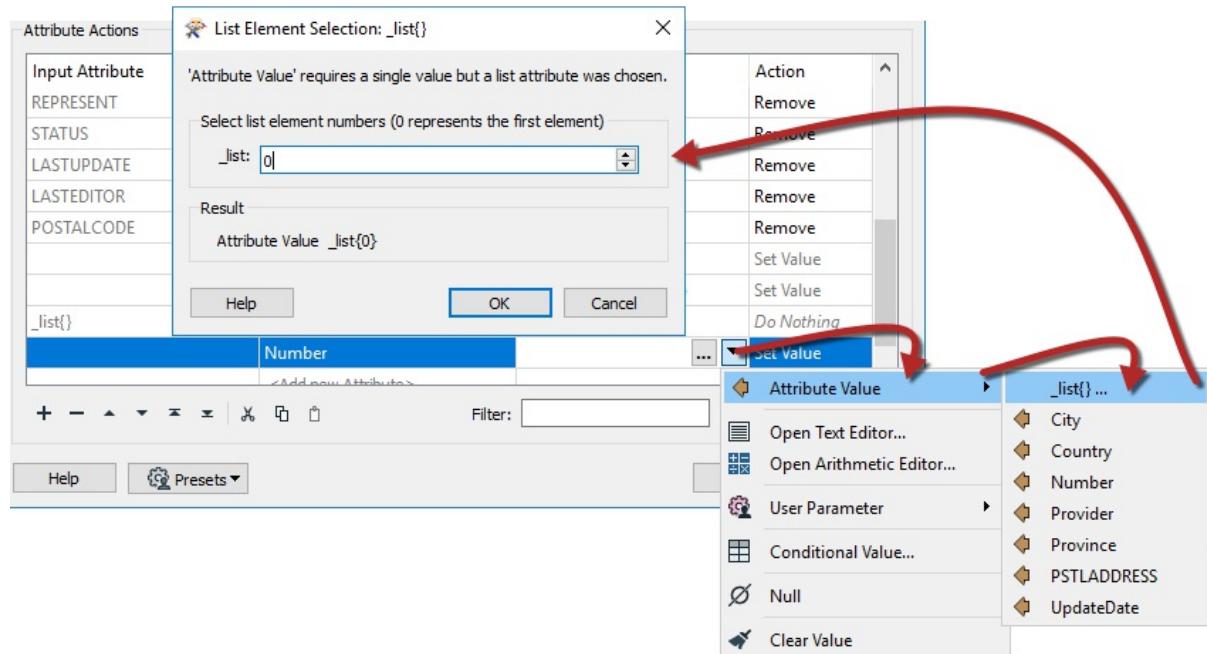
Now let's handle the Number field in the output. Go back to the AttributeManager parameters.

Notice that there is now an entry for the list attribute called `_list{}`. However, this is just the list attribute "in general" - it isn't showing each element (value) in the list.

What we need to do is create a new attribute and copy the list element we want into it. So, in the Output Attribute field create a new attribute called Number by selecting it from the drop-down list.

For the Attribute Value field click the drop-down arrow and select Attribute Value > `_list{}`.

You will now be prompted to select the element in the list. Ensure it is set to zero (0) and click OK.



Click Apply/OK to confirm the changes. Run the workspace and inspect the AttributeManager:Output cache to ensure the number is being copied.

## 12) Construct Attribute

The final step is to recreate the Street attribute, without it being prefixed by the address number.

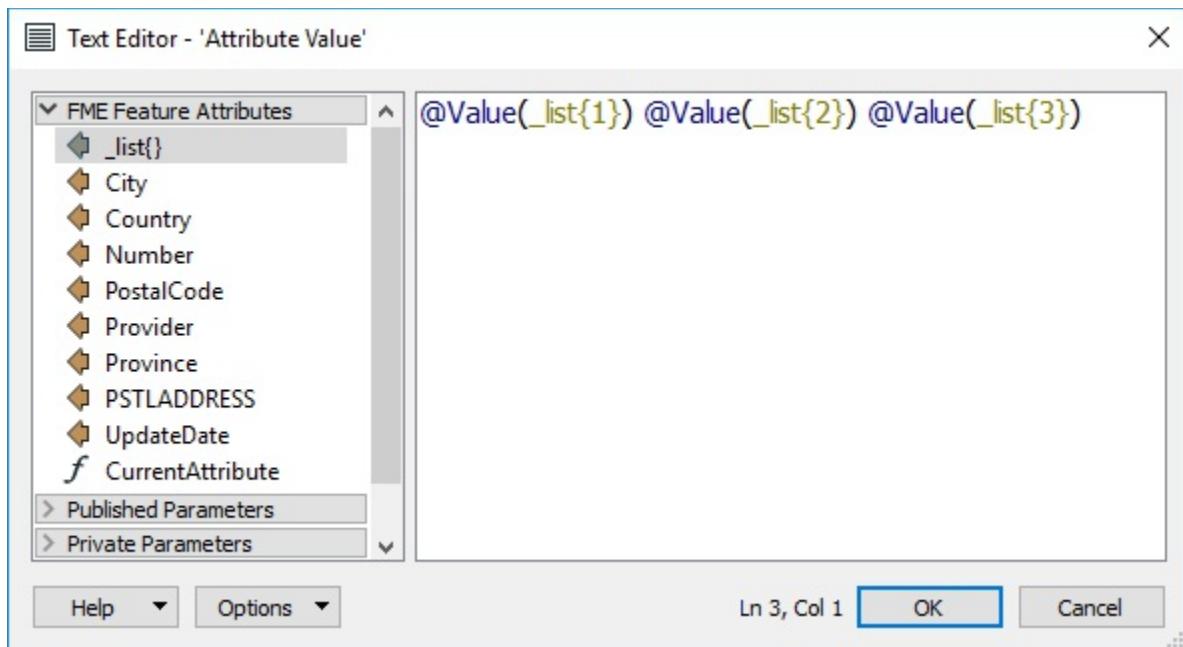
View the AttributeManager parameters again. This time in the Output Attribute field create a new attribute called Street by selecting it from the list.

Unlike the Number field, we want to create this value by concatenating several list elements. So click the drop-down arrow in the Attribute Value field and choose Open Text Editor.

Locate `_list{}` in the FME Feature Attributes menu and carry out the following steps:

- Double-click `_list{}` and when prompted select element 1. Click OK
- Press the spacebar to enter a <space> character
- Double-click `_list{}` and when prompted select element 2. Click OK.
- Press the spacebar to enter a <space> character
- Double-click `_list{}` and when prompted select element 3. Click OK.

The dialog will now look like this:



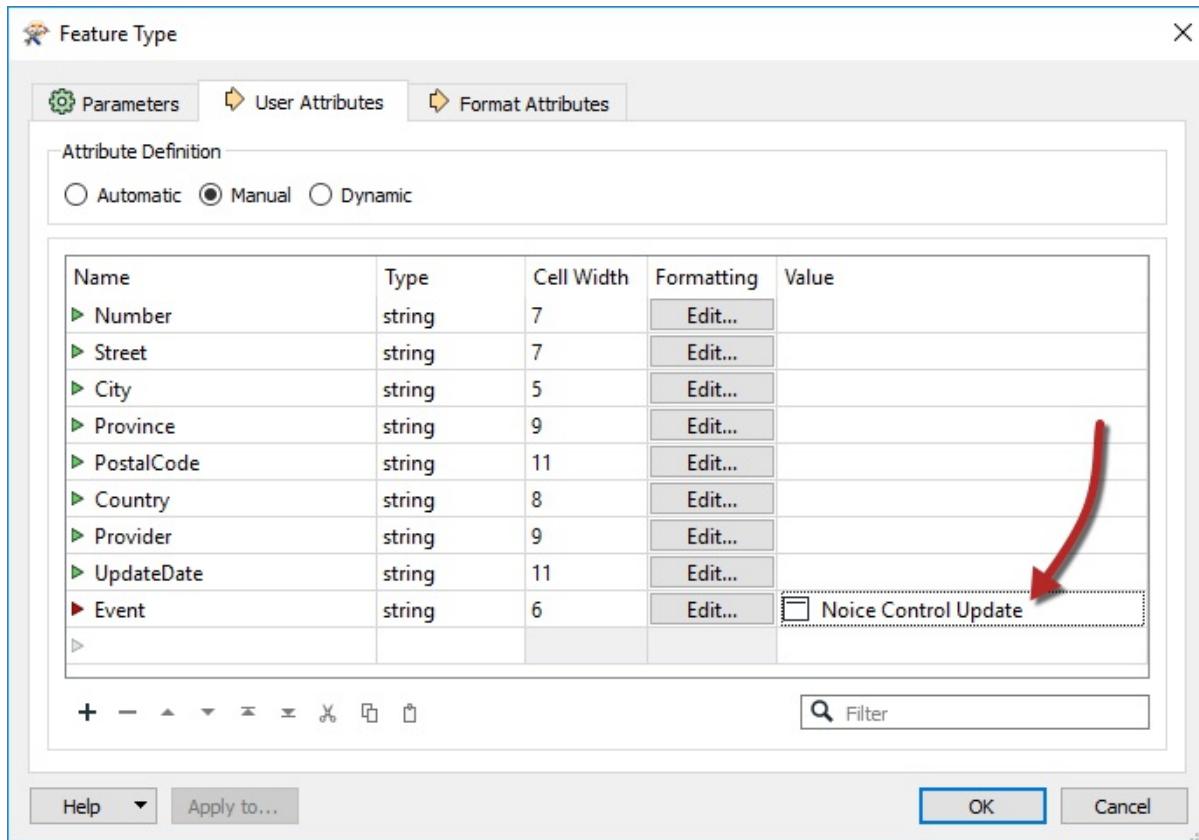
In this way we will have concatenated all parts of the street name back together, for example:

"W"+"17th"+"St" becomes "W 17th St"

We're assuming that no street name has more than three parts to it, but that's reasonable for our example.

### 13) Set the Event Field

As a final step, open the Feature Type dialog for the Excel writer, click on the User Attributes tab, and set a fixed value for the Event field:

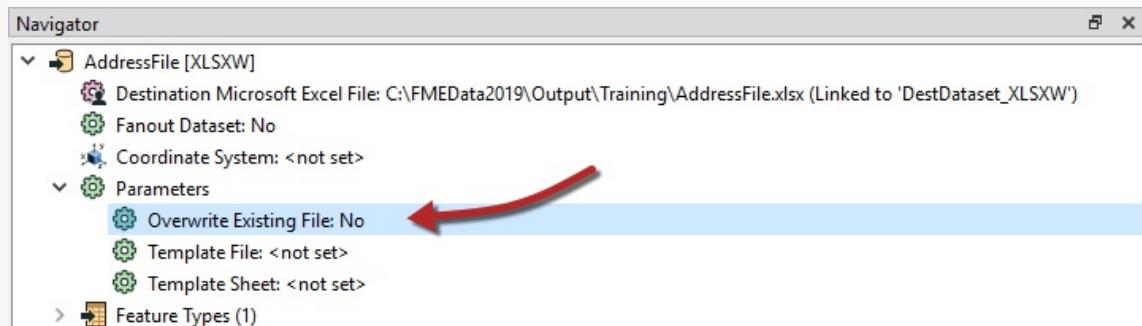


#### 14) Run the Workspace

Save the workspace and then run it.

## WARNING

The Excel writer has a parameter called **Overwrite Existing File**, which by default is set to **No**.



This is probably a good time to change the parameter to Yes, if you have been running the entire workspace each time.

Click on the PostalAddress writer feature type to open the popup menu then click on the View Written Data button to inspect the data in Visual Preview. The output (in the Table View) should look like this:

	Number	Street	City	Province	PostalCode	Country	Provider	UpdateDate	Event
1	1188	W Pender St	Vancouver	British Columbia	V6E4V5	Canada	Safe Software	20190326111501...	<missing>
2	1661	Ontario St	Vancouver	British Columbia	V5Y2Q7	Canada	Safe Software	20190326111501...	<missing>
3	535	Smithe St	Vancouver	British Columbia	V6B8E1	Canada	Safe Software	20190326111501...	<missing>
4	181	W 1st Av	Vancouver	British Columbia	V5Y4W5	Canada	Safe Software	20190326111501...	<missing>
5	141	W 1st Av	Vancouver	British Columbia	V5Y0W9	Canada	Safe Software	20190326111501...	<missing>
6	808	Gore Av	Vancouver	British Columbia	V6A2T7	Canada	Safe Software	20190326111501...	<missing>
7	26	E 15th Ave	Vancouver	British Columbia	V5T3SF	Canada	Safe Software	20190326111501...	<missing>

## CONGRATULATIONS

By completing this exercise you have learned how to:

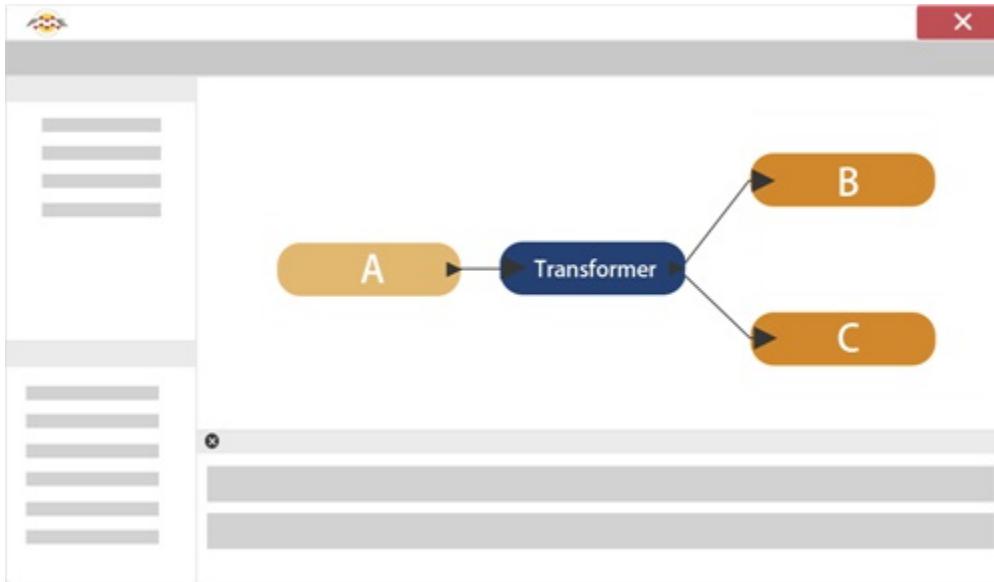
- Use the AttributeManager transformer to create, delete, and rename attributes
- Use the AttributeSplitter to split attributes into a list attribute
- Handle list attributes in the AttributeManager
- Use a Date/Time function in the AttributeManager text editor
- Use a feature type dialog to set an attribute value

## Conditional Filtering

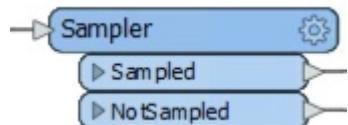
Transformers that filter, don't transform data content, yet surveys show that they're the most commonly used type of transformer there is!

### What is Filtering?

Filtering is the technique of subdividing data as it flows through a workspace. It's the case where there are multiple output connections from a transformer, each of which carries data to be processed differently. Here (for example) incoming stream A is filtered into two new streams, B and C:



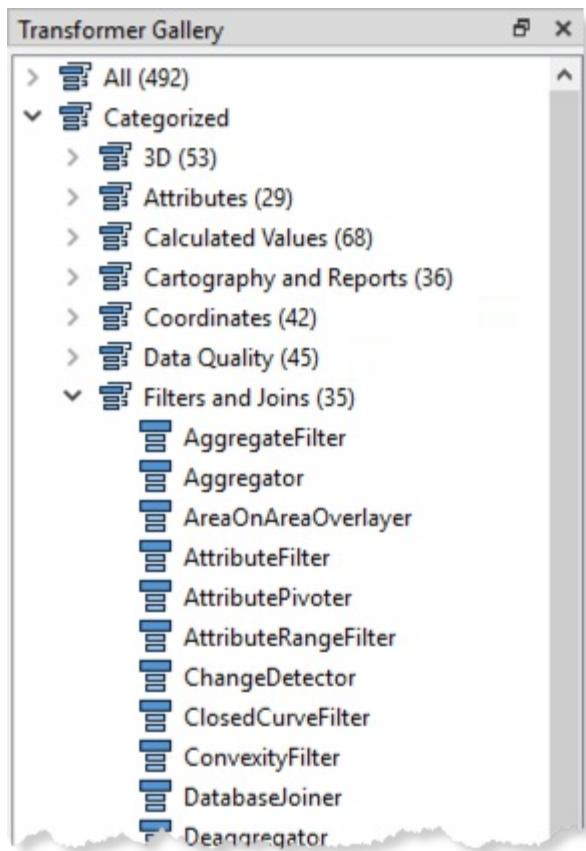
A filtering transformer may be any transformer with multiple output ports, such as the GeometryFilter or Sampler transformers, the latter of which creates a sample selection of data and separates it out through Sampled and NotSampled output ports:



However, these are mostly in-built, fixed tests. Conditional filtering is where the decision about which features are output to which connection is decided by some form of *user-defined* test or condition. The Tester transformer is the most obvious example of this. It carries out a test and has different output ports for features that pass and fail the test.

### Transformers that Filter

Many transformers in the Filters and Joins category carry out these tests and redirect data according to the results:



Although the Tester transformer is the most used of this category, there are many other transformers such as the TestFilter, GeometryFilter, AttributeFilter, SpatialFilter, and Sampler.

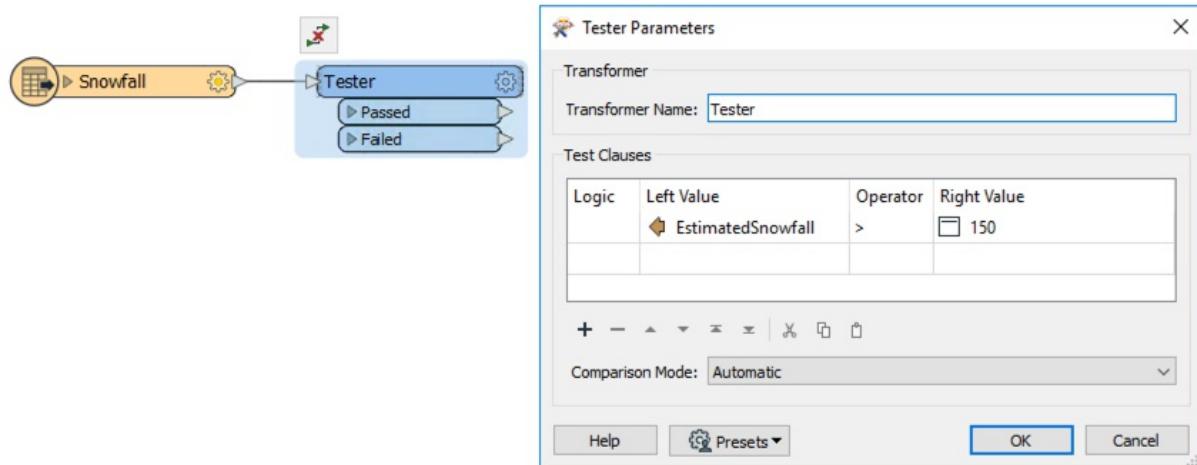
## The Tester and TestFilter Transformers

The Tester and TestFilter are the two key transformers for conditional filtering. They test the values on attribute values.

### Tester

The Tester transformer is generally for single tests that produce a Yes/No result.

For example, here we wish to decide whether to send out snow plows to a particular road based on whether the value of the Snowfall attribute is greater than 150 mm (approximately 6 inches):



If snowfall is greater than 150, the road feature will pass the test and snow plows will be sent.

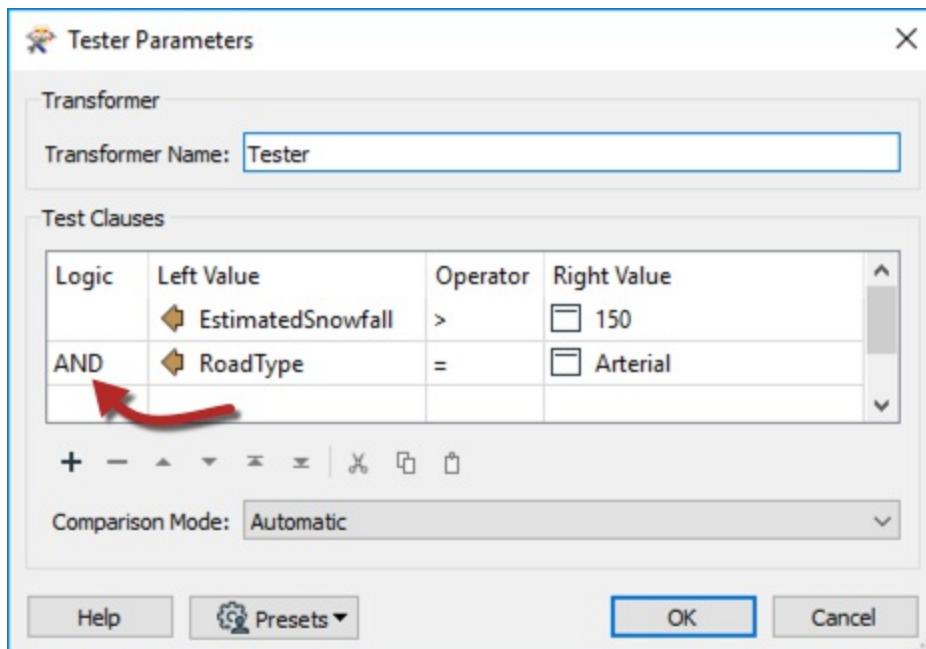
### Multiple Clauses

Each clause in the Tester is an individual test that allows a Passed/Failed result. For example, each of the following criteria might be separate tests:

- Has there been more than 100 mm (4 inches) of snowfall?
- Is this a major road?
- Is the temperature less than zero degrees Celsius?
- Was sand last applied more than 24 hours ago?

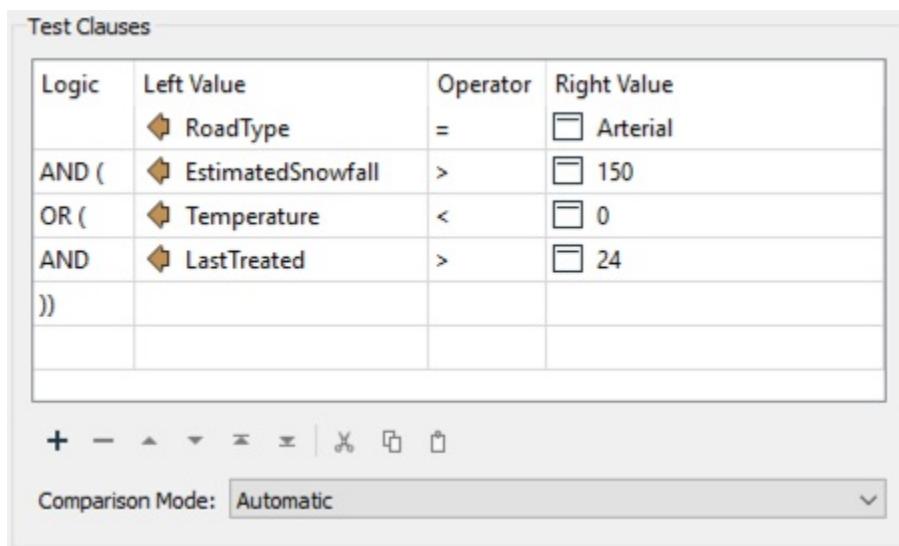
However, the Tester allows the combination of multiple tests, where a user can combine any number of clauses using an AND and OR statement. So instead of individual tests, I might ask:

- Is this an Arterial (major) road AND has there been more than 100 mm of snow?



The Tester also allows the mixing of AND and OR statements using what is called a Composite Test. For example:

- Is this an Arterial road AND (has there been more than 100 mm of snow OR (is the temperature less than zero AND was the road treated (with sand) more than 24 hours ago))?

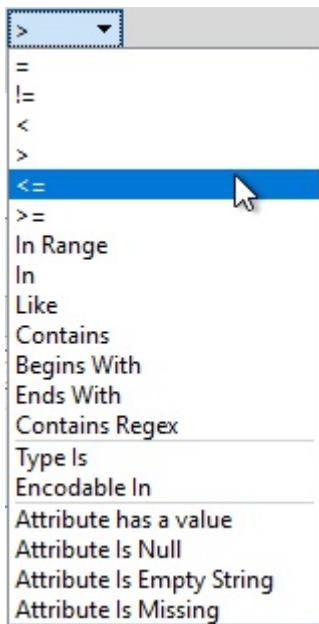


But - however complex the test becomes - it still results in a single Yes/No (binary) result; features will either pass or fail this set of tests.

It's also worth remembering that we aren't restricted to simple tests of equality ( $A=B$ ); in the above, there are also "greater than" and "less than" tests. That's because there are many different operators available for use in a test clause.

## Operators

The list of operators available in the Tester transformer (or in many of the other locations that make use of the Tester dialog) looks like this:



Besides the usual operators, there are also some based on a SQL where clause. These include:

- In
- Like
- Contains
- Begins With
- Ends With
- Contains Regex

...plus there are other tests that check for the existence of attributes and values:

- Attribute has a value
- Attribute is Null
- Attribute is Empty String
- Attribute is Missing

### TIP

*"Attribute has a value" is the opposite of the three other tests; for example, this attribute is not Null, AND it is not an empty string, AND it is not missing. Incidentally, "missing" means the attribute does not exist at all on the feature being tested.*

### TIP

*"Contains Regex" means only part of the string needs to match. For example...*

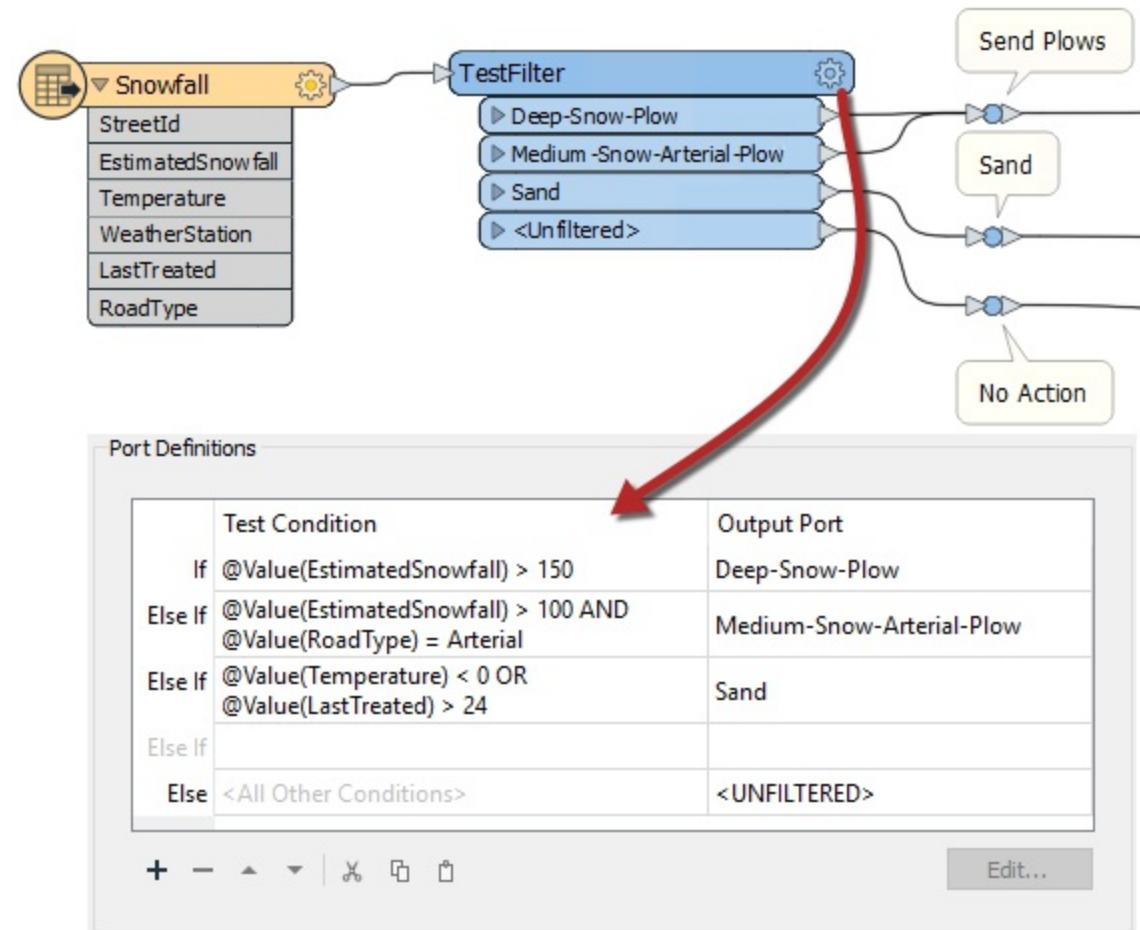
*Attribute Value: abcd  
Search String: ^ab  
Contains Regex: Passed*

*For example, the entire string doesn't need to match.*

## TestFilter

The TestFilter allows a number of conditions to be tested, each of which can have a number of test clauses. Each condition is given an output port, with an additional output port for features that fail all of the test conditions.

The TestFilter is very similar to the CASE or SWITCH command in programming or scripting languages. In Workbench it looks like this:



Notice that there are multiple conditions, and an output port for each. Each condition/port combination is equivalent to a single Tester transformer; hence the TestFilter is a good way to combine multiple Tester transformers into one.

The TestFilter output ports can be given custom names, rather than a simple PASSED/FAILED, which is another advantage to this transformer over the Tester.

The TestFilter has the full set of operators available with the Tester such as equals, greater than, less than, and so forth. Each condition is tested in turn.

Features that pass are output through the matching output port. Features that fail are sent on to the next condition in the list. Therefore it's very important to get the conditions in the correct order.

### FME Lizard says...

*The TestFilter is very good for filtering a feature by a set of cascading conditions, for example here are a set of tests to again determine whether to send out a snow plow:*

- Has there been more than 150 mm of snowfall?
- Has there been more than 100 mm of snowfall AND is this an Arterial road?
- Is the temperature less than zero degrees Celsius AND was sand last applied more than 24 hours

*ago?*

*It's a set of cascading tests because if there has been more than 150 mm of snow, the plows are sent out anyway; you don't need to test any other criteria. So the test order can be very important. If every test is a fail, then the plows are not sent out.*

---

If using the above snowfall example, you were using three Tester transformers, you could save space on the workspace canvas and replace that setup with just a single TestFilter.

---

### FME Lizard says...

*Because the TestFilter can carry out a single test (as well as multiple ones) it's possible to use it exclusively instead of the Tester transformer.*

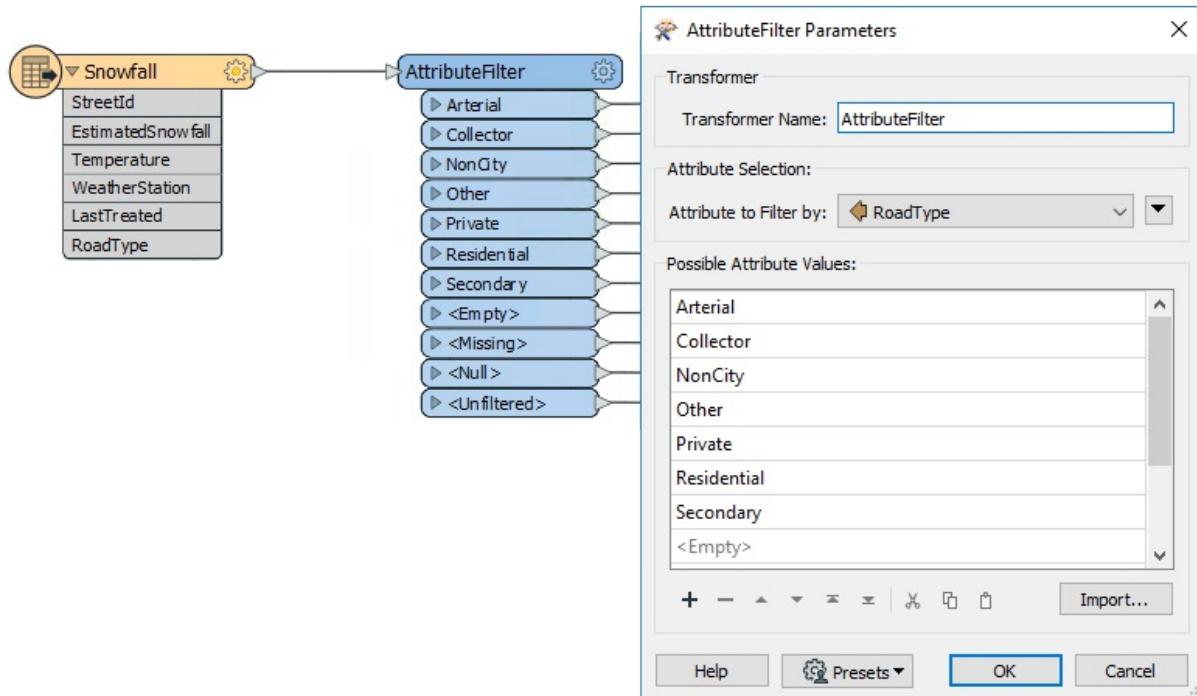
## Other Key Filter Transformers

The Tester and TestFilter are not the only useful filter transformers.

### AttributeFilter

The AttributeFilter transformer directs features by values in a chosen attribute. It is not a binary test (Yes/No) but a way to separate many values for a single attribute, for example:

- Is that road an Arterial, Collector, NonCity, Other, Private, Residential, or Secondary?



If you were using seven Tester transformers to separate this data, you could save space on the workspace canvas and replace that setup with just a single AttributeFilter.

### TIP

*Use the Import... button to quickly add attributes from existing datasets.*

### FME Lizard says...

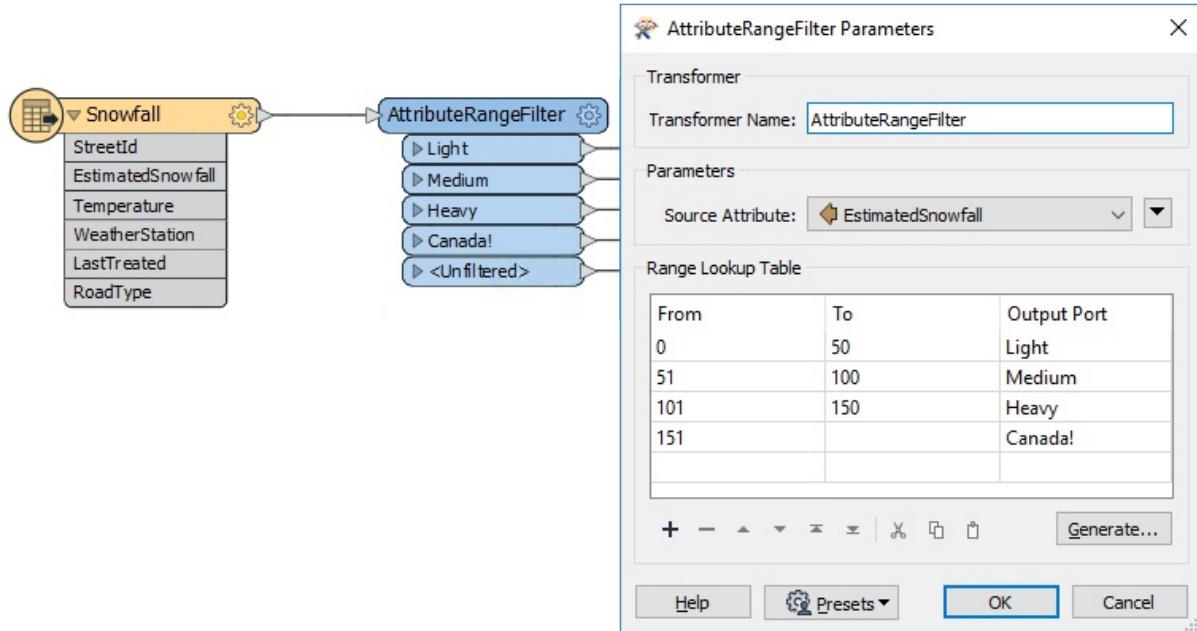
*In almost every scenario where you are using multiple Tester transformers, it's possible to use a different filtering transformer to achieve the same result but using much less space on the canvas.*

The AttributeFilter also works with numeric values; however, its only "operator" is to find equivalency (=), so you would rarely use it for arithmetical tests. In that scenario, the better solution is the AttributeRangeFilter.

### AttributeRangeFilter

The AttributeRangeFilter carries out the same operation as the AttributeFilter, except that it can handle a range of numeric values instead of just a simple one-to-one match.

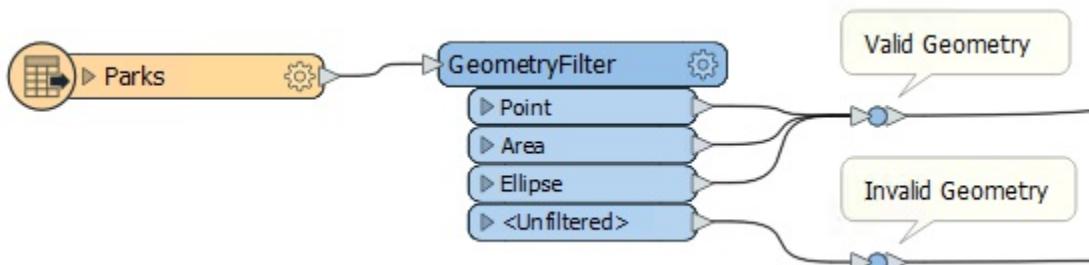
For example, we might want to separate data based on a range of snowfall values, like so:



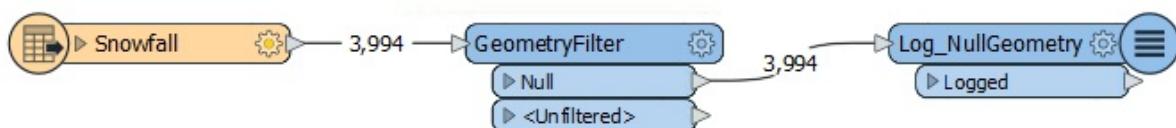
Notice that the AttributeRangeFilter parameters dialog has a Generate button to generate ranges automatically from a set of user-defined extents.

## GeometryFilter

The GeometryFilter directs features on the basis of geometry type; for example, point, line, area, ellipse:



It can even filter data based on null geometry (tabular records):



The GeometryFilter is useful for:

- Filtering out unwanted geometry types; for example, removing non-linear features before using an AreaBuilder transformer
- Validating geometry against a list of permitted types; for example, where the dataset is constrained to either point or area features (above)
- Dividing up geometry types to write to separate destination Feature Types; for example, when writing to a geometry-restricted format such as Esri Shapefile

## FME Lizard says...

*If the Tester, TestFilter, and AttributeFilter all filter features on the basis of an attribute condition, then what's the difference? When would I use each?*

*The best solution is to check out these two articles on the Safe Software blog:*

- [Conditional Processing in FME](#)
- [Test Clauses Remade: Data Filtering in FME 2019](#)

*There's also a useful table I put together:*

	Single Test		Multiple Tests		Test Type		Operators	Attributes
	Single Clause	Multi Clause	Single Clause	Multi Clause	String	Numeric		
Tester	Y	Y	-	-	Y	Y	16	Multiple
TestFilter	Y	Y	Y	Y	Y	Y	16	Multiple
AttributeFilter	Y	Y	-	-	Y	-	1	1
AttributeRangeFilter	Y	Y	-	-	-	Y	6	1

### Exercise 3

### Noise Control Laws Project (Spatial Filtering)

<b>Data</b>	Addresses (File Geodatabase) Zoning (MapInfo TAB) Roads (AutoCAD DWG)
<b>Overall Goal</b>	To find all residential addresses within 50 meters of an arterial highway
<b>Demonstrates</b>	Methods of conditional filtering
<b>Start Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\Transformers-Ex3-Begin.fmw
<b>End Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\Transformers-Ex3-Complete.fmw

As you know, city councilors have voted to amend noise control laws and residents living in affected areas must be informed of these changes.

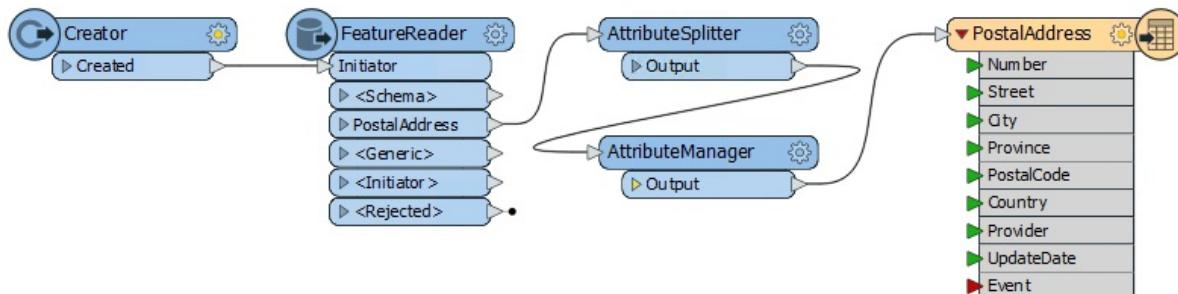
You have been recommended by your manager to take on the task, and there's a tight deadline.

In the first part of the project, you created a workspace to convert addresses from Geodatabase to Excel, mapping the schema at the same time.

This exercise is the second part of the project: locating all affected residents. You must locate all single-family residences within 50 meters of a major highway and filter out all others from the stream of address data.

#### 1) Start FME Workbench

Start FME Workbench (if necessary) and open the workspace from Exercise 2. Alternatively you can open C:\FMEData2019\Workspaces\DesktopBasic\Transformers-Ex3-Begin.fmw



The workspace already has a FeatureReader to read addresses, transformers to edit the address schema, and a writer to write data to an Excel spreadsheet.

#### 2) Add Roads Data Reader

Use Readers > Add Reader to add a reader for the roads data. The roads data will be used to determine distance from an arterial route.

<b>Reader Format</b>	Autodesk AutoCAD DWG/DXF
<b>Reader Dataset</b>	C:\FMEData2019\Data\Transportation\CompleteRoads.dwg

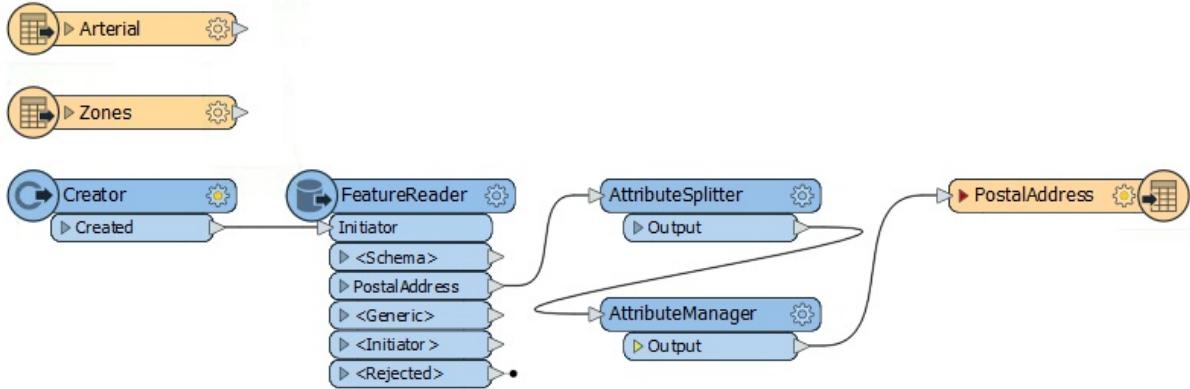
When prompted, select only the feature type called Arterial.

#### 3) Add Zoning Data Reader

Use Readers > Add Reader to add a reader for zoning data. The zoning data will be used to determine whether an address is single-family residential or not.

<b>Reader Format</b>	MapInfo TAB (MITAB)
<b>Reader Dataset</b>	C:\FMEData2019\Data\Zoning\Zones.tab

With attribute lists collapsed, the workspace will now look like this:



Feel free to inspect all of the source data to familiarize yourself with the contents. You can even run the workspace to make sure all caches are up to date.

#### 4) Add a Tester Transformer

Add a Tester transformer to the Zoning feature type.

This Tester will be used to filter residential zones from the other zoning areas. All single-family residential zones will start with RS, so the Tester should be set up like this:

Logic	Left Value	Operator	Right Value
ZoneName		Begins With	<input type="checkbox"/> RS

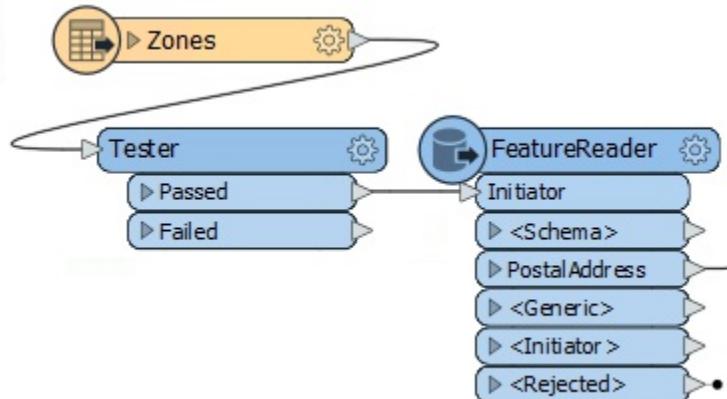
Comparison Mode: Case Insensitive

The important thing is to set up the test with the “Begins With” operator.

#### 5) Connect Tester to the FeatureReader

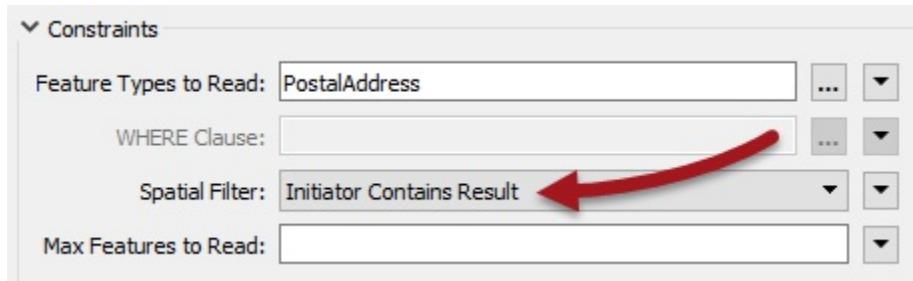
One way to filter data is to use a SpatialFilter transformer, and we will do this with the road features. But another method is to use filtering inside the FeatureReader transformer.

So, delete the Creator transformer and connect the Tester:Passed port to the FeatureReader:Initiator port:



#### 6) Set up the FeatureReader

Now inspect the FeatureReader's parameters. Set the Spatial Filter parameter to *Initiator Contains Result*:



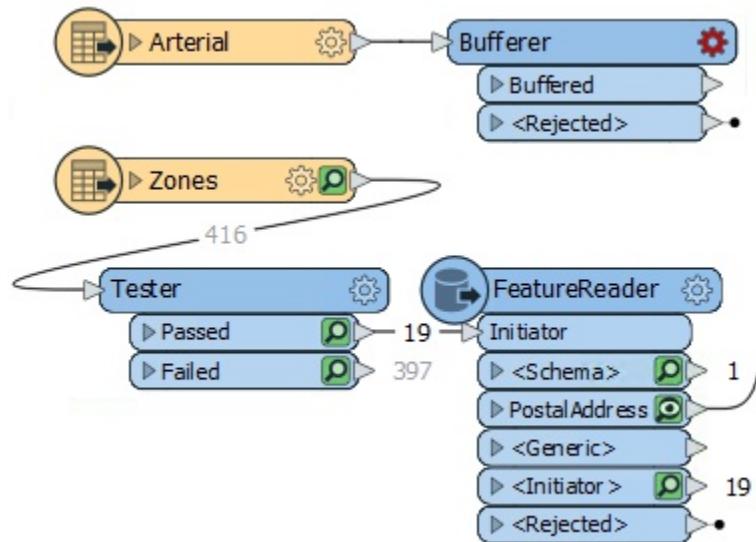
This ensures that only addresses that fall inside a Single Family Residence zone will be read from the database. Make sure feature caching is turned on and run the workspace to the FeatureReader. Inspect the Tester:Passed and FeatureReader:PostalAddress caches to confirm that the results are correct.

For the Tester, there should be 19 features coming from the Tester:Passed output port. The FeatureReader:PostalAddress output port will now only have 1853 features, compared to the 13597 features before we filtered by zone.

### 7) Add a Bufferer Transformer

Now we can determine which of the filtered addresses fall within 50 meters of an arterial route. First, we will need to add a Bufferer transformer to set the 50-meter distance around the roads.

Add a Bufferer transformer to the workspace. Connect it to the Arterial roads data:



Set the Bufferer Buffer Amount parameter to be 50.

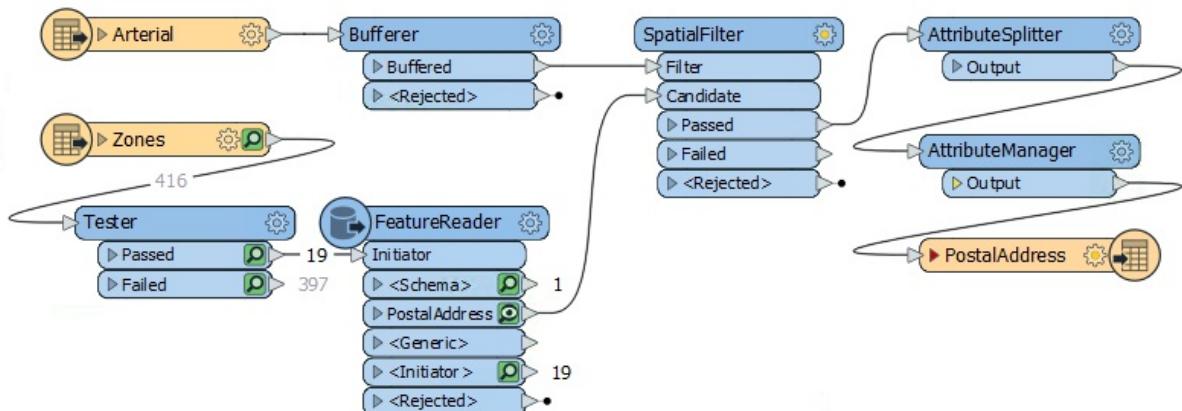
#### TIP

*Optionally you can set the Bufferer to dissolve the polygons to merge all the buffered features together based on an attribute like the fme\_feature\_type.*

*The results of the translation will be the same (in terms of addresses selected) but the data will look better in Visual Preview.*

### 8) Add a SpatialFilter Transformer

Add a SpatialFilter transformer. The buffered arterial routes are the Filter. The Candidate port can be connected between the FeatureReader and the AttributeSplitter:



This way the AttributeSplitter and AttributeManager are operating only on filtered features. If the SpatialFilter was connected after the AttributeManager, then data would be getting processed and then discarded.

## 9) Set SpatialFilter Parameters

Set up the SpatialFilter parameters as follows:

<b>Filter Type</b>	Multiple Filters	There are multiple buffer polygons
<b>Pass Criteria</b>	Pass Against One Filter	A single address cannot be in <b>all</b> buffers
<b>Spatial Predicates to Test</b>	Filter Contains Candidate	Find addresses contained in the arterial buffers

Tests

Filter Type:	Multiple Filters
Pass Criteria:	Pass Against One Filter
Support Mode:	Support Aggregates
Spatial Predicates to Test:	"Filter Contains Candidate"
Use Bounding Box:	No
Curve Boundary Rule:	Default Rule

That is, there are multiple road buffers, but an address only has to be inside one buffer to pass, not all of them.

## 10) Run the Workspace

Run the workspace by pressing F5 and check the output to confirm the dataset has been written correctly. There should be 148 records in the spreadsheet, ready to send to the administration department for a bulk mailing.

If you have more than 148 rows in the final spreadsheet, remember to set the Excel Writer parameters to Overwrite Existing File to Yes.

## CONGRATULATIONS

By completing this exercise you have learned how to:

- Use the Tester transformer to filter by an attribute value
- Use the Spatial Filter option in the FeatureReader transformer
- Use the Bufferer transformer to set up a "within x distance of" test
- Use the SpatialFilter transformer to filter by geometry

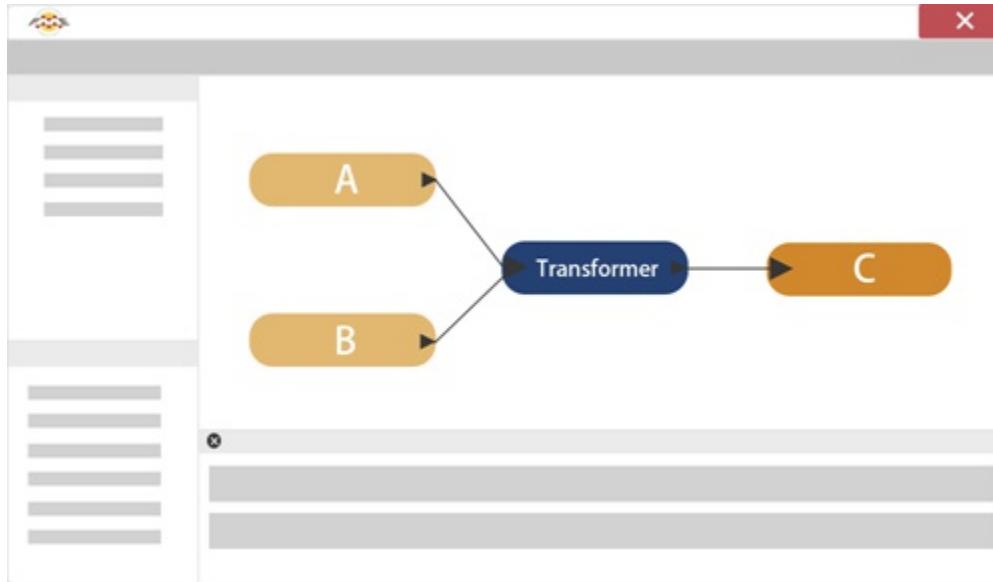


## Data Joins

Transformers that join data together are also very commonly used in FME. It's so related to filtering that the transformer category is called Filters and Joins.

### What is a Data Join?

Whereas Filter transformers divide data into different streams, other transformers bring data streams together, merging the data according to a set of user-defined conditions. Here (for example) incoming streams A and B are joined together into a new stream, C:



It is necessary to do more than just draw two connections into the same input port to merge data in FME Workbench. Connecting data streams this way will only combine the data into a single stream, not fuse it together. You may know this as a union of data.

---

To merge data it is necessary to define a relationship for the basis of the join, and this is done with one of a number of transformers.

These transformers allow you to merge not just data that is being processed by the workspace but provide the ability to form a join against a database or other external dataset.

Joins in FME can either be based on matching attribute values (DatabaseJoiner or FeatureMerger/FeatureJoiner), or they can be based on a spatial relationship such as an overlap between features or proximity from one feature to another (NeighborFinder or SpatialRelator).

## Key-Based Join Transformers

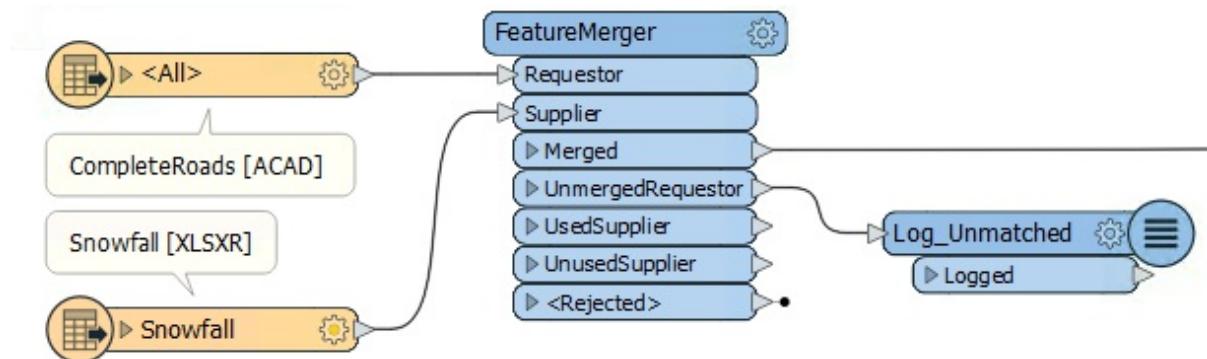
Several transformers can join data by matching attribute values (keys). Some of these are more oriented towards geometry, while others have a more SQL-like style. Some join streams of data within one workspace, while others join one stream of data to an external database.

Which you use depends on your join requirements and performance needs.

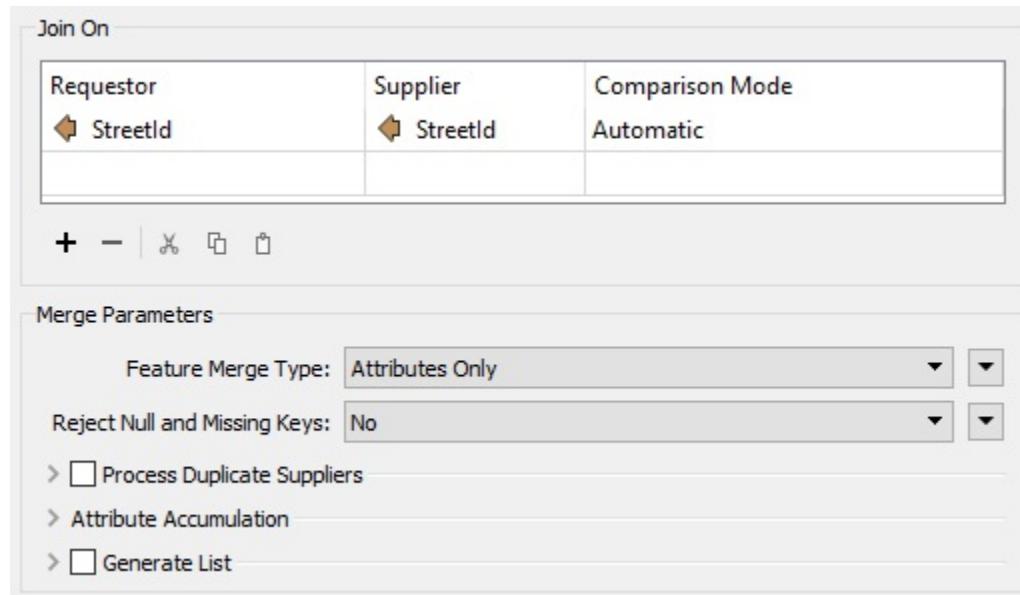
### FeatureMerger

The FeatureMerger is a transformer for joining two (or more) streams of data within a workspace based on a key field match.

Here, for example, a dataset of roads has a StreetId number. The FeatureMerger is being used to combine information from a spreadsheet of snowfall information onto the roads data:



The parameters dialog for the FeatureMerger looks like this:

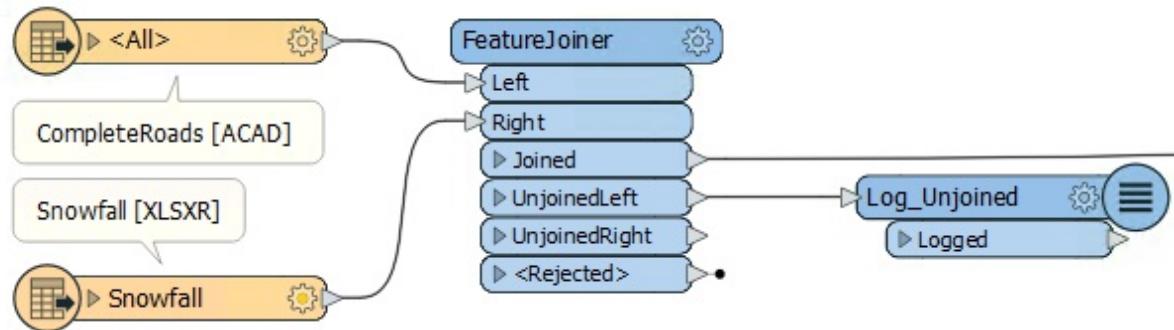


This screenshot shows the join is made using StreetId as a key. All Requestor (Road) features that have a matching snowfall record are output through the Merged output port. All Road features without a match are output through the UnmergedRequestor port for inspection to determine why a match did not occur.

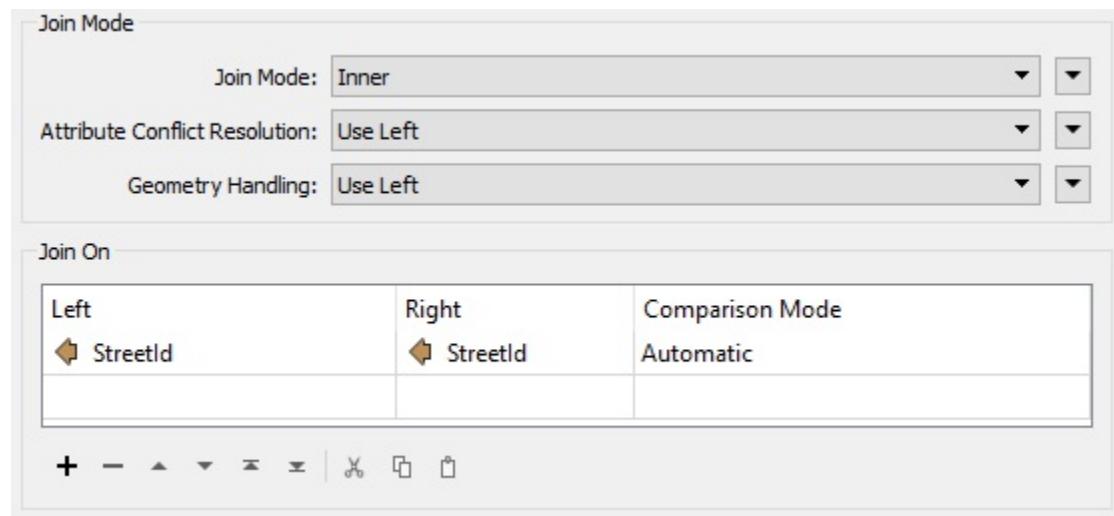
There are additional parameters to handle conflicts of information, duplicate keys, and whether to merge attributes only or geometry as well.

### FeatureJoiner

The FeatureJoiner is another transformer for joining two streams of data within a workspace based on a key field match.



Here, for example, is the same Roads/Snowfall match in the FeatureJoiner. The parameters for the transformer looks like this:



As you can see, this transformer is based more on traditional SQL queries. The Join Mode parameter can take one of three values:

Mode	Description	Depiction	Joined Output	Unjoined Left	Unjoined Right
Left	Left features look for a match and are output whether they find a match or not		All matches plus unmatched Left features	None	Unused Right features
Inner	Left features look for a match and are output if they find one		All matches only	Unmatched Left features	Unused Right features
Full	Both Left and Right features output through the Joined output port, whether they find a join or not		All matches plus unmatched Left and Right features	None	None

Other terms you might be familiar with are *Outer Join* and *Right Join*. An Outer join is simply a different name for what the Full Join does here. To do a Right join, you would switch which features are being sent to which input port and use the Left Join option.

## WARNING

*The key thing to be aware of here is that a feature is output for every match that occurs. For example, if 1 Road feature matches 5 Snowfall records, there will be 5 features output as Joined.*

*Joined features are always output as a Join. Left, Inner, and Full really only control which unmatched records are included in the Joined output.*

With a Left join the user either believes that all roads will have a matching snowfall record, or it does not matter if there is not a match. In fact, no features will ever appear from the Unjoined Left output port.

If it was essential to ensure a match, then the chosen mode should be Inner. Then records that exited the Unjoined Left output port could be treated as an error and investigated as to why there is no match.

Like the FeatureMerger, there are parameters to handle conflicts of information and whether to merge attributes only or geometry as well.

## TIP

*So the key difference between the FeatureMerger and the FeatureJoiner is what happens to multiple matches.*

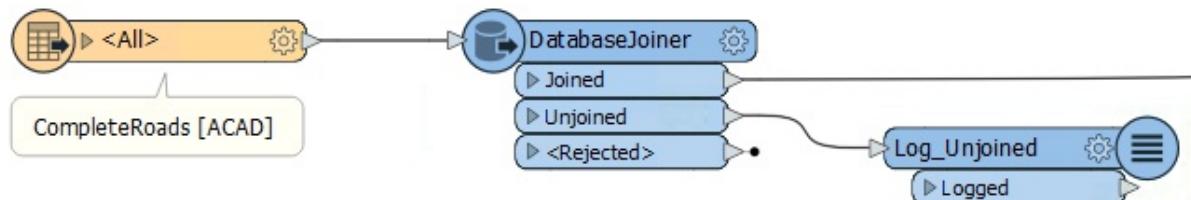
*If the FeatureJoiner has multiple matches, it will output multiple features.*

*If the FeatureMerger has multiple matches, it will output one feature only. That feature will either have only one matched record, or a list of matched records, depending on the Process Duplicate Suppliers parameter.*

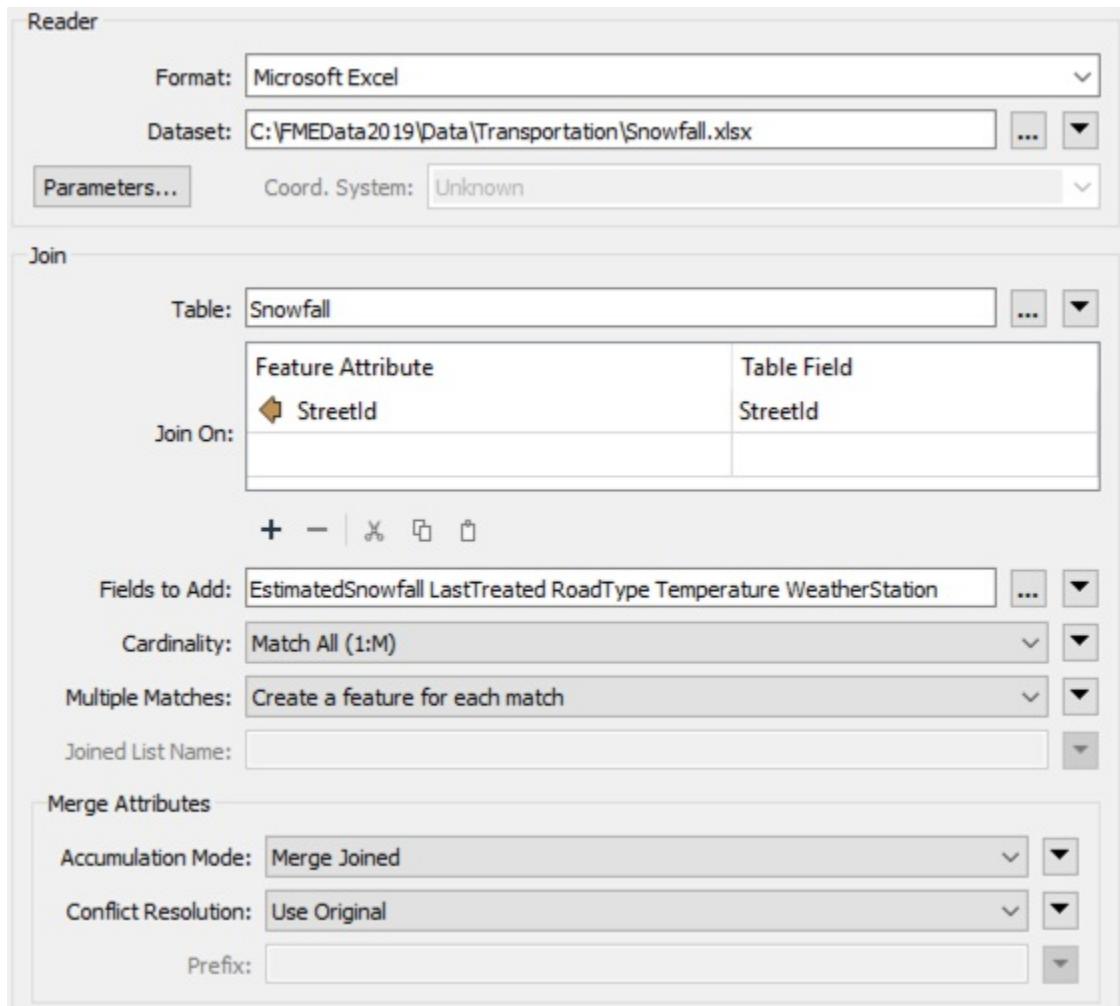
## DatabaseJoiner

The DatabaseJoiner transformer is different to the FeatureMerger and FeatureJoiner because, instead of merging two streams of features, it merges one (or more) stream(s) of data with records from an external database.

Here is the same example as for the FeatureMerger above. In this case, the roads features are obtaining snowfall data directly from a table in an Excel spreadsheet:



The parameters dialog for the DatabaseJoiner looks like this:



Again, StreetID is being used from both feature and database table to facilitate a merge between the two.

As with the other transformers, there are parameters to control the attributes that are accumulated and how conflicts are resolved.

### FME Lizard says...

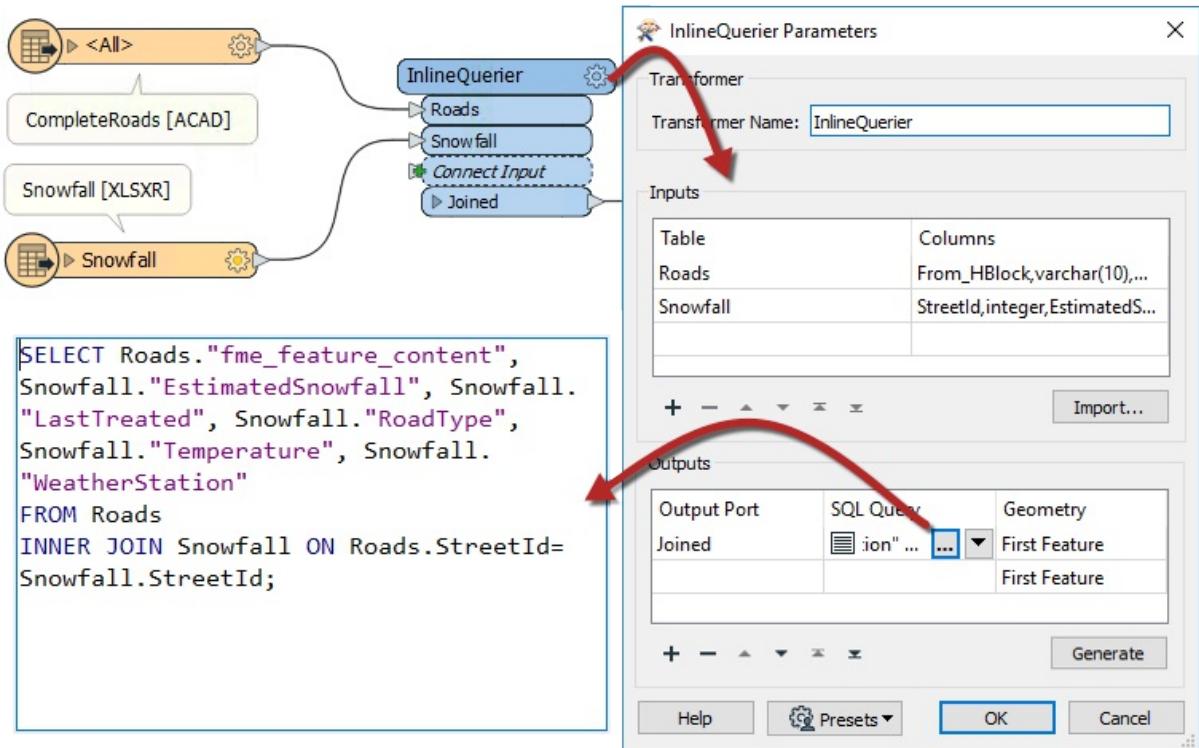
*The DatabaseJoiner has a number of advantages over the FeatureMerger. Firstly it has parameters to control how multiple matches are handled, as well as parameters for optimizing the database query.*

*Secondly, it allows features to be joined without having to read the entire dataset into a workspace. FME can just query the database and select the individual records it needs. This can improve performance greatly.*

*It does, of course, require the supplier records to be stored in an appropriate database format!*

### InlineQuerier

The InlineQuerier transformer accepts features from the workspace and generates a temporary database. With that database it's possible to apply any SQL commands required - including Joins - across a number of tables:



The InlineQuerier has the distinct advantage of allowing its input to be reused multiple times in a single transformer; whereas multiple joins would otherwise require multiple FeatureJoiner transformers. However, there is a performance overhead involved in generating that initial database.

### TIP

*With all of these transformer choices, it is hard to choose which transformer is the right one for the job. Thankfully there is a flowchart to help you decide, check out the [Merging or Joining Spreadsheet or Database Data](#) article.*

## Spatially Based Join Transformers

Multiple transformers can join data by spatial relationship. Which you use depends on the spatial relationship to be tested and your exact join requirements. The following are some of the key transformers.

### FME Lizard Says...

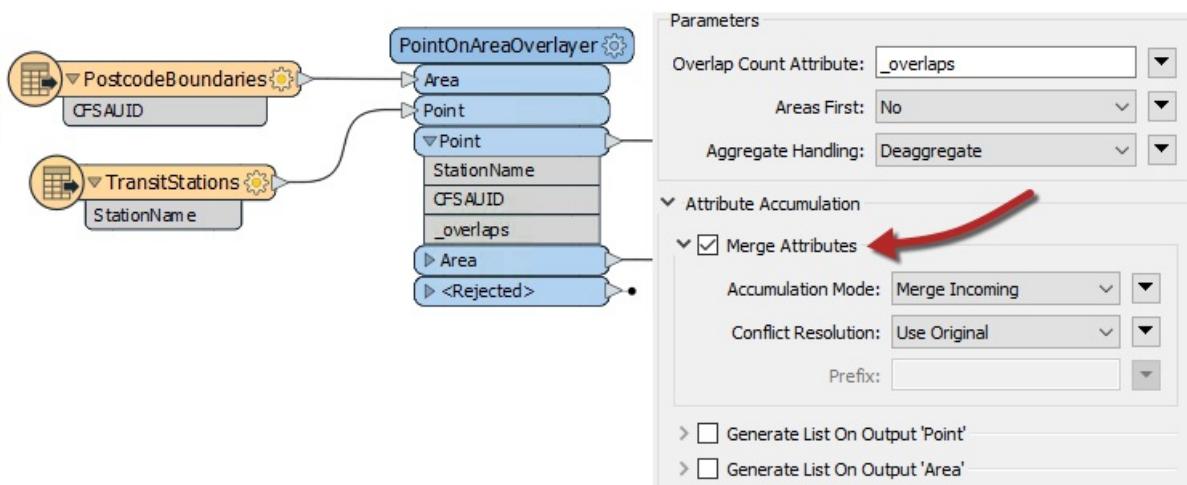
*Just like the key-based joining transformers, there is an flowchart to also help you choose a spatial join transformer. See the [Merging or Joining Spatial Data](#) article.*

## Overlays

There are a number of different "overlay" transformers, each handling a different form of overlay.

For example, the PointOnAreaOverlay transformer carries out a spatial join on points that fall inside area (polygon) features. This operation is sometimes called a "Point in Polygon" overlay.

As the help explains, "each point receives the attributes of the area(s) it is contained in, and each containing area receives the attributes of each point it contains."



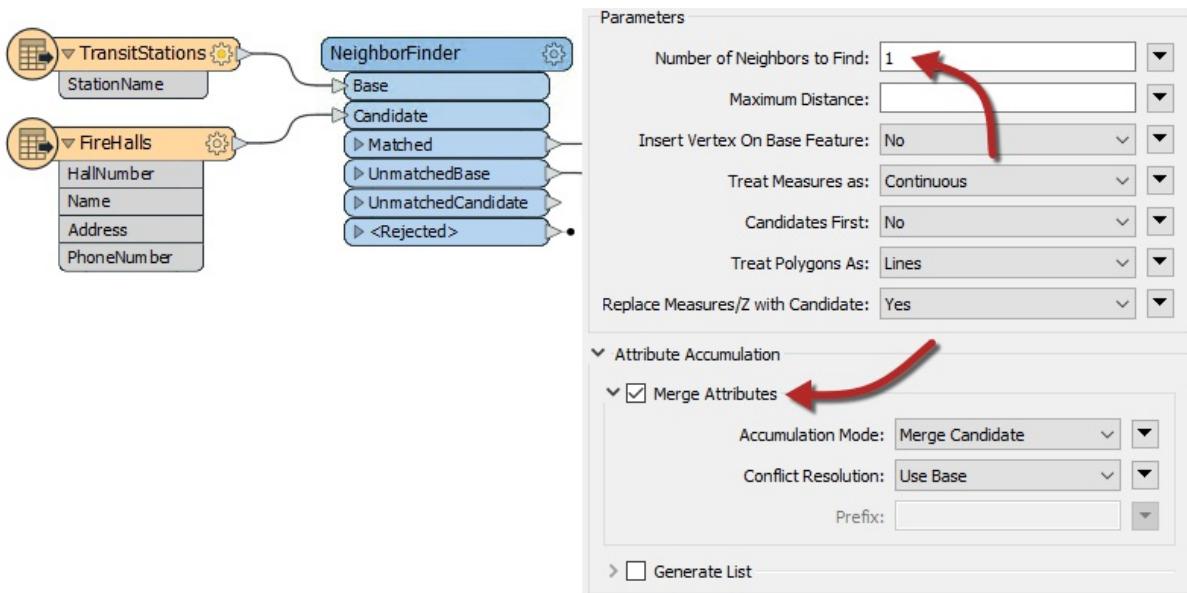
Here the TransitStation features are being provided with a postal code (CFSAUDID) depending on which PostcodeBoundary polygon they fall inside.

The "\_overlaps" attribute is another useful outcome of this transformer. It tells us how many polygons each station fell inside; in this case, overlapping postal codes might be spotted by a station having more than one overlap.

Conversely, the Area output would have an "\_overlaps" attribute that would tell us how many stations fell inside each postal code.

## NeighborFinder

The NeighborFinder transformer carries out a spatial join based on a proximity relationship. Here the NeighborFinder is being used to identify the closest fire hall to each transit station:

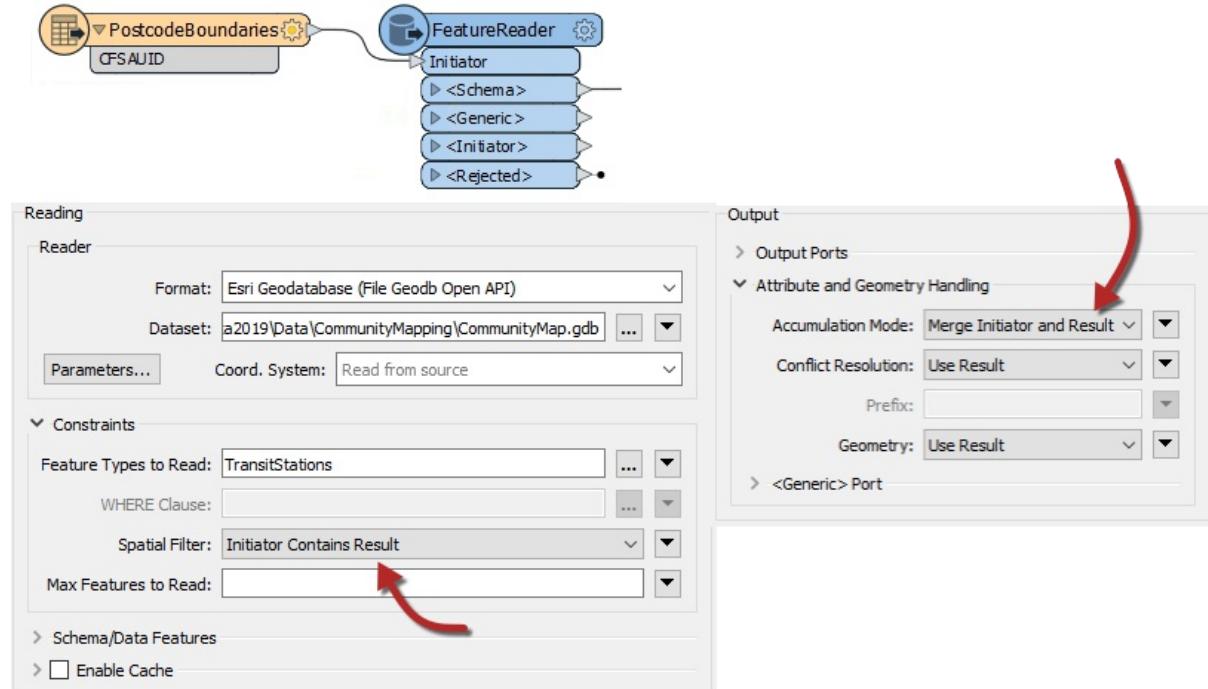


The parameters of the NeighborFinder includes the ability to specify a maximum distance for the relationship, or the maximum number of neighbors to find.

## FeatureReader

The FeatureReader is the spatial equivalent of the DatabaseJoiner transformer. It reads from an external dataset and forms a match based on a spatial relationship between the initiating feature and features in that dataset.

One difference is that the output is not the original feature, but the queried feature; hence the name FeatureReader:

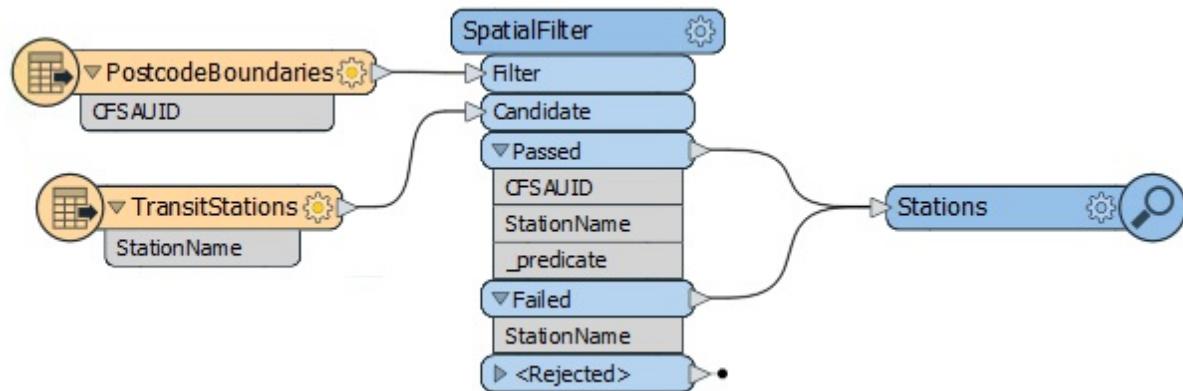


For example, here the FeatureReader is being used to carry out the same overlay of transit stations and postal codes. The PostcodeBoundaries features are read into the workspace and used as a means to spatially query TransitStations (a table in a Geodatabase). The stations are retrieved with the attributes of the postcode feature they fall inside.

This also acts as a form of filter, as stations are not outputted unless they fall inside the postcode boundary.

## SpatialFilter

The SpatialFilter - as its name suggests - filters data according to a spatial relationship. However, it does also merge attributes from one feature to another, therefore can be said to be a type of Spatial Join.



The important part is to connect both Passed and Failed output ports unless you do want to also filter the data.

## Exercise 4

## Noise Control Laws Project (Crime Data Joins)

<b>Data</b>	Crime Statistics (CSV)
<b>Overall Goal</b>	Carry out a join between crime statistics and address features
<b>Demonstrates</b>	Attribute-Based Joins
<b>Start Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\Transformers-Ex4-Begin.fmw
<b>End Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\Transformers-Ex4-Complete.fmw

As you know, city councilors have voted to amend noise control laws and residents living in affected areas were informed of these changes.

In the first part of the project, you created a workspace to convert addresses from Geodatabase to Excel, mapping the schema at the same time. In the second part of the project, you continued the workspace to locate all single-family residences within 50 meters of a major highway and filter out all others from the stream of address data.

Now a data journalist with a national newspaper is concerned that the relaxation of noise control laws may lead to more crime in the city. They have therefore requested recent crime figures for each of the affected addresses. They intend to compare this against future data to see if their theory is correct.

This request is a significant test of the city's open data policy, and there's no question of not complying. However, a crisis arises as the current datasets for crime (CSV, table data) is not joined to the address database in any way.

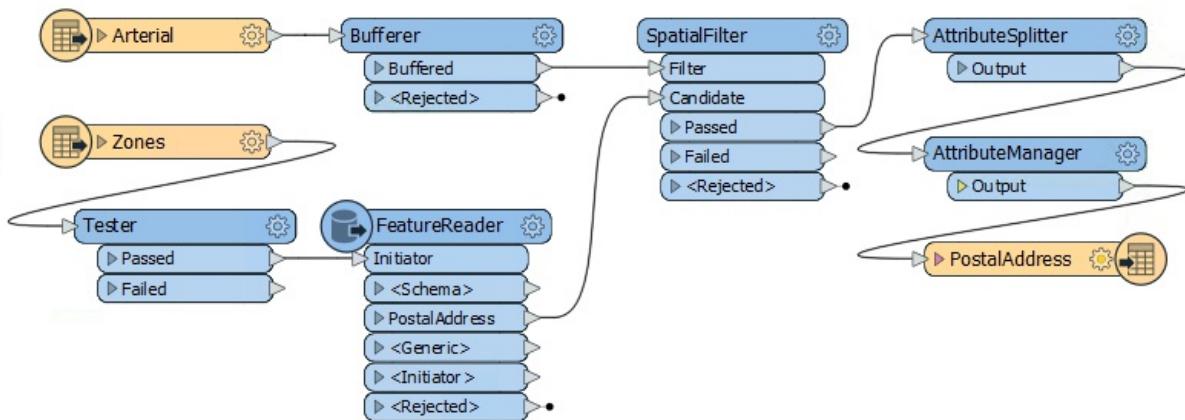
So, for the final part of this project, you must take the existing noise control workspace and amend it to incorporate crime statistics.

Pull this off, and you will be a data superhero!

### 1) Start FME Workbench

Start FME Workbench (if necessary) and open the workspace from Exercise 3. Alternatively you can open C:\FMEData2019\Workspaces\DesktopBasic\Transformers-Ex4-Begin.fmw

The workspace is already set up to read addresses, filter them spatially, and write them to an Excel spreadsheet.



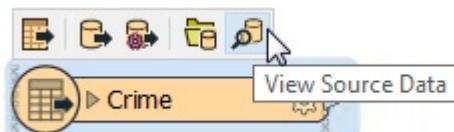
### 2) Add a CSV Reader

Now let's start working with crime data. There is no benefit from using a FeatureReader, so add a reader to the workspace using Readers > Add Reader from the menu bar. Use the following parameters:

<b>Reader Format</b>	Comma Separated Value (CSV)
<b>Reader Dataset</b>	C:\FMEData2019\Data\Emergency\Crime.csv
<b>Reader Parameters</b>	Feature Type Name(s): From File Name(s) Fields:Delimiter Character: , (Comma)

### 3) Inspect the Data

The next task is to familiarize yourself with the source data. Click on the Crime feature type to open the popup menu, then click on the View Source Data button.



The data will look like this in Visual Preview:

**Visual Preview**

Table

Crime

ID	Type	Month	Block
1	Theft From Aut...	01	63XX BOUNDARY RD
2	Theft From Aut...	01	19XX E HASTINGS ST
3	Theft From Aut...	01	RICHARDS ST / W GEORGIA ST
4	Theft From Aut...	01	8XX AVISON WAY
5	Theft From Aut...	01	56XX EAST BLVD
6	Theft From Aut...	01	7XX W GEORGIA ST
7	Theft From Aut...	01	8XX AVISON WAY
8	Theft From Aut...	01	39XX INVERNESS ST
9	Theft From Aut...	01	38XX W 29TH AV
10	Theft From Aut...	01	17XX BARCLAY ST

in any column 23684 row(s)

Notice how there is only data in the Table View, if you open the Graphics View it says that there is no geometry. We will need to use Block to spatially relate the crime data to our other data.

### FME Lizard says...

*Since this is a crime dataset, the exact numbers are blocked out by Xs. Be aware that 7XX W Georgia Street means the seventh block on Georgia Street west of Ontario Street and covers building numbers 700-800. 7XX E Georgia Street would be 14 blocks away, the seventh block east of Ontario.*

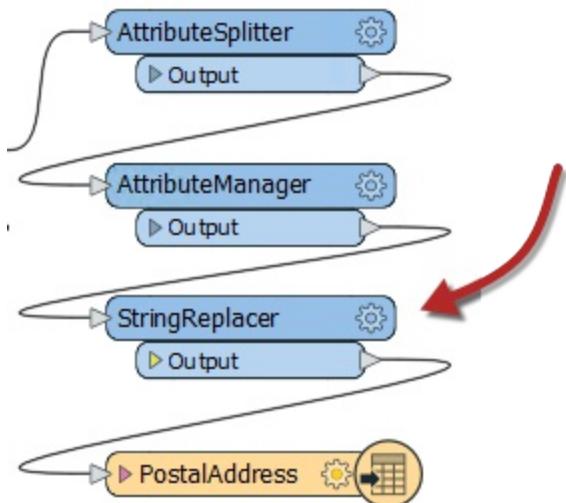
You might have spotted that each address feature has a number (not a block ID like "7XX"), and that the road data is stored in Title case ("W Georgia St") in the roads dataset, whereas the crime dataset is upper case ("W GEORGIA ST").

Both of these will make it harder, but not impossible, to join the two sets of data together.

### 4) Add a StringReplacer Transformer

To merge the data we need to reduce the address number to a block number that matches the crime data in structure; for example, we need 74XX instead of 7445.

So, add a StringReplacer transformer and connect it between the AttributeManager and the PostalAddress feature type:



Set the following parameters:

Attributes	Number
Mode	Replace Regular Expression
Text to Replace	..\$
Replacement Text	XX

The text to replace (..\$) means to replace the last two characters of the string, and they are replaced with XX to match the crime data.

Parameters

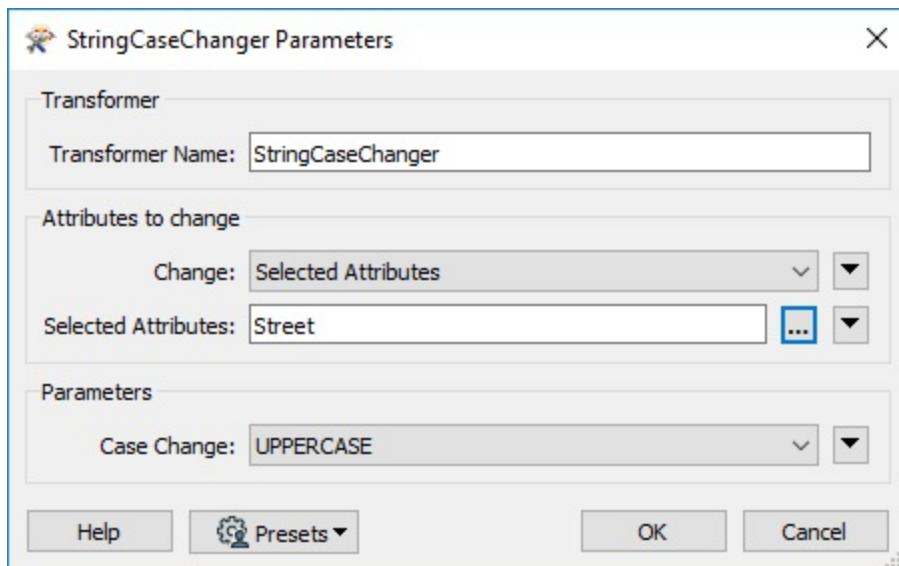
Attributes:	Number
Mode:	Replace Regular Expression
Case Sensitive:	No
Text To Replace:	..\$
Replacement Text:	XX

Run the workspace (using *Run to This* on the StringReplacer) and inspect the caches to ensure the transformer is working as expected. Each of the features in the Number column should have XX at the end.

## 5) Add a StringCaseChanger Transformer

The other difference in crime/road data was in UPPER>Title case street names. This disparity can be fixed with a StringCaseChanger transformer.

Add a StringCaseChanger transformer after the StringReplacer and set the parameters to change the value of Street to upper case:



## 6) Build Join Key

Having updated the attributes to match the crime data, we now have to construct a key out of them.

Add an AttributeCreator to the canvas after the StringCaseChanger. Create a new attribute called JoinKey. Open the Text Editor for the attribute and enter (select):

```
@Trim(@Value(Number) @Value(Street))
```

This will match the structure of the crime data (be sure to include a space character between the two attributes). The Trim function is there to ensure there are no excess spaces on those attributes.

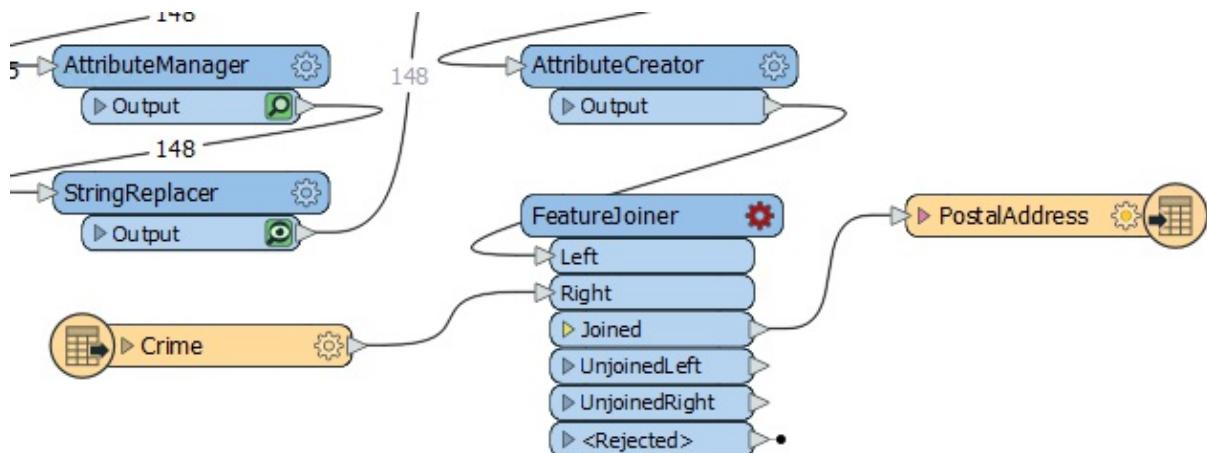
## 7) Add a FeatureJoiner Transformer

Now we've sorted out the structure of our join keys we can merge the data together with a FeatureJoiner.

Add a FeatureJoiner to the canvas.

Connect the address data (the AttributeCreator output) to the Left input port. Connect the crime data (the CSV reader feature type) to the Right input port.

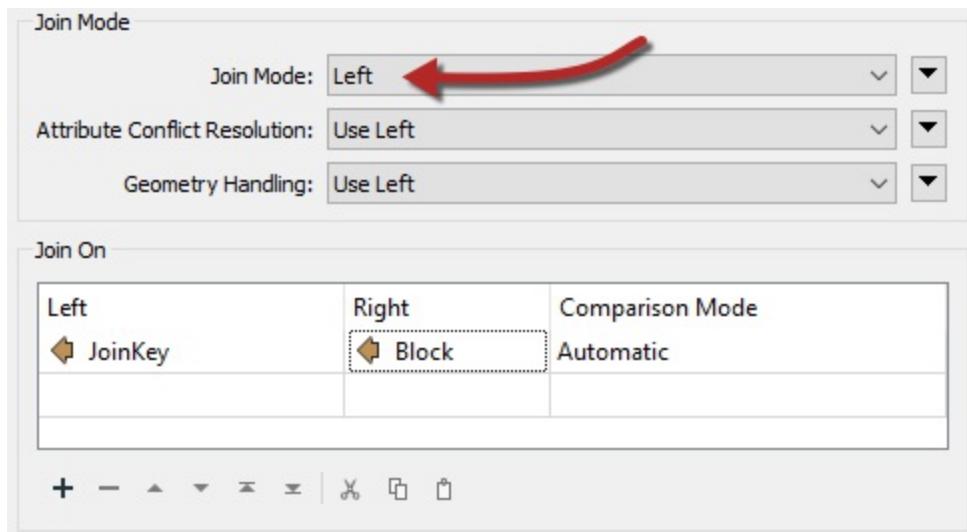
Connect the FeatureJoiner:Joined output port to the PostalAddress writer feature type:



Inspect the parameters for the FeatureJoiner.

For the Join Mode select *Left*. This means that we want all of the addresses to be output, whether they match to a crime record or not.

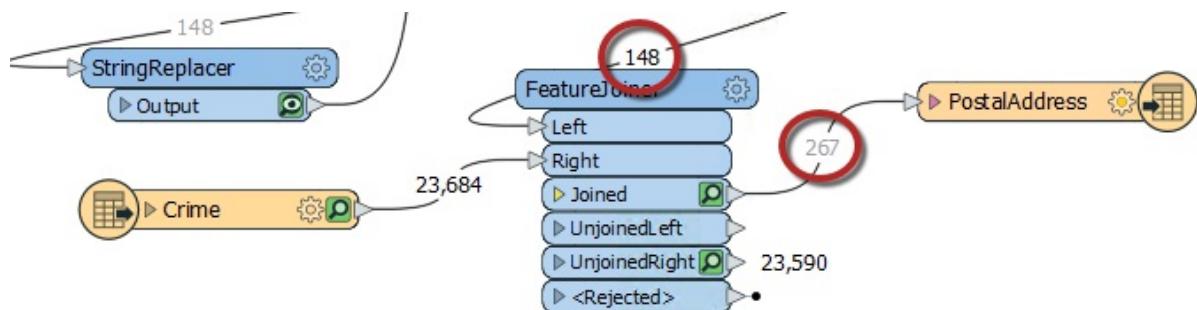
In the Join On parameters select the JoinKey attribute for the Left value, and the Block attribute for the Right value.



Run that part of the workspace to see what the results of this translation are.

### 8) Add a Aggregator Transformer

Look at the FeatureJoiner to see the Joined feature counts. Interestingly, although 148 addresses enter the FeatureJoiner, 267 emerge from it:



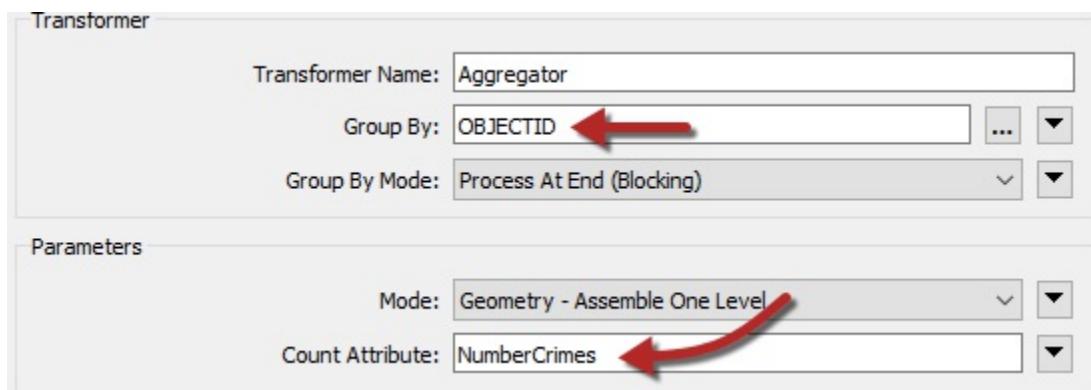
That's because there are multiple crimes per block and there were 267 matches with the data.

We can aggregate that data together using an Aggregator transformer. So place an Aggregator transformer after the FeatureJoiner:Joined port

Inspect the parameters. We need to set the group-by parameter by selecting attributes that will group matches back into the original addresses. There is no ID for each address because we removed them in a previous step, so either:

- Return to the AttributeManager, undo the Remove option for OBJECTID by switching the Action to Do Nothing, and use OBJECTID as the Aggregator group-by
- Use UpdateDate as the Aggregator group-by (because each address will have received a unique timestamp)

Then set the Count Attribute to a value of NumberCrimes:



### 9) Write Data

If you expand your attributes for the PostalAddress writer, you will notice that NumberCrimes doesn't appear. When we edited the User Attributes for this writer in an earlier exercise, we changed its Attribute Definition from Automatic to Manual when we added the Event value. This means it will not automatically update to add new attributes created during translation. Therefore we have to either switch back to Automatic (which would bring along many other unwanted attributes), or simply add a new attribute called NumberCrimes here. Give it the type "number" and cell width 6. The data from the Aggregator will now have its attribute on the writer:

Attribute Definition

Automatic  Manual  Dynamic

Name	Type	Cell Width	Formatting	Value
▶ Street	string	7	Edit...	
▶ City	string	5	Edit...	
▶ Province	string	9	Edit...	
▶ PostalCode	string	11	Edit...	
▶ Country	string	8	Edit...	
▶ Provider	string	9	Edit...	
▶ UpdateDate	string	11	Edit...	
▶ Event	string	6	Edit...	<input type="checkbox"/> Noice Control Update
▶ NumberCrimes	number	6	Edit...	
▶				

+ - ▲ ▼ ▶ ▷ ⌂ ⌃ ⌄ Filter

Finally, turn off feature caching and rerun the entire workspace. View the written data in Visual Preview. The data will include the number of crimes, and the reworking of the attributes means that individual addresses have been anonymized as well. This is important because this data is being made public.

### Advanced Exercise

*This workflow was complex with data coming from multiple sources and lots of attribute changes. If you have time, it would be a good idea to add some bookmarks and annotations to tidy up the workspace before we wrap up this workspace, in case we need to come back to it in the future.*

### CONGRATULATIONS

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

- Pre-process data to get join keys with a matching structure
- Build a join key for use in a FeatureJoiner
- Join non-spatial data with a join key in the FeatureJoiner
- Use an Aggregator transformer to merge joins and count the number of joins



## **Module Review**

This module was designed to introduce you to a broader range of FME transformers, plus techniques for applying transformers more efficiently.

## **What You Should Have Learned from this Module**

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

### **Theory**

- There are distinct groups of transformers that do work other than transforming data attributes or geometry
- FeatureReader and FeatureWriter transformers read and write data at any point in a workspace
- Integrated editing dialogs allow the author to replace transformers with built-in functions
- A large proportion of the most-used transformers are related to attribute-handling
- Filtering is the act of dividing data. Conditional filtering is the act of dividing data using a test or condition
- Data joins are carried out by transformers that merge data, from within Workbench or from external data sources

### **FME Skills**

- The ability to locate a transformer to carry out a particular task, without knowing about that transformer in advance
- The ability to use FeatureReader and FeatureWriter transformers
- The ability to read data from and write data to, web services
- The ability to build strings and calculate arithmetic values using integrated tools
- The ability to use common transformers for attribute management
- The ability to use transformers for filtering and dividing data
- The ability to use transformers for merging data together

## Practical Transformers Quiz

Each section ends with a quiz to test your new knowledge. Make your selection and click "Check my answers" to check each individual question. If you want an explanation for the answer, click "Explain".

**Note:** your score won't be tallied; this is just for review purposes.

1) Which of the following is NOT a category of transformers?

- A. Attributes
- B. Workflows
- C. Images
- D. Data Quality

Match the following transformer to its category:

1) Chopper

- A. Attributes
- B. Geometry
- C. Rasters
- D. Calculated Values

2) NullAttributeMapper

- A. Attributes
- B. Geometry
- C. Rasters
- D. Calculated Values

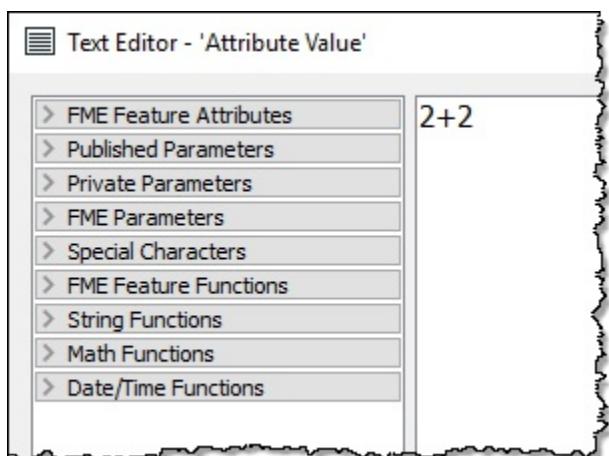
3) ImageFetcher

- A. Attributes
- B. Geometry
- C. Rasters
- D. Calculated Values

4) ExpressionEvaluator

- A. Attributes
- B. Geometry
- C. Rasters
- D. Calculated Values

5) Look at this screenshot of an editing dialog and tell me what the value returned to the attribute will be:



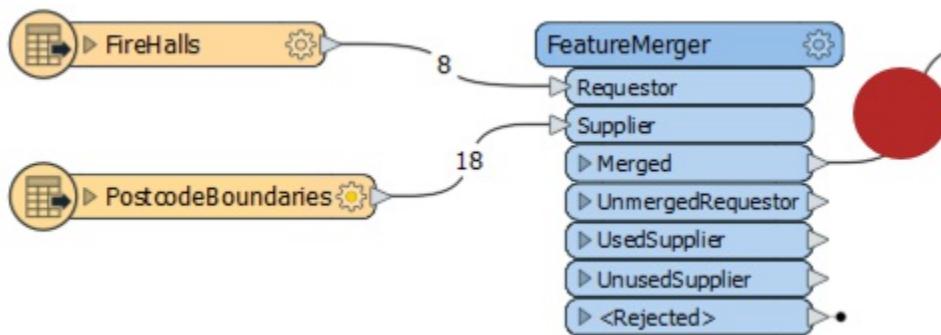
- A. 2+2
- B. 4
- C. 4.0

D. Error!

6) Which is the BEST transformer to use, if you wanted to test for a Yes/No value?

- A. Tester
- B. ExpressionEvaluator
- C. AttributeManager
- D. AttributeFilter

7) Look at the following screenshot, then answer how many features will appear in the output connection...



- A. Eight (8)
- B. Eighteen (18)
- C. Twenty-six (26)
- D. Can't tell

## Best Practice

If a workspace runs to completion and produces the output you want, it can't be bad, can it? Well, yes it can. It's not enough just to put together a functioning workspace; it's also vital to use FME in a manner that is both efficient and scalable.

### What is Best Practice?

In general terms best practice means the best way of doing something; in other words, carrying out a task effectively and efficiently.

Despite the word 'best,' we're not presuming the ideas here will meet every need and occasion. The best description of this concept I've heard – and one that fits well here – is:

"A very good practice to consider in this situation based on past experience and analysis."

### Why Use Best Practice?

Best practice in FME can help a user to...

- Debug a workspace when it doesn't work the way intended
- Use FME in a project-based environment
- Create workspaces that use the correct functionality in the correct place
- Create high-performance workspaces
- Use FME Workbench in a more efficient way
- Create well-styled workspaces that are self-documenting

#### FME Lizard says...

*I learned about best practice the hard way, when I was tasked with working on a set of someone else's workspaces. They were so badly organized the whole operation took me three times as long as it should have and ruined my plans to spend the afternoon basking in the sun!*

In this chapter, we'll cover three different aspects of best practice:

#### Debugging

This section covers tools and methods to help identify and fix problems in translations.

#### Methodology

This section covers which techniques make efficient use of FME Workbench and its components - and which don't! Using FME Workbench the right way makes for a more productive and efficient experience.

#### Style

This section is a guide to the preferred design for workspaces. The correct style makes a workspace easier to interpret, particularly in the long run when the author might return to it after a period of inactivity.

#### FME Lizard says...

*FME best practices are very much like software development best practices. For example, a computer programmer adds comments to their code to explain their actions, and you should do the same in*

*FME.*

## Debugging

Even skilled FME users seldom produce new workspaces with zero defects. For that reason, it's essential that all users are aware of the debugging techniques available in FME.

### Debugging Methodology

Generally, debugging in FME consists of first determining whether a problem has occurred and then tracking down the source of the problem (for example, where in the workspace does it occur). Once it's determined where a problem arises, the nature of the problem can be investigated.

There are various debugging techniques available in FME, and it's important to use these in the correct order. For example, it's not particularly useful to change parameters and re-run the workspace when a simple log message already explains the issue!

A logical order would be:

- Check for any problem
  - Interpret the log for warnings and errors
  - Inspect the output datasets
- Locate the problem
  - Review connection feature counts
  - Inspect the data at key stages of the translation
- Determine the problem
  - Check reader, writer, or transformer parameters at the point of failure
  - If necessary, run the workspace in feature debug mode

### FME Lizard says...

*In the world of software development, debugging a process after completion is called "post-mortem debugging"! We're trying to find out what caused a fatality.*

*If we can't determine the cause by post-mortem, we'll re-run the process with various tracing options turned on. We can call that "interactive debugging."*

## Logging and Log Interpretation

FME logs contain a record of all stages and processes within a translation. The contents are therefore vital for debugging purposes.

### Log Message Types

There are different message types that show in the log window including:

**Error:** An error, denoted in the log by red text and the term **ERROR**, indicates that a problem has caused FME to cease processing. For example, FME is unable to write the output dataset because of incorrect user permissions.

**Warning:** A warning, denoted by blue text and the term **WARN**, indicates a processing problem. The problem is sufficiently minor to allow FME to complete the translation, but the output may be adversely affected and should be checked. For example, FME is unable to write features because its geometry is incompatible with the writer format. The features will be dropped from the translation and a warning issued in the log.

**Information:** Information messages, denoted by the term **INFORM**, indicate a piece of information that may help a user determine whether their translation has been processed correctly. For example, FME sometimes logs confirmation of a particular dataset parameter, such as the coordinate system.

**Statistics:** Statistics messages, denoted by the term **STATS**, provide information on various numbers relating to the translation; for example, the number of features read from a source dataset, and the time it took to do so.

### Spatial Log File

Besides writing the log to a text file (<workspace name>.log) FME also writes a spatial log:

 Design-Ex3-Complete.fmw	Type: FMW File	Date modified: 3/27/2019 9:50 AM
 Design-Ex3-Complete.log	Type: Text Document	Date modified: 3/27/2019 9:50 AM Size: 25.6 KB
 Design-Ex3-Complete_log.ffd	Type: FFS File	Date modified: 3/27/2019 9:50 AM Size: 5.01 KB



The spatial log is a dataset of features (in FME Feature Store format) that have been mentioned in the log - either because of a warning from FME, use of the Logger transformer or <Rejected> features.

The dataset can be opened within FME Data Inspector or the Visual Preview window in FME Workbench to inspect the features and identify any problems that caused them to be rejected.

### Interpreting the Log Window

The log window should be the **first** place to check when a translation is completed. It will tell the user whether there are any concerning errors or warnings.

#### Errors

If an **ERROR** occurs, it is likely that the translation will be halted. There will be a lot of red text and some statements such as:

Program Terminating

Translation FAILED.

There may be several **ERROR** messages, so scroll back up the log window to try and identify the first of these, which is likely to be the cause of the problem. For example this message:

```
ERROR |Error connecting to PostgreSQL database(host='postgis.train.safe.com', port='5432',  
dbname='fmedata', user='fmedata', password='*'): 'FATAL: password authentication failed for user  
'fmedata'" FATAL: password authentication failed for user "fmedata"
```

...is an obvious problem with authenticating a database connection.

## Warnings

Even when a translation succeeds, it's important to check the log for the following comment:

Translation was SUCCESSFUL with X warning(s)

If there are any warnings (for example, if  $X > 0$ ) then use the search option to look for the word WARN. Any warning messages might have important consequences for the quality of the output data.

## Inspecting Output

Even if a workspace ran to completion without warnings or errors, it does not follow that the output matches what is expected or required. For whatever reason, the workspace may be producing data in the wrong manner. We can determine this by inspecting the translation output.

Inspecting the output is merely a case of viewing it in either Visual Preview or in the application in which the data is intended to be used.

As noted already in this manual, a number of different aspects of data may be inspected, including the following:

- **Format:** Is the data in the expected format?
- **Schema:** Is the data subdivided into the correct layers, categories, or classes?
- **Geometry:** Is the geometry in the correct spatial location? Are the geometry types correct?
- **Symbology:** Is the color, size, and style of each feature correct?
- **Attributes:** Are all the required attributes present? Are all integrity rules being followed?
- **Quantity:** Does the data contain the correct number of features?
- **Output:** Has the translation process restructured the data as expected?

It should be straightforward to check a dataset and see if any of its components are incorrect.

### FME Lizard says...

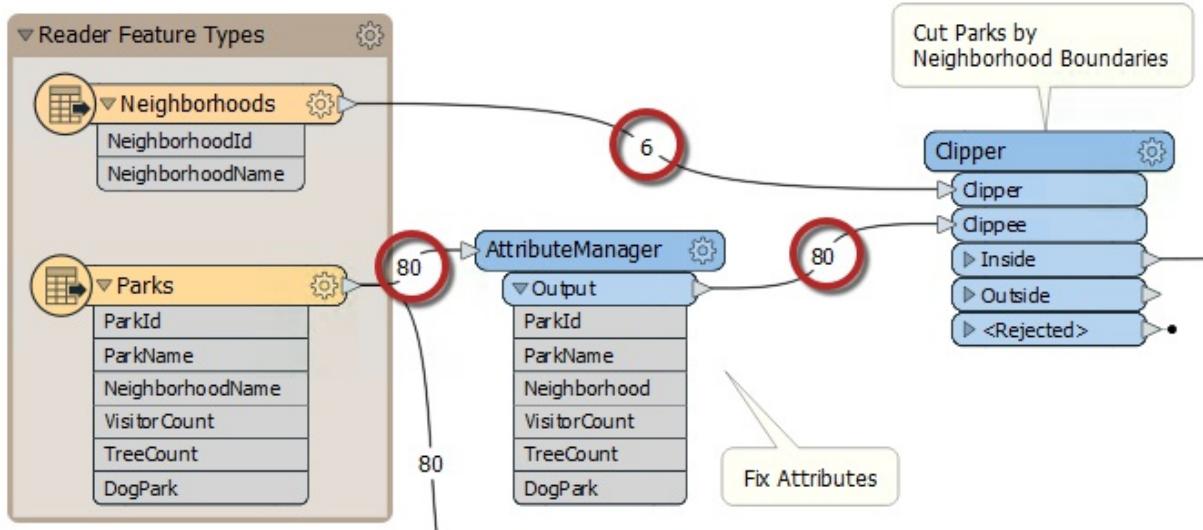
*This stage is solely to determine IF there are any problems.*

*If there are no problems, then we can be satisfied the translation was a success.*

*If there are problems, we should go on to determine where the problem occurred. It's important not to jump to conclusions at this point. The fact that the output is incorrect does not tell us **where** that issue was introduced.*

## Feature Counts

A workspace **feature count** refers to the numbers shown on each connection once a translation is complete:



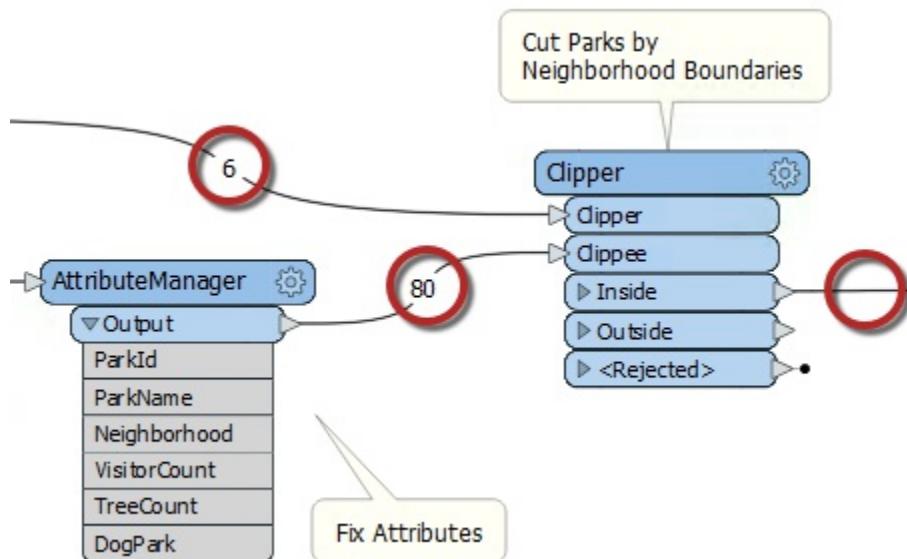
Once an error or problem has been determined to exist, feature counts help us identify *where* that problem occurred.

In the above screenshot, if the Clipper output were incorrect, then you would inspect the prior feature counts to see if any counts looked wrong (perhaps you know that there are seven neighborhoods, but the feature count shows only six).

## Incorrect Output

When the number of output features is incorrect, then there are several things to check.

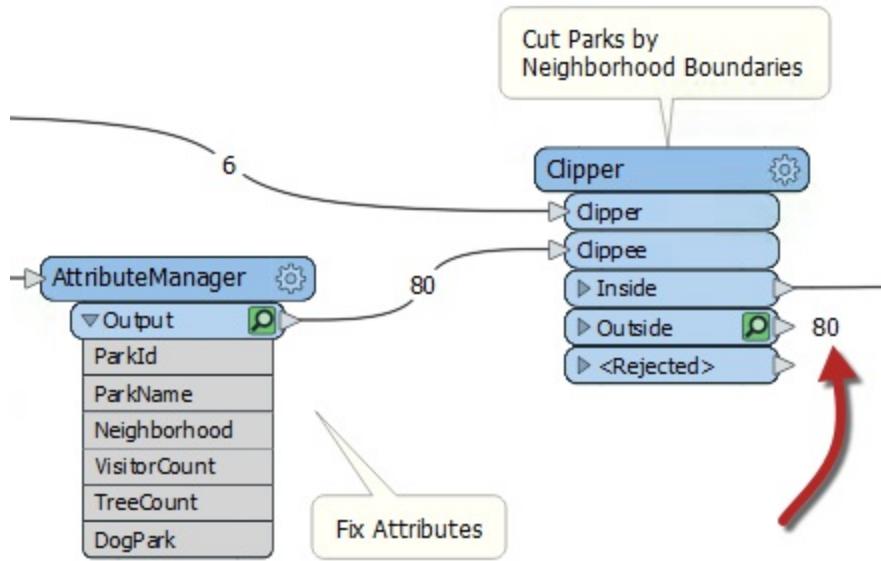
If you get zero output, and the feature counts show that all features entered a transformer, but none emerged, then you can be fairly confident that the transformer is the cause of the problem:



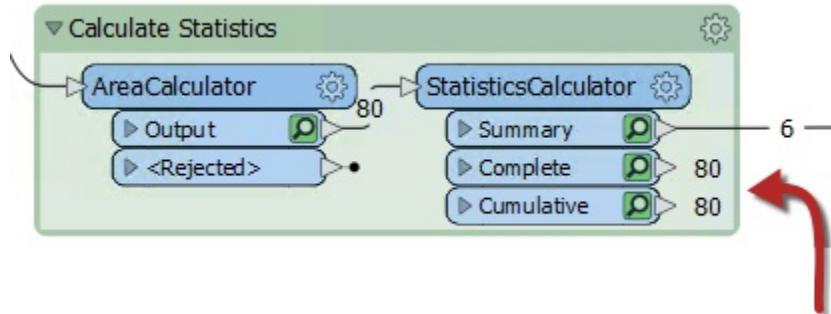
Here, for example, 80 features enter the Clipper transformer (to be clipped against a single boundary) but none emerged. The Clipper transformer is almost certainly the cause of any incorrect output.

The data is not rejected as invalid; it merely does not pass the test expected. It's possible that Clipper and Clippee don't occupy the same coordinate system; hence, one does not fall inside the other.

Turning on feature caching helps to confirm this to be the case:



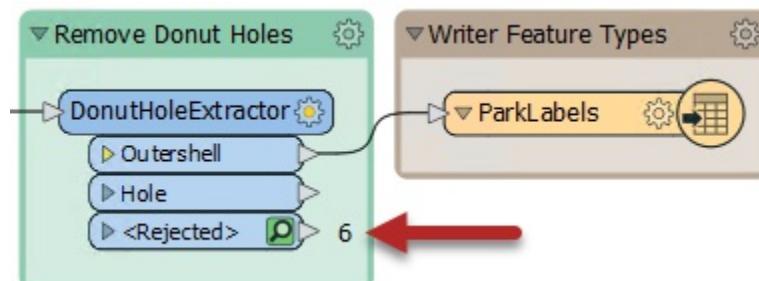
Alternatively - and this is a common cause of missing features - the author has connected the wrong output port! For example, this user has connected the StatisticsCalculator Summary output port, when they really wanted the Complete port connected:



Again feature caching gives us a clue to what port we should connect.

## Rejected Features

Sometimes when features go missing, they are being rejected by a transformer. Many transformers include a <Rejected> port to output these invalid features:



Remember, features are automatically counted and stored on a <Rejected> port, even if feature caching is turned off.

As an additional benefit, the rejected features will often include a rejection code attribute:

Table

DonutHoleExtractor\_&lt;Rejected&gt;

Columns...

	NeighborhoodName	AverageParkArea	fme_rejection_code
1	Downtown	12036.966515919528	INVALID_GEOMETRY_TYPE
2	Fairview	26152.365817752605	INVALID_GEOMETRY_TYPE
3	Kitsilano	24941.622360681748	INVALID_GEOMETRY_TYPE
4	Mount Pleasant	11660.252762014796	INVALID_GEOMETRY_TYPE
5	Strathcona	19835.90145833002	INVALID_GEOMETRY_TYPE
6	West End	266443.572024079	INVALID_GEOMETRY_TYPE

## Checking Key Stages

If the feature counts cannot help to pinpoint the location of a problem, then the next step is inspecting data at key stages of a translation.

Generally, issues in an output dataset occur due to one of these:

- The data was incorrectly read
- The data was incorrectly transformed
- The data was incorrectly written
- The data is being incorrectly interpreted by another application

### Example

A translation from a File Geodatabase of addresses to CSV format renders attribute values incorrectly in the output. For example, the name "麥毅倫" is rendered as "???".

Obviously, an encoding problem has occurred, but where? The fact that data appears incorrect in another application is no real indication of where that problem was introduced.

In this scenario, the author should check the data at key stages to determine when the data last looked correct.

### Inspect the Source Data Before Reading

It's easy to assume source data is correct without checking it. If possible, the source data should be inspected in its native application to confirm its accuracy:

ArcCatalog - C:\FMEData2019\Data\Addresses\Addresses.gdb\PostalAddress						
File Edit View Go Geoprocessing Customize Windows Help						
C:\FMEData2019\Data\Addresses\Addresses.gdb\PostalAddress						
Contents Preview Description						
OBJECTID *	GlobalID	OWNERNM1	OWNERNM2	PSTLADDRESS	PSTLCITY	
10	{591FE3D2-	Many Purdy	<Null>	98 Main St	Vancouver	British
11	{59F2BAFB	Kent Garbett	<Null>	125 Milross Av	Vancouver	British
12	{FA63E6CC	Ludivina Rego	Shana Cerda	2520 Manitoba St	Vancouver	British
13	{D83E4430-	Gerald Woodburn	<Null>	127 Melville St	Vancouver	British
14	{30254C23-	Marisa Marmol	麥毅倫	1001 Lagoon Drive	Vancouver	British
15	{A26C32CE	Everette Cavazos	<Null>	565 Smithe St	Vancouver	British
16	{86247FB2-	Galen Hinnenkamp	<Null>	729 Campbell Av	Vancouver	British

Obviously, if the data is incorrect at the source, then there is little chance the translation output will be correct. However in our example, it is correct in ArcGIS, so the translation should work.

### Inspect the Source Data After Reading

If the source data is correct in its native application, then inspect it using FME. Either open the data directly in the FME Data Inspector or - if the workspace was run with caching turned on - open it from within FME Workbench:

Table View							
FeatureReader_PostalAddress - FeatureReader_PostalAddress							
	OBJECTID	GlobalID	OWNERNM1	OWNERNM2	PST ADDRESS	PSTLCITY	PSTLPROV
10	10	591FE3D2-95E0...	Many Purdy	<null>	989 Main St	Vancouver	British Columbia
11	11	59F2BAFB-88C5...	Kent Garbett	<null>	125 Milross Av	Vancouver	British Columbia
12	12	FA63E6CC-D4F...	Ludivina Rego	Shana Cerda	2520 Manitoba St	Vancouver	British Columbia
13	13	D83E4430-7376...	Gerald Woodburn	<null>	1277 Melville St	Vancouver	British Columbia
14	14	30254C23-A9F3...	Marisa Marmol	麥毅倫	1001 Lagoon Drive	Vancouver	British Columbia
15	15	A26C32CE-6485...	Everette Cavazos	<null>	565 Smithe St	Vancouver	British Columbia
16	16	86247FB2-5847-...	Galen Hinnenkamp	<null>	729 Campbell Av	Vancouver	British Columbia

If the data is incorrect at this point, then the process of reading the data with FME is at fault. Again, in our example the data is correct, so we must continue to diagnose the problem.

## Inspect the Data Before Writing

Now we should inspect the data between transformation and writing.

If the workspace was run with caching turned on (Run > Enable Feature Caching), then the data is available for inspection already. Otherwise turn on this option (or Writers > Redirect to FME Data Inspector) and re-run the workspace.

Inspect the data to see if it was correct at the point just before it is written to the output:

Table View						
inspector [FFS] - PostalAddress						
	Owner1	Owner2	Number	Street	City	Province
10	Many Purdy	<null>	989	Main St	Vancouver	British Columbia
11	Kent Garbett	<null>	125	Milross Av	Vancouver	British Columbia
12	Ludivina Rego	Shana Cerda	2520	Manitoba St	Vancouver	British Columbia
13	Gerald Woodburn	<null>	1277	Melville St	Vancouver	British Columbia
14	Marisa Marmol	麥毅倫	1001	Lagoon Drive	Vancouver	British Columbia
15	Everette Cavazos	<null>	565	Smithe St	Vancouver	British Columbia
16	Galen Hinnenkamp	<null>	729	Campbell Av	Vancouver	British Columbia

If Visual Preview shows ♦ characters at this point, then we can assume that the problem occurs in data transformation, before it is written. You can use Feature Caching to inspect each step of the process to locate at which transformer the problem appears.

In our example the data is still correct, so we should test the output dataset next.

## Inspect the Output Dataset

If the data is correct before writing, then it might be that it is being written incorrectly.

Open the output dataset in Visual Preview. This step will show the data as FME wrote it (and, of course, read it back). If the data is incorrect here, then the problem will have likely occurred during the writing of the data:

Visual Preview

Table

CSV

	Owner1	Owner2	Number	Street	City	Province	PostalCode	Country	Provider	UpdateDate
10	Many Purdy		989	Main St	Vancouver	British Columbia	V6A0Z9	Canada	Safe Software	20190327104200...
11	Kent Garbett		125	Milross Av	Vancouver	British Columbia	V6A6P3	Canada	Safe Software	20190327104200...
12	Ludivina Rego	Shana Cerd...	2520	Manitoba St	Vancouver	British Columbia	V5T8D1	Canada	Safe Software	20190327104200...
13	Gerald Woodbu...		1277	Melville St	Vancouver	British Columbia	V6E7B2	Canada	Safe Software	20190327104200...
14	Marisa Marmol	???	1001	Lagoon Drive	Vancouver	British Columbia	V6G4W5	Canada	Safe Software	20190327104200...
15	Everette Cavazos		565	Smithe St	Vancouver	British Columbia	V6B5Z4	Canada	Safe Software	20190327104200...
16	Galen Hinnenkamp...		729	Campbell Av	Vancouver	British Columbia	V6A2T7	Canada	Safe Software	20190327104200...
17	Sebastian Files		199	Keefer Place	Vancouver	British Columbia	V6B4U5	Canada	Safe Software	20190327104200...
18	Rogelio Montalto		351	Abbott St	Vancouver	British Columbia	V6B2K7	Canada	Safe Software	20190327104200...
19	Olinda Dykstra	Bo Holston	1155	E 6th Av	Vancouver	British Columbia	V5T2M7	Canada	Safe Software	20190327104200...

in any column

200 row(s)

In the above example, if Visual Preview shows ♦ characters at this point, then the data has been mangled when it was being written. That appears to be the case here.

Another check to make is to open the data in a text editor. For obvious reasons it will not be possible to do this for every dataset (binary files or databases for example) but for text-based files it can provide definitive proof of whether the data is correct at this point:

PostalAddress.csv - Notepad

File Edit Format View Help

```
Owner1,Owner2,Number,Street,City,Province,PostalCode,Country,Provider,UpdateDate
Jake Warnock,,1188,W Pender St,Vancouver,British Columbia,V6E4V5,Canada,Safe Software,20190327104200...
Armand Augustyn,,1661,Ontario St ,Vancouver,British Columbia,V5Y2Q7,Canada,Safe Software
Lieselotte Cota,,535,Smithe St ,Vancouver,British Columbia,V6B8E1,Canada,Safe Software
Lieselotte Cota,,181,W 1st Av,Vancouver,British Columbia,V5Y4W5,Canada,Safe Software,20190327104200...
Ernest Ahlgren,,141,W 1st Av,Vancouver,British Columbia,V5Y0W9,Canada,Safe Software,20190327104200...
Jim Baskerville,,808,Gore Av ,Vancouver,British Columbia,V6A2T7,Canada,Safe Software,20190327104200...
Cassandra Brandis,,266,E 15th Av,Vancouver,British Columbia,V5T3S6,Canada,Safe Software
Caryl Chinn,,36,Water St ,Vancouver,British Columbia,V6B8Y1,Canada,Safe Software,20190327104200...
Gerald Woodburn,,2762,W 3rd Av,Vancouver,British Columbia,V6K6N3,Canada,Safe Software,20190327104200...
Many Purdy,,989,Main St ,Vancouver,British Columbia,V6A0Z9,Canada,Safe Software,20190327104200...
Kent Garbett,,125,Milross Av ,Vancouver,British Columbia,V6A6P3,Canada,Safe Software,20190327104200...
Ludivina Rego,Shana Cerd...,2520,Manitoba St ,Vancouver,British Columbia,V5T8D1,Canada,Safe Software
Gerald Woodburn,,1277,Melville St ,Vancouver,British Columbia,V6E7B2,Canada,Safe Software
Marisa Marmol,???,1001,Lagoon Drive ,Vancouver,British Columbia,V6G4W5,Canada,Safe Software
Everette Cavazos,,565,Smithe St ,Vancouver,British Columbia,V6B5Z4,Canada,Safe Software
Galen Hinnenkamp,,729,Campbell Av ,Vancouver,British Columbia,V6A2T7,Canada,Safe Software
Sebastian Files,,199,Keefer Place ,Vancouver,British Columbia,V6B4U5,Canada,Safe Software
Rogelio Montalto..351.Abbott St .Vancouver.British Columbia.V6B2K7.Canada.Safe Software
```

For our example, this confirms that the problem occurs when writing the data. However, this might not always be the case.

## Inspect the Output Dataset in Another Application

If FME (and a text editor) can display the output data, then it might be that the intended application is not interpreting the data correctly.

So, open the output dataset in the application in which it is intended to be used. If FME can read the data correctly, and it looks correct in a text editor, then the problem is more likely to be with how the end application interprets the data:

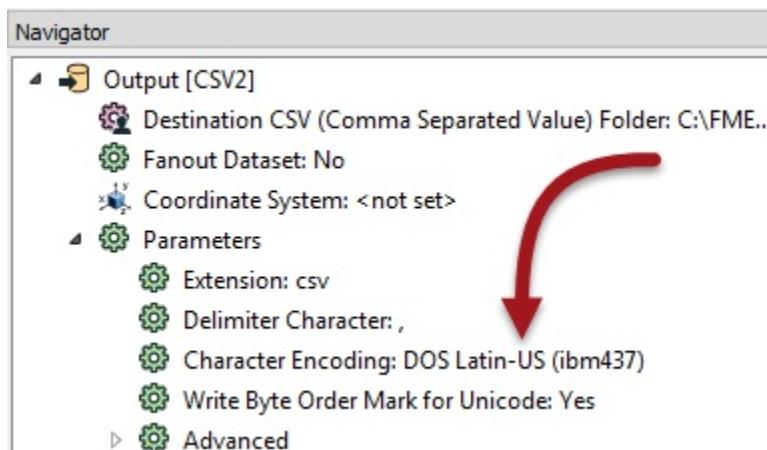


	A1	f(x)	Owner1	C	D	E	F	G	H
11	Many Purdy			989	Main St	Vancouver	British Co	V6A0Z9	Canada
12	Kent Garbett			125	Milross Av	Vancouver	British Co	V6A6P3	Canada
13	Ludivina Rego	Shana Cerda		2520	Manitoba	Vancouver	British Co	V5T8D1	Canada
14	Gerald Woodburn			1277	Melville S	Vancouver	British Co	V6E7B2	Canada
15	Marisa Marmol	???		1001	Lagoon Dr	Vancouver	British Co	V6G4W5	Canada
16	Everette Cavazos			565	Smithe St	Vancouver	British Co	V6B5Z4	Canada
17	Galen Hinnenkamp			729	Campbell	Vancouver	British Co	V6A2T7	Canada

That would be particularly true if the format were non-native to that application; for example, reading a Geodatabase outside of an Esri product.

All of these techniques narrow down where an error might have occurred, but don't always specify the cause. For example, incorrect output could mean that FME has a limitation in that writer or that the workspace author has set an incorrect parameter, or maybe one application uses a different default encoding to another.

In this case, knowing the writer is at fault, we can check the writer parameters and find one that appears to be setting an incorrect encoding:



In short, these techniques identify where to investigate first, but won't provide an absolute answer by themselves.

## TIP

*Encoding is a good example for us here, but it's also an example of where you should check that your computer is capable of viewing such data at all! If your computer is set up in the wrong locale then it might not even be possible for you.*

## Exercise 1

## Debugging a Workspace

<b>Data</b>	Addresses (Esri Geodatabase) Crime Data (CSV - Comma Separated Value) Parks (MapInfo TAB) Swimming Pools (OSM - OpenStreetMap)
<b>Overall Goal</b>	Work on Vancouver Walkability Project
<b>Demonstrates</b>	Debugging Best Practice
<b>Start Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\BestPractice-Ex1-Begin.fmw
<b>End Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\BestPractice-Ex1-Complete.fmw

You have just been assigned to take over a project from your colleague and they passed their workspace on to you. This project is to calculate the "walkability" of each address in the city of Vancouver. Walkability is a measure of how easy it is to access local facilities on foot. It will include a measure of the distance to the nearest park, the amount of crime in an area, and other similar metrics.

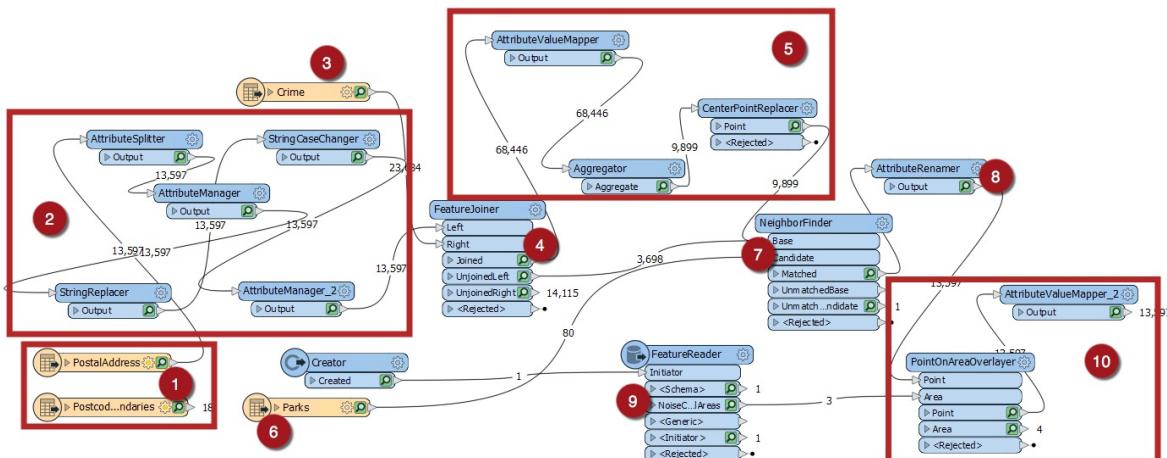
The workspace currently assesses crime, parks, and noise-control areas, but it doesn't give an overall measure of walkability.

So let's do that, and then see if there are any other aspects we can include.

### 1) Start FME Workbench

Start FME Workbench. Open the workspace template C:\FMEData2019\Workspaces\DesktopBasic\BestPractice-Ex1-Begin.fmw. Then run the workspace to cache the data.

This workspace is a bit messy, but we will fix that in a later exercise. First, let's figure out what this workspace does:



1. PostalAddress and PostcodeBoundaries are being read in from the Addresses.gdb
2. Attributes from the PostalAddress feature type are being cleaned up to create a separate Number and Street attribute. Then the last two digits of the Number are being replaced by XX to create an attribute that will be the Join Key for joining the crime data.
3. Crime.csv read in, the street number for each crime incident is protected by XX as the last two digits.
4. PostalAddress and the Crime data is joined in the FeatureJoiner based on the Join Key attribute that was created in 2. and the Block attribute from Crime.
5. The [crime] Type attribute is given a number based on severity, and then the total CrimeValue is calculated for each address block. Then with the CenterPointReplacer, only one point is extracted if there are multiple crime incidents in the same location.
6. Parks MapInfo TAB file read in. This will be used to determine if any of the crimes are in or close to a park.
7. Using the NeighborFinder the parks closest to each crime is determined
8. The \_distance attribute that was created with the NeighborFinder is renamed ParkDistance

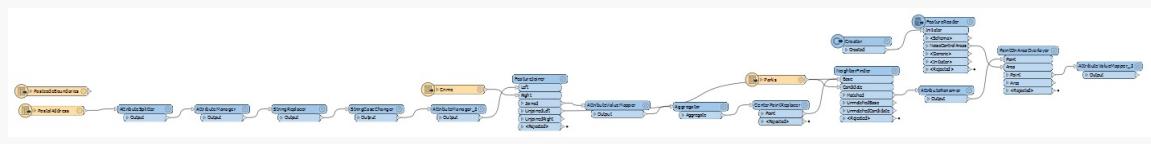
9. A Planning Restrictions OGC Geopackage is read in using the Creator and the FeatureReader. Then from that dataset, the NoiseControlAreas are read in to obtain the noise restrictions areas. This data will be joined with the crime and address data.
10. Using the PointOnAreaOverlayer, the point data containing the crime, distance to park and addresses is joined with the NoiseControlAreas polygons. This assigns the noise restrictions to any overlapping points. The AttributeValueMapper is used to assign the zone with a score, creating the attribute NoiseZoneScore.

## TIP

*This workspace is very messy as our colleague didn't follow any best practices. We will be cleaning it up in a later exercise, but if it is too hard for you to make sense of its messy state, click on the Autolayout button to quickly tidy it up:*



**Note that this will change how your workspace looks compared to the screenshots and the instructor, so pay attention to the transformer names and ports:**



## 2) Add an ExpressionEvaluator Transformer

We can create a measure of walkability that combines all of our current values using the ExpressionEvaluator transformer.

So add an ExpressionEvaluator transformer to the end of the workspace and connect it to the AttributeValueMapper\_2.

Inspect its parameters. Set it up to create a new attribute called Walkability that is:

```
@Value(ParkDistance) + @Value(CrimeValue) - @Value(NoiseZoneScore)
```

The screenshot shows the 'Parameters' dialog in FME Workbench. Under 'Evaluation Mode', 'Create New Attribute' is selected. The 'New Attribute' field contains 'Walkability'. Below it, 'Attributes To Overwrite' is set to 'No items selected.' A link 'Advanced: Attribute Value Handling' is visible. In the 'Arithmetic Expression' section, a tree view on the left lists categories like 'FME Feature Attributes', 'Published Parameters', etc. The main area displays the expression '@Value(ParkDistance)+@Value(CrimeValue)-@Value(NoiseZoneScore)'.

With this expression, the smaller the result, the better. Run the workspace. Because we ran the workspace in the beginning to cache all the data, only the ExpressionEvaluator will run.

### 3) Assess the Result

Let's assess whether the result of the translation is correct.

Firstly check the log window for errors and warnings. There are no errors, but there are several warnings, which is not a good sign:

The screenshot shows the 'Translation Log' window. It displays the following log entries:

```

223 ~~~
224 -----
225 Translation was SUCCESSFUL with 132 warning(s) (0 feature(s) output)
226 Stored 3 feature(s) to FME feature store file 'C:\FMEData2019\Workspaces\DesktopBasic\BestPractice-Ex1-Begin_log.ffd'
227 FME Session Duration: 4.0 seconds. (CPU: 3.4s user, 0.4s system)
228 END - ProcessID: 5300, peak process memory usage: 42240 kB, current process memory usage: 42240 kB
229 Translation was SUCCESSFUL

```

The 'Warnings' button in the toolbar is highlighted.

**Note:** The number of warnings showed in the Translation Log may be different, this is based on the Logging Parameters set in FME Options.

Click on the warnings button to filter out the warnings. The warnings say:

```

ExpressionEvaluator: Failed to evaluate expression '@real64(560.3272250455418+
<null>-0)'.
Result is set to null

```

Inspect the output cache on the ExpressionEvaluator, and some addresses do indeed have a Walkability value of <null>.

So we know there is a problem, let's try and figure out where the problem is and why it occurs.

#### TIP

A useful test would be to right-click on CrimeValue in the Table View window, and sort by ascending numeric order. That will put any null values to the top of the table.

#### 4) Locate the Problem

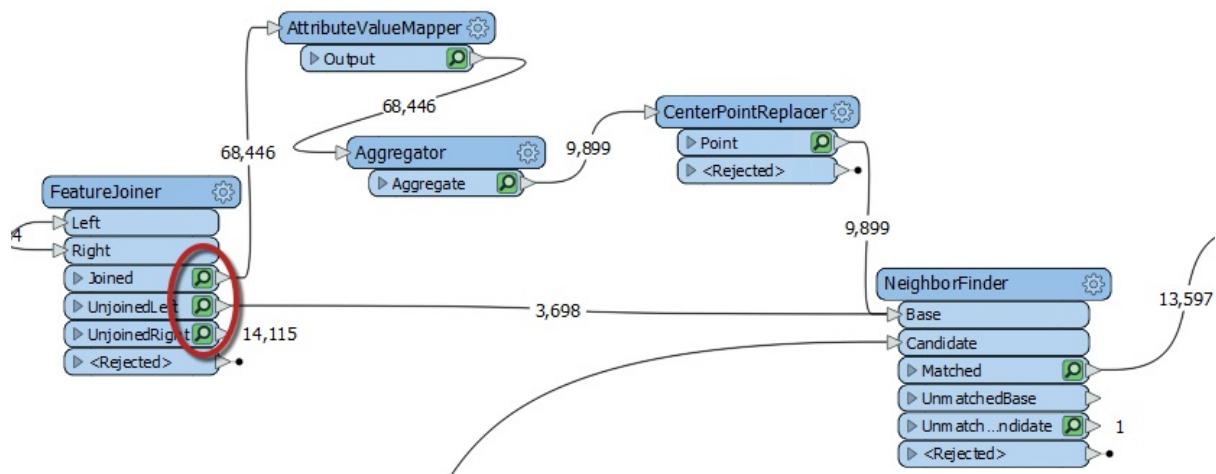
We can tell the warning comes from the ExpressionEvaluator, but that doesn't necessarily mean that is where the problem lies. The calculation fails because the middle value is <null>

If the expression is:

```
ParkDistance + CrimeValue - NoiseZoneScore
```

Then we know that it must be the CrimeValue that is an issue because it is the middle value.

Let's find out where it becomes an issue. First, organize the workspace a bit, and then inspect the caches on the FeatureJoiner transformer. We are inspecting the FeatureJoiner because that's where we first get our Crime data:



There are no <null> values coming from the FeatureJoiner, so let's move along the translation. Check the cache for the AttributeValueMapper. That's where values are set, so perhaps nulls are coming out of there?

On inspection, there are no <null> values for the CrimeValue or the crime Type attribute in there. There are also no nulls for the Aggregator and CenterPointReplacer caches.

Checking each feature cache is a bit time consuming, let's try a different method. Check the feature counts on each connection. There are 68,446 features tagged with a crime (FeatureJoiner:Joined), but then that is reduced to 9,899 after the Aggregator and then there are 3,698 features that are not tagged with a crime (FeatureJoiner:UnjoinedLeft). That gives a total of 13,597, coming out of the NeighborFinder, which is correct.

Oh. Do you see it yet? The 3,698 features that are not tagged with a crime: what CrimeValue do they get? Inspect the UnjoinedLeft output from the FeatureJoiner, and you will see that they do not have the CrimeValue attribute. That's why the ExpressionEvaluator says that there are nulls.

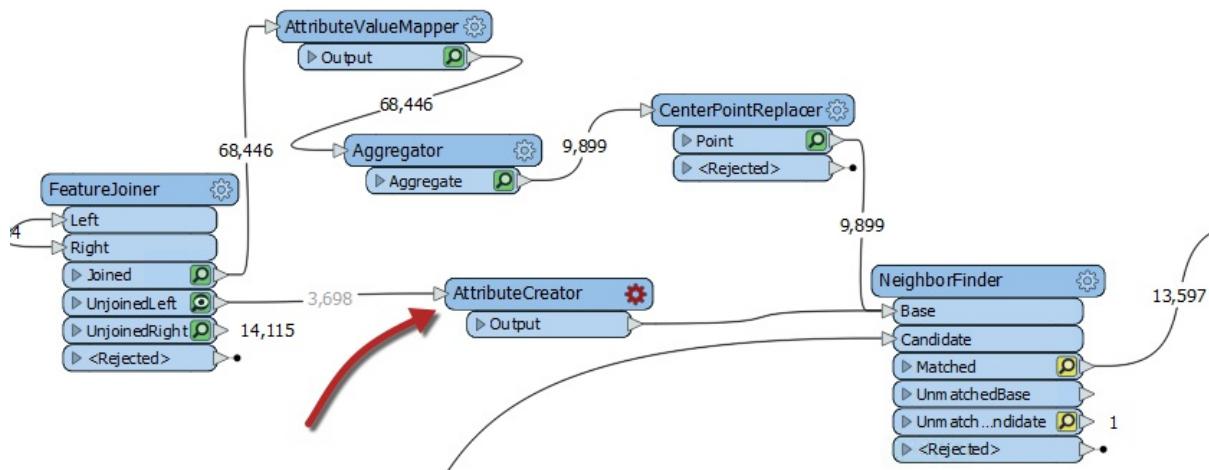
#### FME Lizard says...

*To confirm this I copied the log into a text editor and searched for the phrase "ExpressionEvaluator: Failed to evaluate expression".*

*It appeared 3,698 times, the same as the number of features that exit the UnjoinedLeft port. Coincidence?*

#### 5) Fix the Problem

If those features do not have a CrimeValue attribute, then we should give them one. To do so, add an AttributeCreator transformer to the workspace between the FeatureJoiner:UnjoinedLeft output port and the NeighborFinder:Base input port:



Open up its parameters and create an attribute called CrimeValue with a value of zero (0).

Attributes To Create	
New Attribute	Attribute Value
CrimeValue	<input type="checkbox"/> 0

Run the workspace, which will run from the AttributeCreator to the ExpressionEvaluator. You should now find that there are fewer warnings and that the output contains no <null> values.

## **6) Add Swimming Pools**

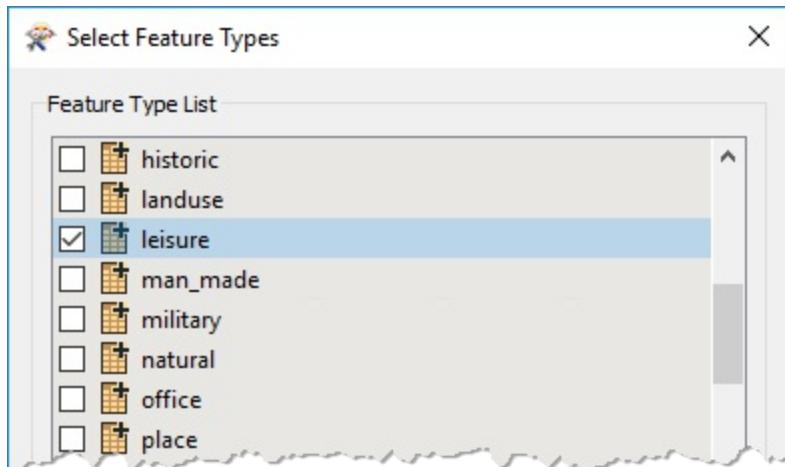
The city has decided that parks are not a great candidate for walkability scores because there is usually a park nearby. They decided to evaluate how easy it is to walk to a swimming pool, and this evaluation might be used to decide later where a new pool should be built.

We can reuse the same workflow for swimming pools that was built for parks, with just a few minor updates.

First let's add a new reader with the following parameters:

<b>Reader Format</b>	OpenStreetMap (OSM) XML
<b>Reader Dataset</b>	C:\FMEData2019\Data\OpenStreetMap\leisure.osm

When prompted, select only the leisure feature type:

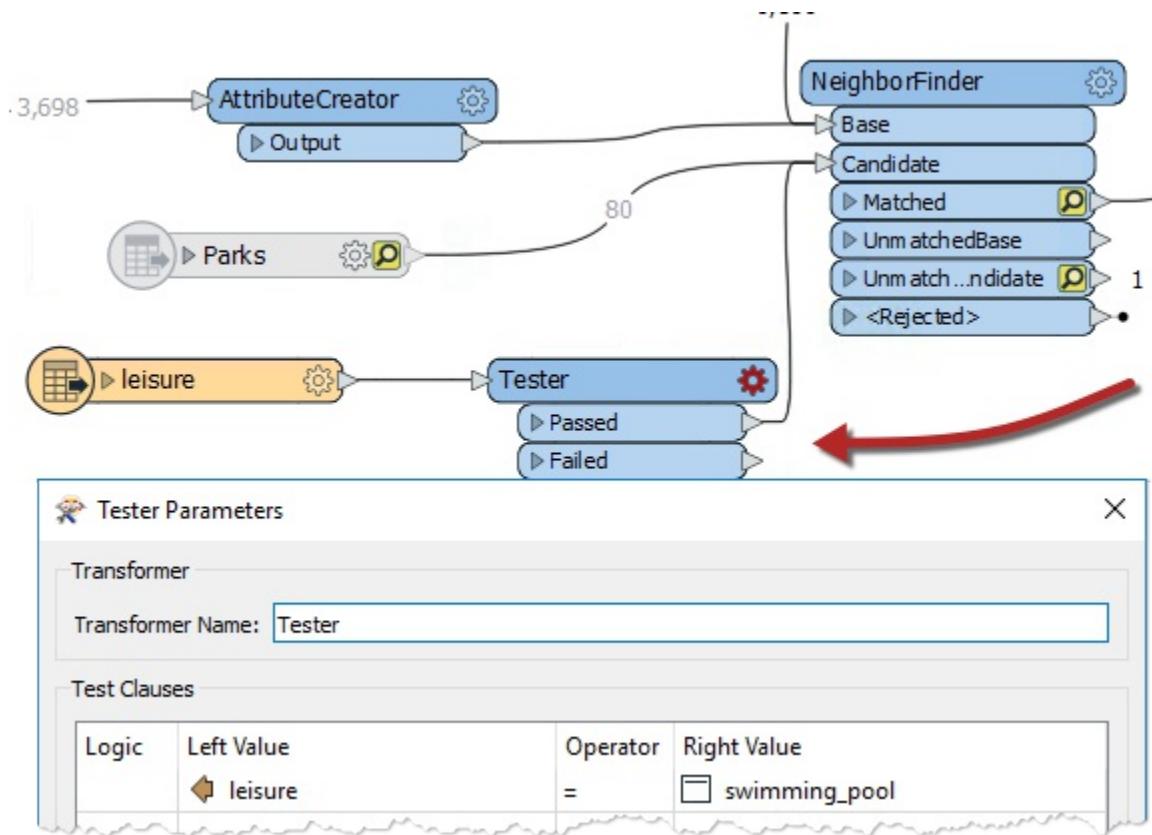


Then move the new leisure reader near the Parks reader and connect it to the NeighborFinder:Candidate input port. Then right-click on the Parks reader and select disable.

### 7) Filter Leisure Data

If you inspect the leisure data, you'll notice that there are various types of leisure facility, the type being recorded in the *leisure* attribute.

So, add a Tester transformer between the leisure reader and the NeighborFinder. Set up the parameters to test for *leisure = swimming\_pool*



### 8) Update Transformer Parameters

Now update AttributeRenamer to be PoolDistance instead of ParkDistance. The renaming of this attribute will cause the ExpressionEvaluator to turn red.

To fix the ExpressionEvaluator, open the parameters and change `@Value(ParkDistance)` to `@Value(PoolDistance)` to take account of the new PoolDistance attribute:

```
@Value(PoolDistance) + @Value(CrimeValue) - @Value(NoiseZoneScore)
```

Re-run the workspace. Check the log for warnings and errors, and then inspect the ExpressionEvaluator cache.

Notice that the walkability scores are exceedingly large all of a sudden, due to the PoolDistance. Something is wrong, but what?

### 9) Locate Problem

The PoolDistance is the source of the problem. There is no related log message to give a clue, and the Feature Count numbers look correct.

Let's inspect the data. Click on the leisure reader and while holding the shift key, click on the NeighborFinder. Then right-click on either object and select Inspect Cached Features. This will open all the selected caches in Visual Preview.

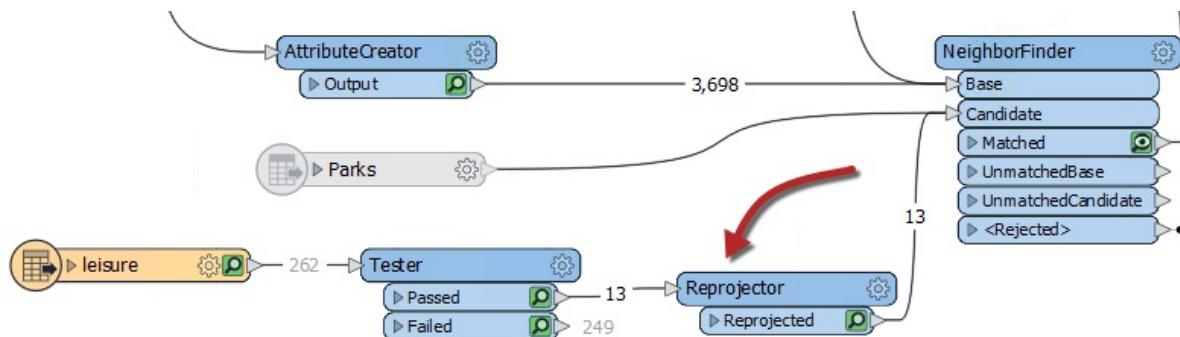
Right-click in the Graphics view, go to Background Map and ensure Background map off is selected. Visual Preview shows two specks of data, a long distance apart. This result is typical of a mismatch of coordinate systems.

Click on some features and select the Feature Information button. In this window you will see that the main data has a coordinate system of UTM83-10, while the leisure data from OSM has a coordinate system of LL84.

This disparity is why the "nearest" pool to each address is such a high distance.

### 10) Fix Coordinate System Problem

The obvious solution is to reproject the pools to the correct coordinate system. So, add a Reprojector transformer to reproject the leisure data before it gets to the NeighborFinder:



Inspect its parameters and set it up to reproject from LL84 to UTM83-10.

Re-run the appropriate parts of the workspace. Check the log window and inspect the ExpressionEvaluator cache.

Each address now has a walkability score account for pools instead of parks, with a lower number being better and a higher number worse.

## CONGRATULATIONS

By completing this exercise you have learned how to:

- use the ExpressionEvaluator transformer
- Check the log window for errors and warnings
- Locate problems through use of Feature Counts and Visual Preview
- Identify and fix problems in a workspace

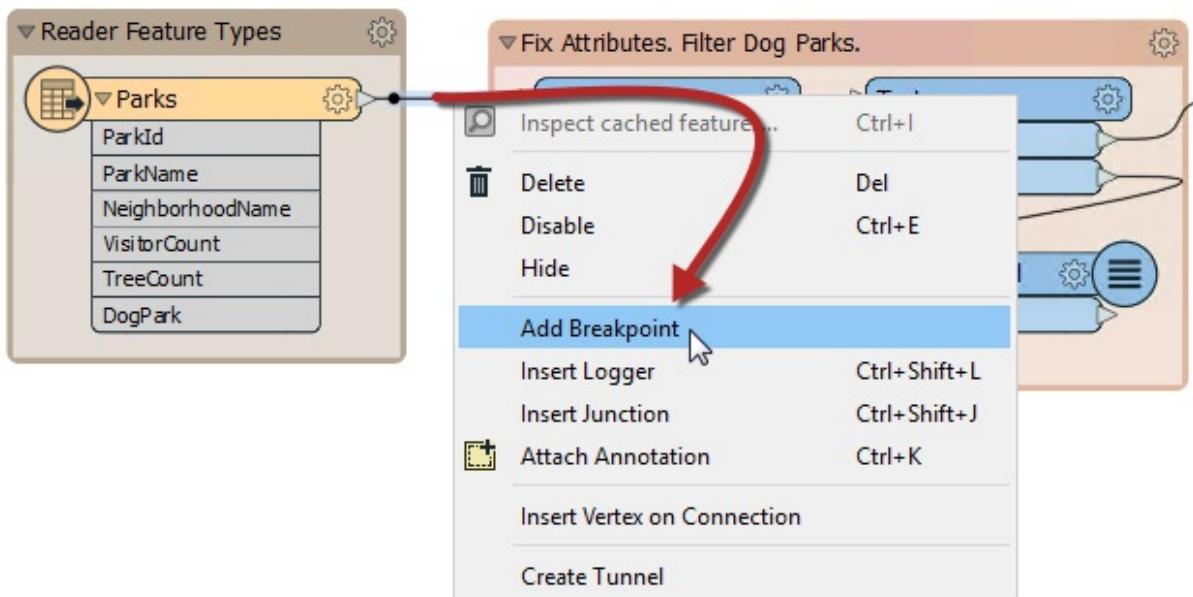


## Feature Debugging

Feature Debugging is a tool that allows individual features to be inspected during a translation. It differs from inspecting data at a particular location in that it inspects features one at a time, and allows the author to trace that feature's progress through a workspace.

This is most useful when a problem has been identified as being during transformation, but the point of failure is unknown.

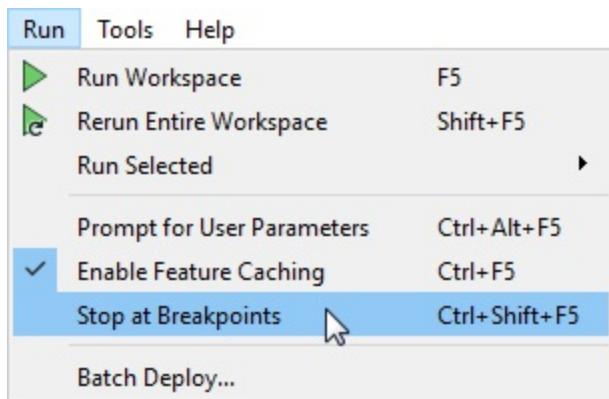
Feature Debugging is triggered by "breakpoints"; workspace connections that are flagged by the user as a location where features should be inspected:



The connection is highlighted in a darker black color with a red "stop" sign, to denote its new status:

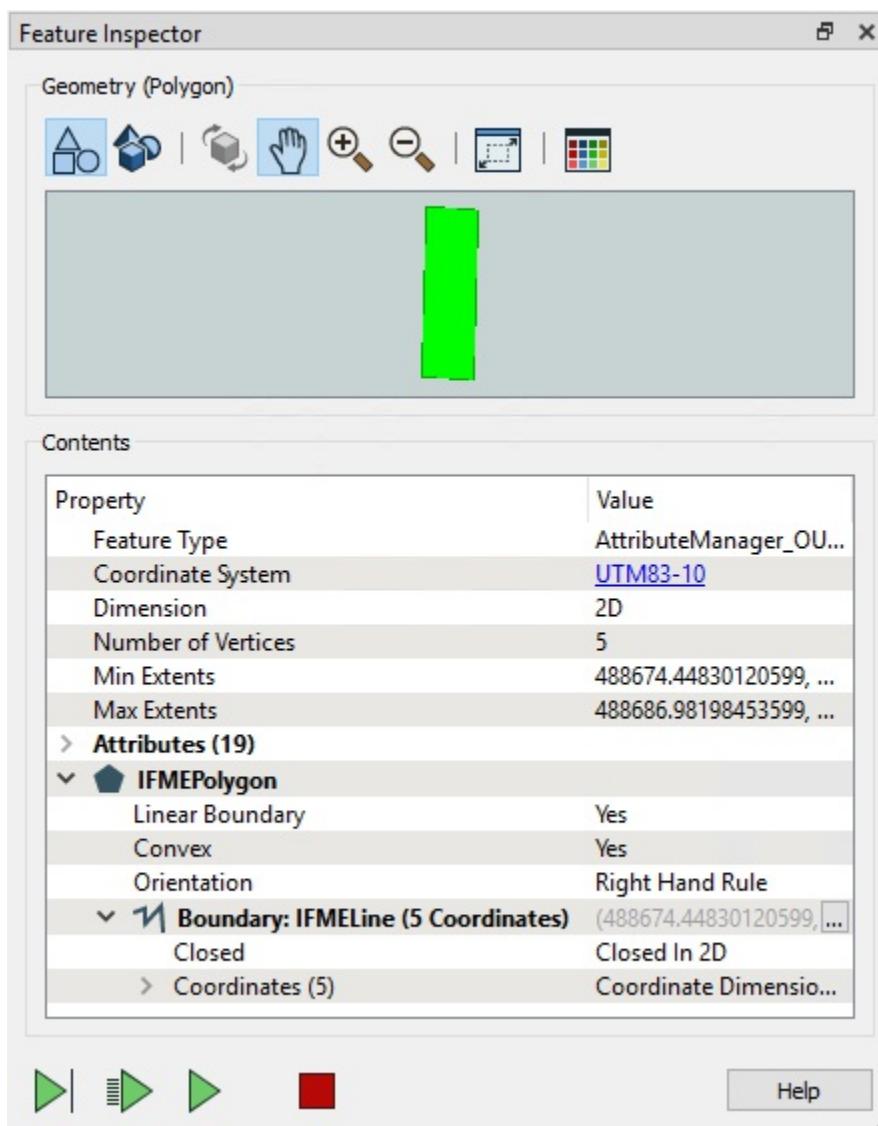


Now turn the "Stop at Breakpoints" on Run > Stop at Breakpoints and then run the translation:



When the first feature arrives at the breakpoint, the translation is temporarily paused and information about the feature displayed in a Feature Inspector window.

The upper section of the window shows a graphic representation of the feature; the lower section lists properties such as Feature Type and Coordinate System; plus attribute and geometry information.



There are four buttons at the foot of the Feature Inspector window:

Button	Operation	Description

	Step to Next Connection	This tool steps through the workspace one feature at a time, showing the status of a feature as it is processed.
	Step to Next Breakpoint	This tool re-starts the translation, stopping the next time a feature reaches an inspection point.
	Continue Translation	This tool re-starts the translation, ignoring all further breakpoints.
	Stop Translation	This tool stops the translation.

The currently active connection is highlighted red to show it is the location where the translation is currently paused.

## Methodology

*“Bad methodology impacts style. A poor style can be indicative of poor methodology.”*

Methodology is a less-obvious component of Best Practice than style. It manifests itself in two key ways:

- Maintenance
- Performance

In both cases, the first step in implementing a good project is to learn what methodology is poor, and how to avoid it.

### Maintenance

Maintenance is the ability to update the workspace and scale it up to a larger solution.

Maintenance relates to the concept of **design patterns**. A design pattern is the optimum layout of transformers to carry out a given task. A poor design pattern leads to complications when maintaining or scaling a workspace.

Poor design can often be detected in the style of a workspace; so if a workspace uses bookmarks and annotation correctly, but still doesn't look good, it may indicate flaws in the pattern of transformers.

### Performance

Performance is the efficiency of the workspace in carrying out the translation using the fewest resources.

Performance is important to most users but difficult to quantify. If a workspace runs to completion in an adequate time, then performance is often ignored. Any flaws in performance then manifest themselves when an attempt is made to increase the scale of the project to handle more data.

Poor performance surprisingly also manifests itself in the style of a project, as well as in the design patterns used.

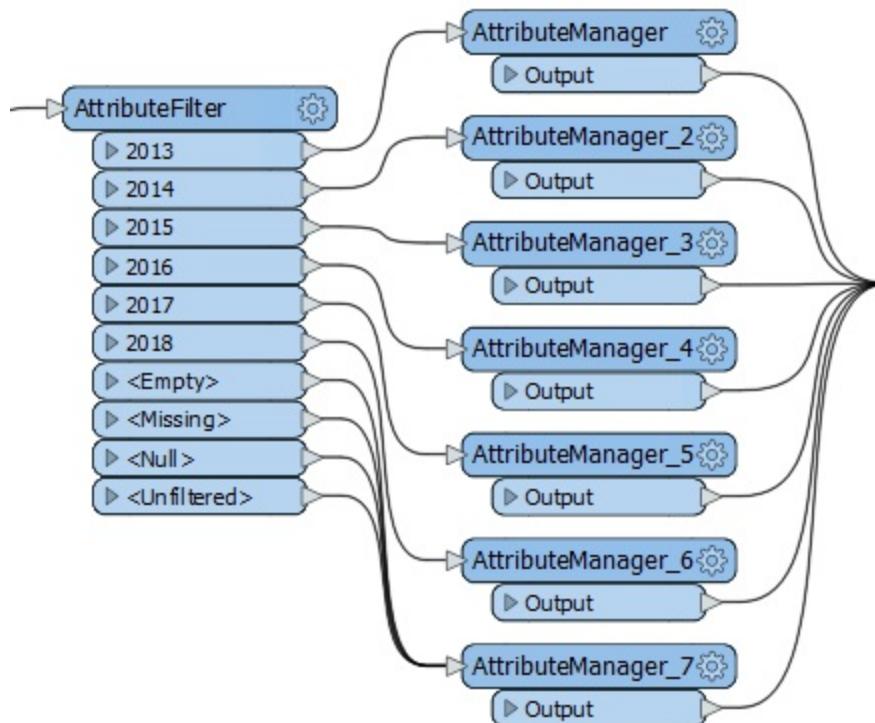
## Maintenance Methodology

Maintenance methodology is weak when a workspace's design doesn't allow easy updates or expansion. Some key indicators indicate design weakness.

### Duplicated Transformers

Duplicating the same transformer again and again – maybe creating multiple streams to do so – indicates a weak design.

For example, the multiple AttributeManager transformers here will cause maintenance problems:



The first problem is that each additional year to be supported requires a new transformer to handle it. The second problem is that a small change in the AttributeManager requires each transformer to be individually edited.

We can also say that this layout results in a style that is not as compact as it should be.

The solution here is to replace all of the above with a single AttributeValueMapper transformer (or possibly a SchemaMapper) to allow scaling with the minimum number of edits.

The general solution is to watch for instances of multiple transformers and try to assess whether there is a better design.

### Complexity

A workspace design is weak when it is more complex than necessary. Complexity means a solution becomes harder to maintain and harder to scale, especially when carried out by someone other than the original author.

There are various types of complexity to watch for:

- **Excess Scripting:** The use of Python scripting when an equivalent transformer already exists
- **Low-Level Complexity:** The use of FME functions and factories when an equivalent transformer already exists
- **Multiple Workspaces:** When multiple workspaces are chained in a way that is unnecessary
- **Workspace Style:** When workspace style is so convoluted it is difficult to understand

Before putting a project into production, it is worth asking a colleague to evaluate it for clarity. Sections of the workspace that they cannot easily understand, are points that future authors will also find difficult to maintain.

## TIP

*For a new FME user with strong scripting skills, Python transformers provide particularly tempting shortcuts. However, other workspace authors may not be as skilled in Python, and find it difficult to debug without a full Python development environment.*

## Database Connections

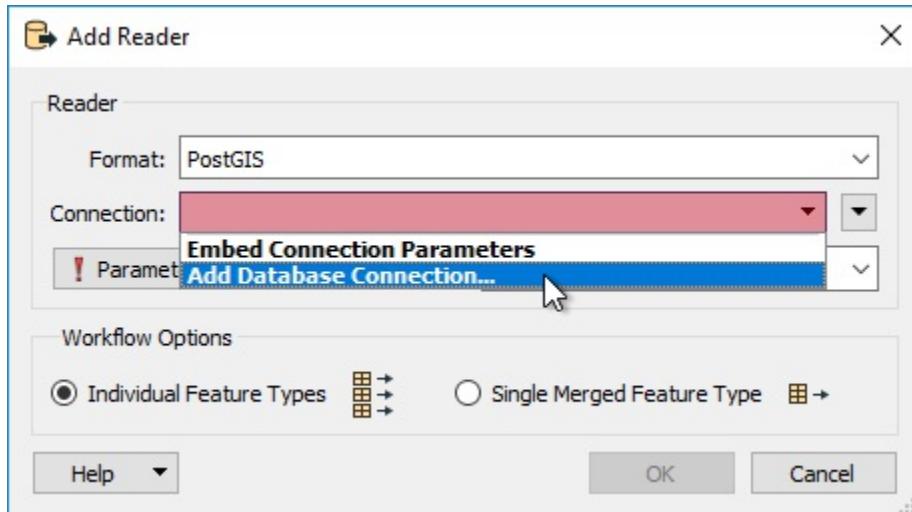
Database formats require connection parameters. Sometimes these parameters need to be applied multiple times in the same workspace, and sometimes the parameters need to be changed when switching platforms (for example from testing to a live environment).

It is possible to embed database connection values inside a workspace, adding that information wherever it is required. However, a better solution involves a tool in FME called Database Connections.

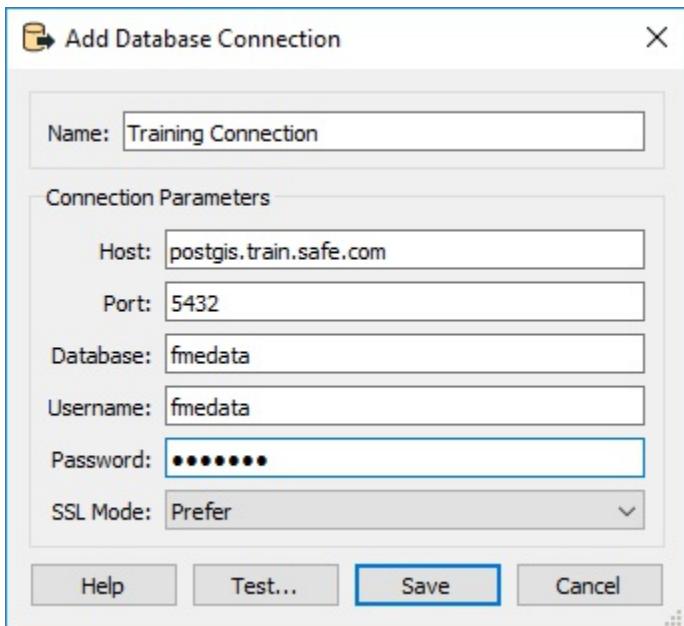
Database Connections are a pre-defined set of connection and authentication parameters, stored under a single name. Once created, a connection name is used instead of the actual parameters.

### Creating a Database Connection

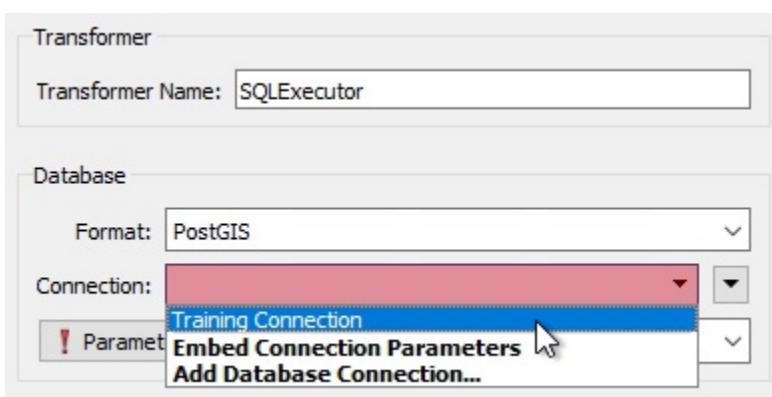
When using a database format and prompted for connection details, choose the option to add a connection:



A new connection is defined by selecting the database type, entering connection parameters, and giving the connection a name:



Now, whenever a database connection is required (here in an SQLExecutor transformer) the pre-defined connection is selected:



Now the repeated use of a database does not require repeated entry of the same credentials, and if the credentials change only one connection needs to be edited. Both of these features benefit the maintenance and scaling of a workspace.

Another benefit is that of security. Embedded connection information is stored inside the workspace file, posing a potential security risk. Database Connections are stored securely, outside of the workspace, and can not easily be accessed by someone else. If the workspace is passed to another user, they will have to set up their connection with their own parameters.

### FME Lizard says...

*It's better to be safe than sorry. You don't want to get burned by bad designs. If you're unsure about a workspace, consult other FME users, or contact the Safe Software support team for advice.*

*Also, be sure to check out this blog post on [FME and Code Smells](#).*



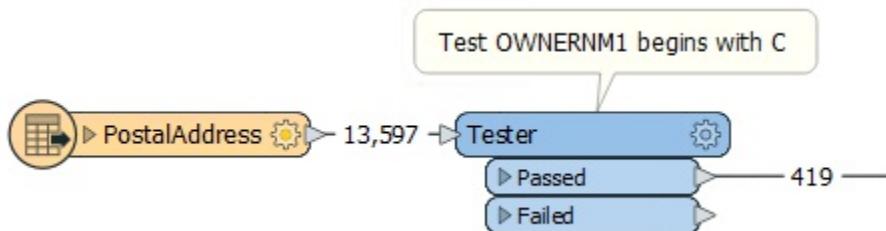
## Performance Methodology

Performance methodology is weak when a workspace's design causes the workspace to use more system resources (CPU and memory) than necessary. Like maintenance methodology, some key indicators indicate design weakness.

Performance methodology means setting up a workspace to run proficiently.

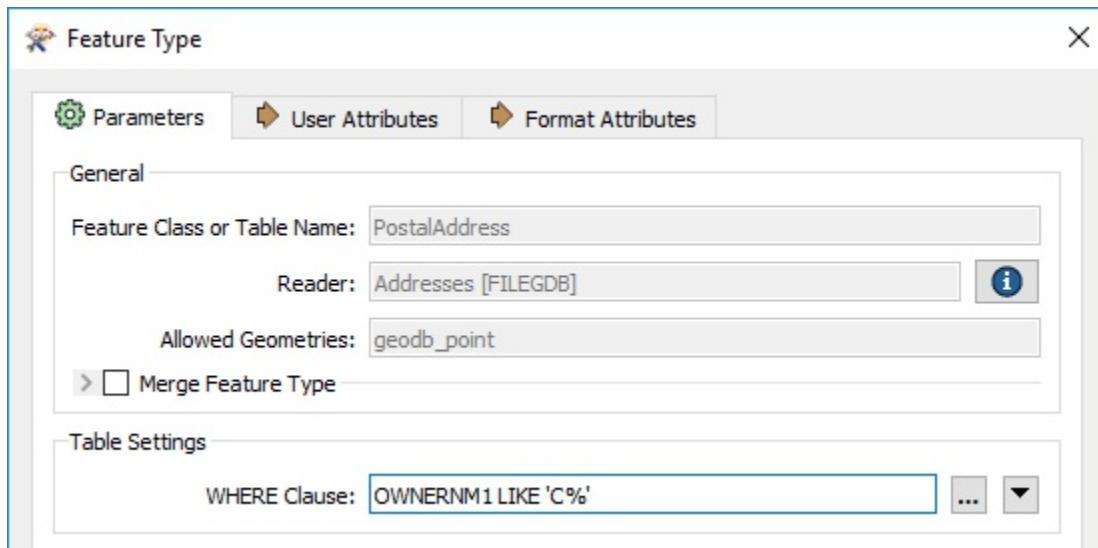
### Filtering Input

A common scenario in FME is to read data into a workspace and then filter out features (records) that are not required:



However, when data is read and immediately discarded, the resources used to read that data are a direct loss of performance.

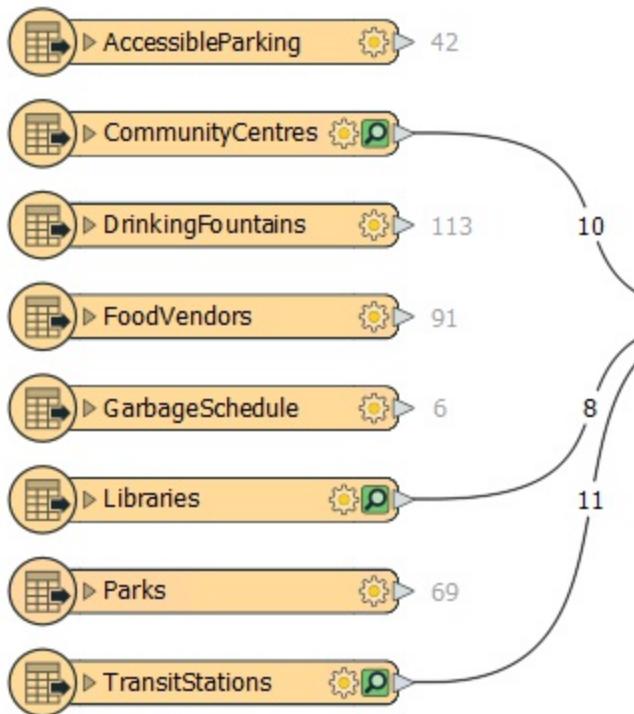
If data is filtered as it is read - rather than afterward - performance is much better, and many formats have parameters to do just that:



Here a 'WHERE Clause' parameter applies the required filter directly to the Geodatabase reader. Only data that matches the where clause is read and enters the workspace.

### Excess Feature Types

The schema of a source dataset is represented on the FME canvas by feature type objects:



Not connecting a feature type to other objects in your workspace is equivalent to reading and discarding data and is likewise detrimental to performance.

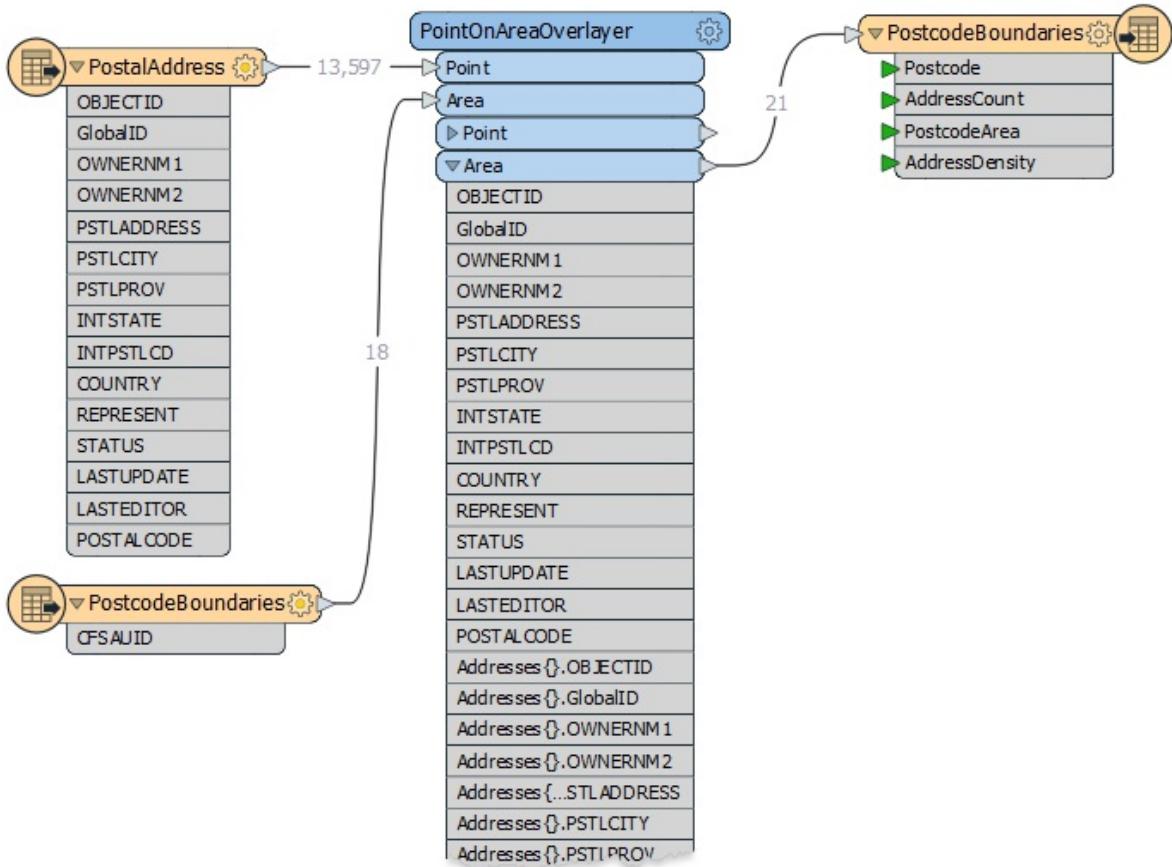
When adding readers, FME prompts the user to select feature types to add to the translation. You should avoid adding feature types you don't need and remove ones that are already added but not connected.

### FME Lizard says...

*Not only do excess feature types slow down your work, they clutter the canvas and make it harder to keep a clean and tidy style.*

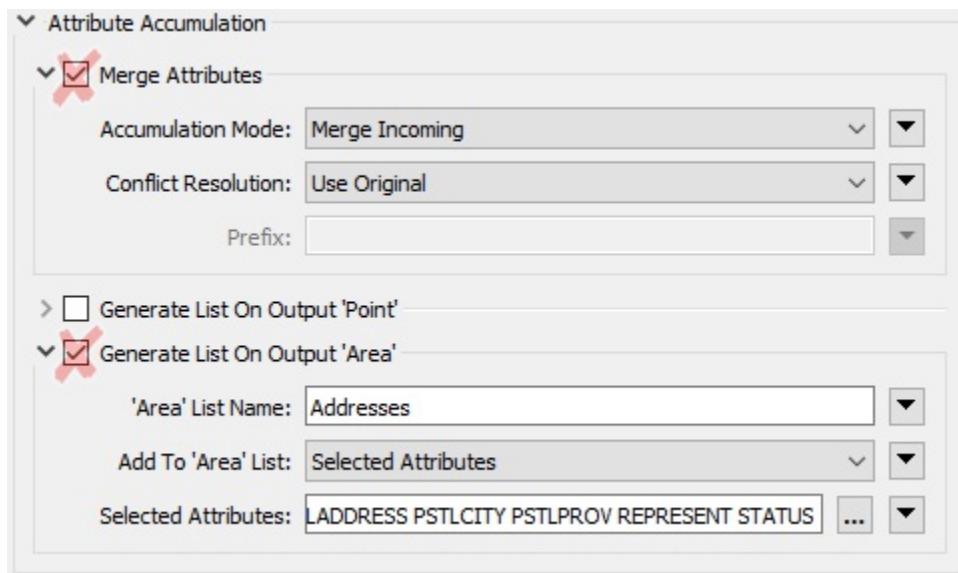
### Excess Attributes and Lists

Once data has been read into a workspace, it still can be reduced in size to assist performance. For example, attributes not defined in the output schema are not necessary to a workspace and can be removed:



Here a workspace author is calculating the number of addresses in each postcode (CFSUID) of a city. The address attributes are not required in the output schema but are copied on to the postcode features, along with a list of addresses, and everything carried through to the end of the workspace.

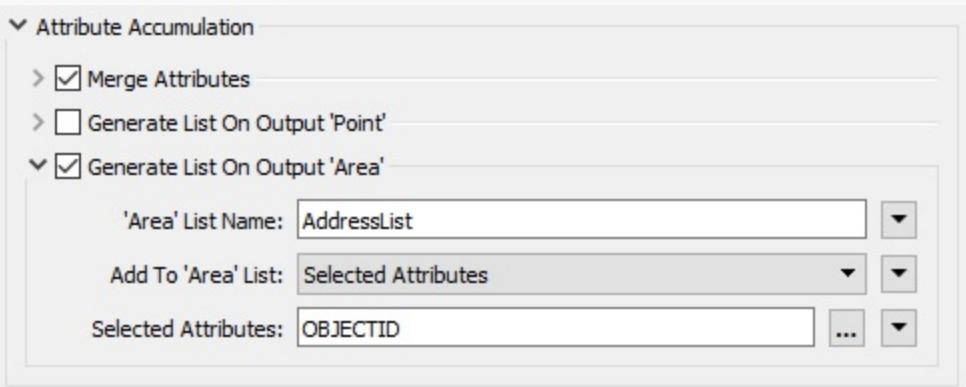
In this scenario the author should avoid the PointOnAreaOverlayer options for copying attributes and creating lists:



This step will reduce the amount of memory required to run the translation, with no effect on its output.

**FME Lizard says...**

*Lists are the worst attribute type to keep for no reason, since they can have multiple values for each record. Parameters in many join transformers allow the author to generate only the list attributes required:*



## Error Trapping

Sometimes scaling up/performance means using more datasets of varying types and quality. If data quality is not considered, future performance can be compromised.

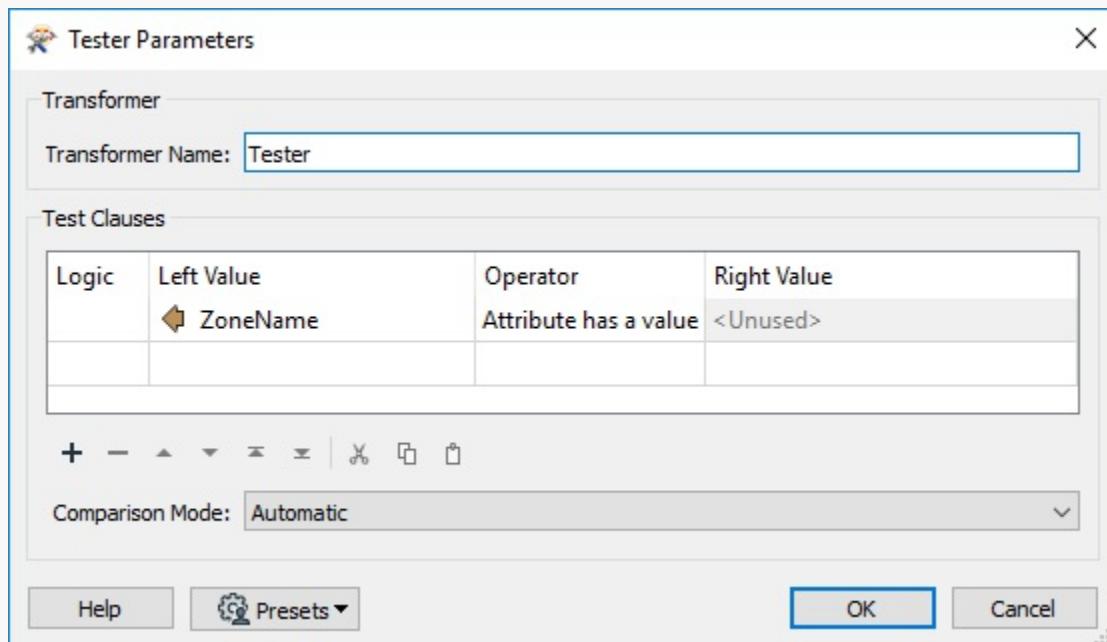
One way to design for future capabilities is by using error trapping.

Error trapping is a way to design a workspace such that unexpected data does not cause the workspace to fail. The author attempts to foresee data problems that might arise and build in methods to handle them.

Error trapping can be as simple as adding a test or filter transformer to weed out bad features, or it can be more complex and include ways to process data in different ways depending on the circumstances.

## TIP

*The Tester transformer has an operator for testing whether an attribute has a value:*



*This is very useful for error trapping, to test whether an attribute has a value before trying to use it as the source for a parameter.*

## Exercise 2

## Methodology

<b>Data</b>	Addresses (Esri Geodatabase) Crime Data (CSV - Comma Separated Value) Parks (MapInfo TAB) Swimming Pools (OSM - OpenStreetMap)
<b>Overall Goal</b>	Work on Vancouver Walkability Project
<b>Demonstrates</b>	Methodology Best Practice
<b>Start Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\BestPractice-Ex2-Begin.fmw
<b>End Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\BestPractice-Ex2-Complete.fmw

Continuing from the previous exercise, you have been assigned to a project to calculate the "walkability" of each address in the city of Vancouver. Walkability is a measure of how easy it is to access local facilities on foot. The initial workspace analyzed crime in the area and distance to the nearest park. Then you were asked to calculate the distance to the nearest swimming pool instead of parks.

In this exercise, we will modify our workspace to improve its performance.

### 1) Continue Workspace

Start FME Workbench and open the workspace from the previous exercise. Alternatively, you can open C:\FMEData2019\Workspaces\DesktopBasic\BestPractice-Ex2-Begin.fmw. Continuing in the workspace from Exercise 1, we will now try and make this workspace run faster.

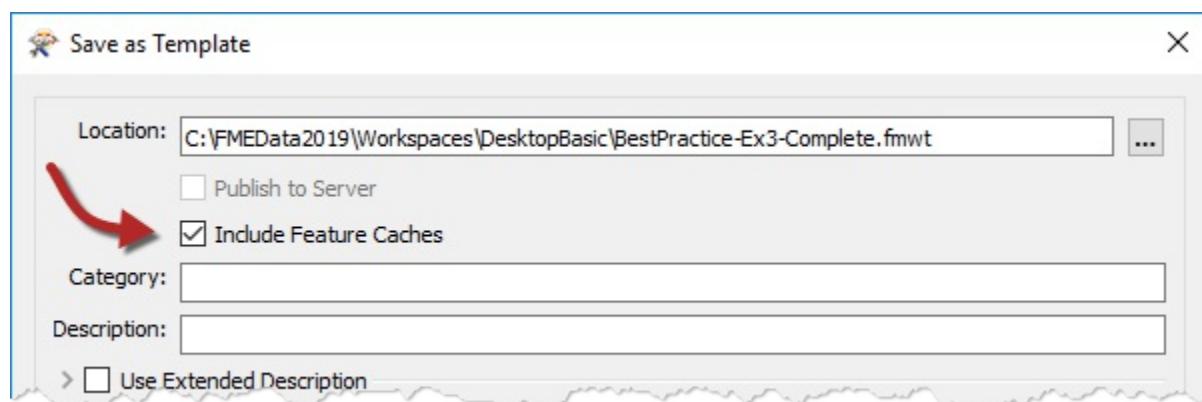
**Note:** If you open the starting workspace, you will need to run it to create the caches.

**Note:** Remember, if you clicked the AutoLayout button in the first exercise, your workspace will look different. Pay close attention to transformer and port names.

### 2) Determine Performance Improvements

While editing the ExpressionEvaluator in the previous exercise, you might have noticed there was a lot of additional attributes like CrimeList{}.City or CrimeList{}.Block. These excess attributes clutter the display and inspecting the output becomes hard. These attributes can hardly be helping the performance of the workspace either - even if that's mitigated by using caches during development.

Let's save the workspace as a template file. In the top menu go to File > Save As Template. When prompted, be sure to have the Include Feature Caches option checked:



Now if we come back to this project later or share it, the user can reopen the template and have all our cached data ready for use.

Check the size of the template file BestPractice-Ex2-Begin.fmw that you just created. You'll see that it is almost 50mb in size, which is fairly large for a template. It's not a problem to have a large template file, but it does indicate a lot of data is being cached and that this could affect the workspace's performance.

One aspect of data is the number of attributes and lists. Since there are a lot of additional attributes to remove but only a few we need to keep, we will use the AttributeKeeper transformer. Place the AttributeKeeper between the AttributeValueMapper\_2 and the ExpressionEvaluator transformers:



Inspect the AttributeKeeper parameters and set them up to keep only CrimeValue, NoiseZoneScore, and PoolDistance. Take note of the names of the attributes that we are not keeping. We might be able to remove them earlier in the workspace.

### 3) Remove Lists

One attribute of interest is a list attribute called CrimeList{}, which doesn't appear necessary for any part of this translation. Track down its source by pressing **Ctrl+F** and search for CrimeList. The search results show up in the Navigator window, and there you will find the Aggregator transformer is creating CrimeList.

Check the parameters for the Aggregator transformer and turn off the Generate List parameter, to prevent the list from being created. This step will cause many caches to become stale, but we will re-run the workspace shortly to solve this.

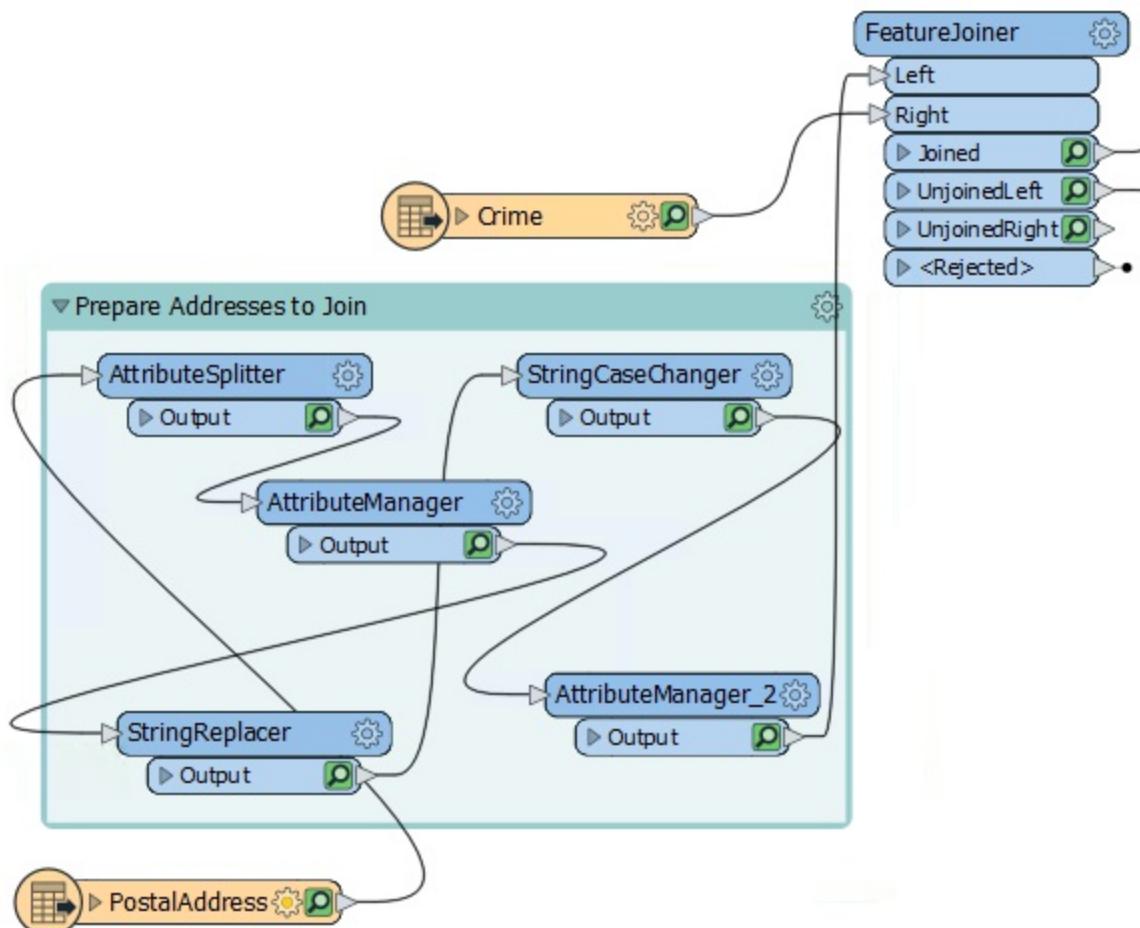
### 4) Remove Extra Feature Types

Another reason a workspace is running slowly is if you are reading in extra data that is not being used in the workspace. It looks like the original author read in the PostcodeBoundaries feature type from the Addresses.gdb. Additionally, we didn't remove the Parks feature type once we were done with it. Delete both of those now and click Yes on any warnings that pop up.

### 5) Collapse the Bookmark

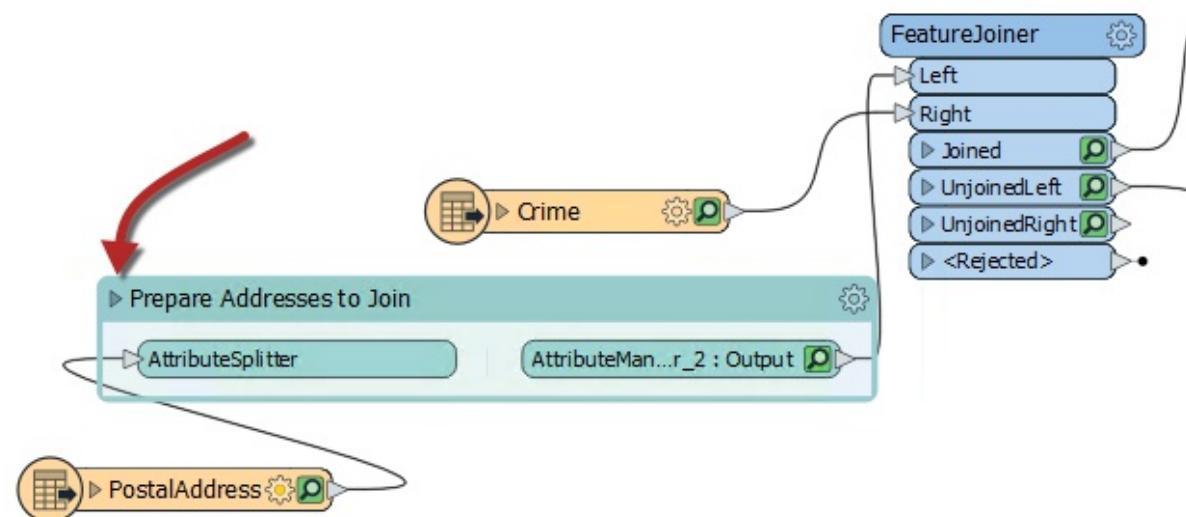
Another source of excess caching are transformers producing output that we don't need to inspect. These can be prevented by hiding these transformers within a collapsed bookmark.

Add a bookmark around all of the transformers between the PostalAddress reader and the FeatureJoiner, by selecting all the transformers and pressing **ctrl+B** on your keyboard. Then name the bookmark Prepare Addresses to Join:



**Note:** Bookmarks will be covered in greater detail later on in this chapter.

Now we can collapse the bookmark and then when we re-run the translation only the last transformer will have a cache. To collapse the bookmark, click on the arrow beside the bookmark name:



## FME Lizard Says...

If you want to work ahead, add a couple more bookmarks around other sections and then collapse them. We will be doing this in the next exercise, so you can see if your bookmarks are the same.

---

---

## 6) Run the Workspace

Now run the workspace by clicking on the ExpressionEvaluator and choosing *Run to This* or just click the run button.

The workspace will run and data will be cached, but for the collapsed bookmark, only one cache will be created for its five transformers. Attributes unnecessary to the output will also be removed by the AttributeKeeper.

Save the workspace as a new template and check the option to include caches. Check the file size of the new template. It should be considerably smaller (around 16mb).

**Note:** If you want to use the workspaces provided, just open them and save them as a template file to use for your comparison

---

## CONGRATULATIONS

By completing this exercise you have learned how to:

- Remove unnecessary attributes to improve performance
- Track down unnecessary lists and remove them
- Improve performance by collapsing bookmarks to prevent excess caching
- Save a workspace as a template, including caches

## Style

*"A good looking, well-organized workspace gives the customer the feeling that you have done quality work."*

Style is perhaps the most obvious component of FME Best Practice. You can tell at a glance when a workspace is well-styled and when it is not. As the quote above implies, a well-designed workspace demonstrates competence.

But style is more than just looks; a properly designed workspace provides many benefits as it is further developed and edited in the future.

## An FME Workspace Style Guide

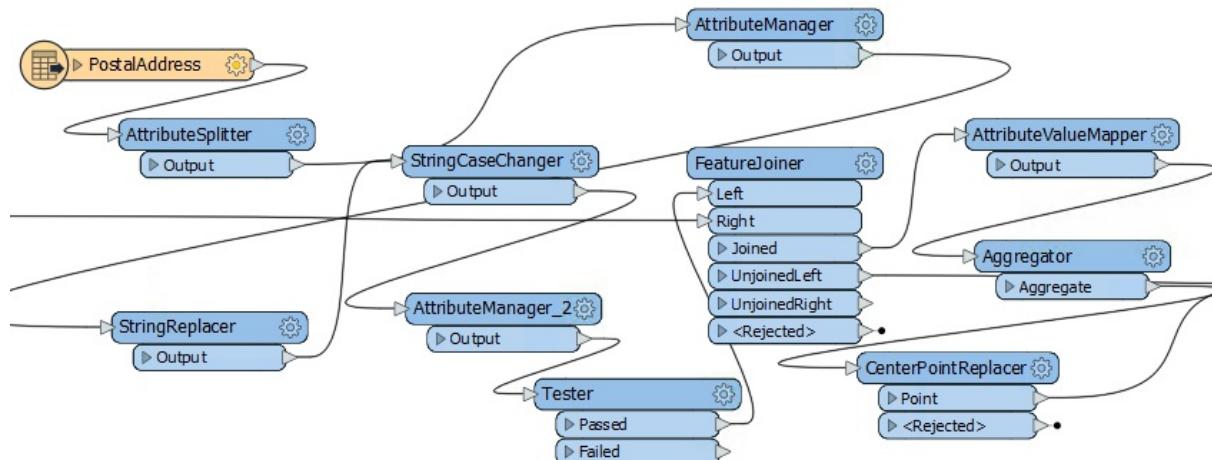
A good style of design makes it easier to navigate and understand an existing workspace. This is important when workspaces might need to be edited by other users, or when you intend to make edits yourself at a later date.

Specifically, a good style can help a user to...

- Distinctly define different sections or components of a workspace
- Quickly navigate to a specified section or particular transformer
- Pass a workspace on to another user for editing
- Rename workspaces and content with a more explanatory title

## Example of Poor Design

Do you need proof? Well, would you want to be given the task of editing this workspace? Can you even tell what this section does or - more importantly - why?



### FME Lizard says...

*As I always say, size doesn't matter! You should always use Best Practice, whether it's a small workspace or training exercise, or a large-scale project. Getting into the habit helps make your smaller projects scalable.*

*If you don't design a workspace well from the very start, it will just become harder and harder to make edits as you work on it.*



## Annotating Workspaces

Annotation is a key method for a clear and comprehensible design.

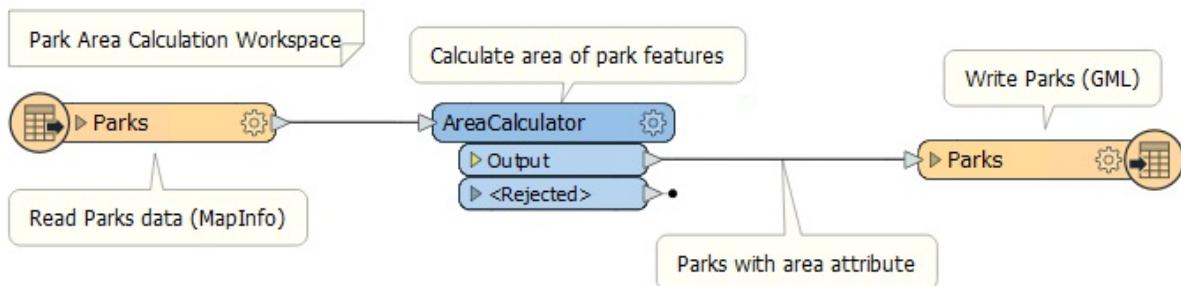
Annotation helps other users understand what is supposed to be happening in the translation and also helps the creator when returning to a workspace after a long interval (take it from me that this is especially important!)

Two different types of annotation can be applied to a workspace.

---

### User Annotation

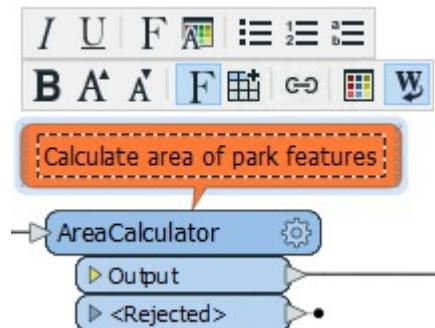
User annotation is a comment created by the user. It can be connected to a workspace object (transformer or feature type), can be connected to a workspace connection, or can float freely within the workspace.



To create attached user annotation, right-click a workspace object and select Add Annotation, or use the shortcut **Ctrl+K** when the object is selected.

To create floating user annotation, right-click the canvas and select Insert Annotation, or press **Ctrl+K** when nothing is selected.

When you place an annotation you have the opportunity to change the font style, font size, and background color; plus you can also add hyperlinks, bullet points, and tables.

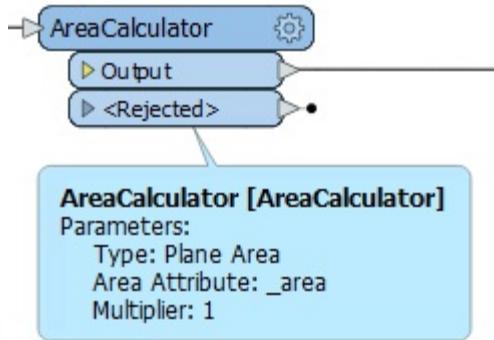


---

### Summary Annotation

Summary annotation is an FME-generated comment that provides information about any object within the workspace. This item can be a source or destination feature type, or a transformer.

Summary annotation is always colored blue to distinguish it from other annotation. It's always connected to the item to which it relates and cannot be detached.

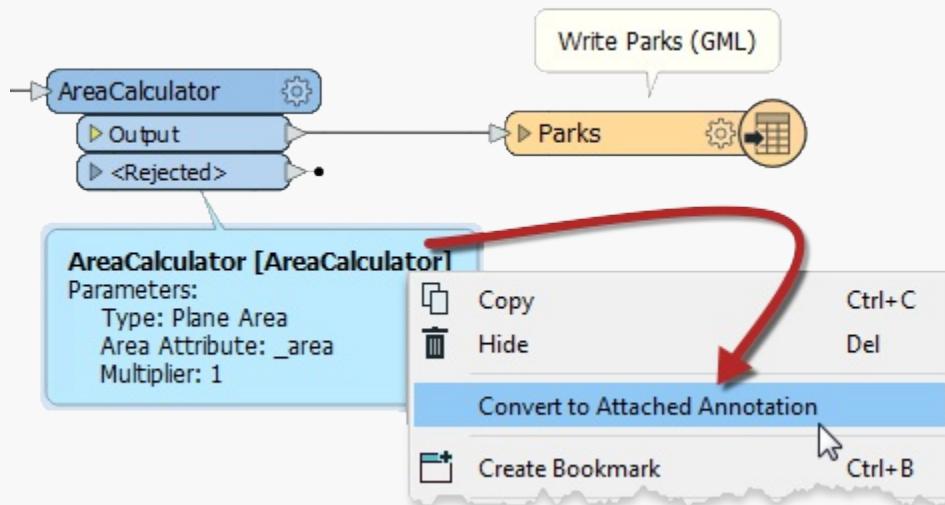


The nice thing about Summary Annotation is that it automatically updates in response to changes. That makes it very useful for checking transformer parameters (or reader/writer schemas) at a quick glance. It's particularly useful in situations where the parameters are set through a wizard and are more awkward to check (for example, the SchemaMapper or FMEServerJobSubmitter transformers).

## TIP

*A good idea is to use summary annotation to show **what** actions are taking place; but use user annotation to clarify **why** an action is being carried out.*

*You can convert a summary annotation to a user (attached) annotation by using this context menu option:*



*This allows you to extract the information from a summary annotation, but edit it as you would a user annotation. Note that a converted summary annotation no longer updates automatically!*

## Bookmarks

A bookmark, like its real-world namesake, is a means of putting a marker down for easy access.

With FME the bookmark covers an area of the workspace that is usually carrying out a specific task, so a user can pick it out of a broader set of transformers and move to it with relative ease.

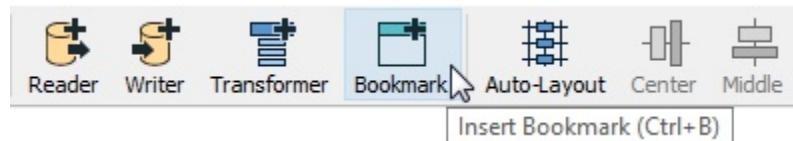
### Why use Bookmarks?

Bookmarks play an important role in a well-styled workspace for a number of reasons, including these.

- Design: As a way to subdivide a workspace and manage those sections
- Access: As a marker for quick access to a specific section of a workspace
- Editing: As a means to move groups of transformers at a time
- Performance: As a means to improve workspace performance when caching data

### Adding a Bookmark

To add a bookmark, click the Bookmark icon on the toolbar.



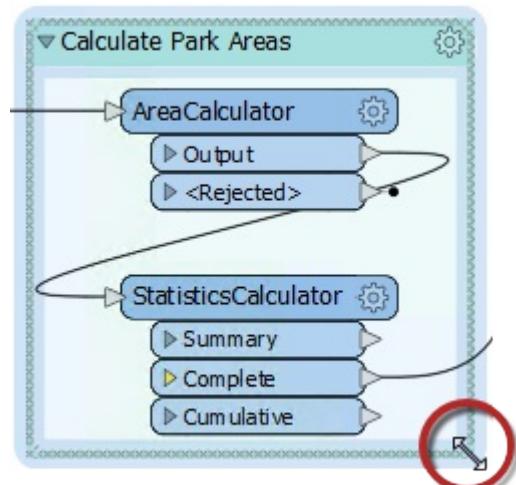
Whereas a traditional bookmark marks just a single page in a book, the FME bookmark can cover a wide area of the canvas. A single workspace can be divided into different sections by applying multiple bookmarks.

#### TIP

*If any objects on the workspace canvas are selected when a bookmark is created, the bookmark is automatically expanded to include those items.*

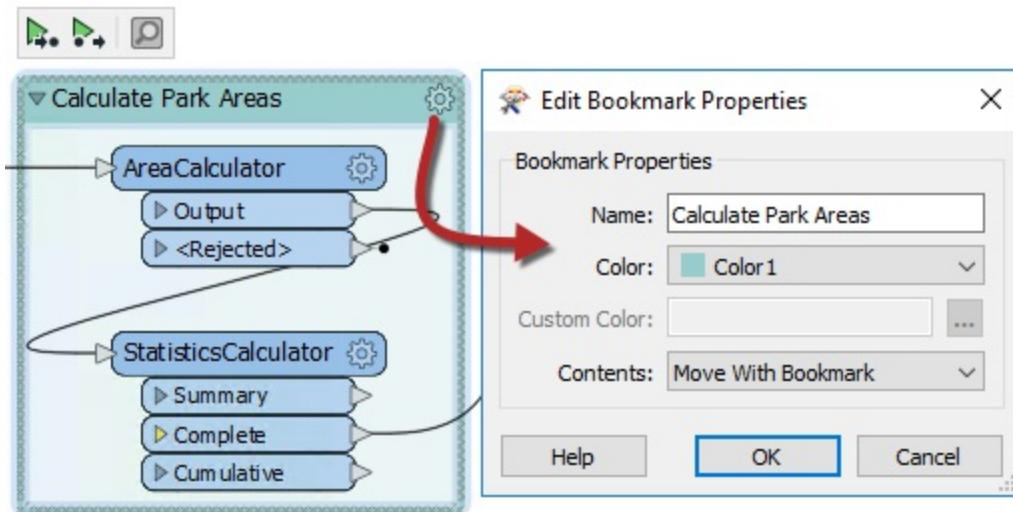
### Resizing and Editing a Bookmark

To resize a bookmark hover over a corner or edge and then drag the cursor to change the bookmark size or shape.



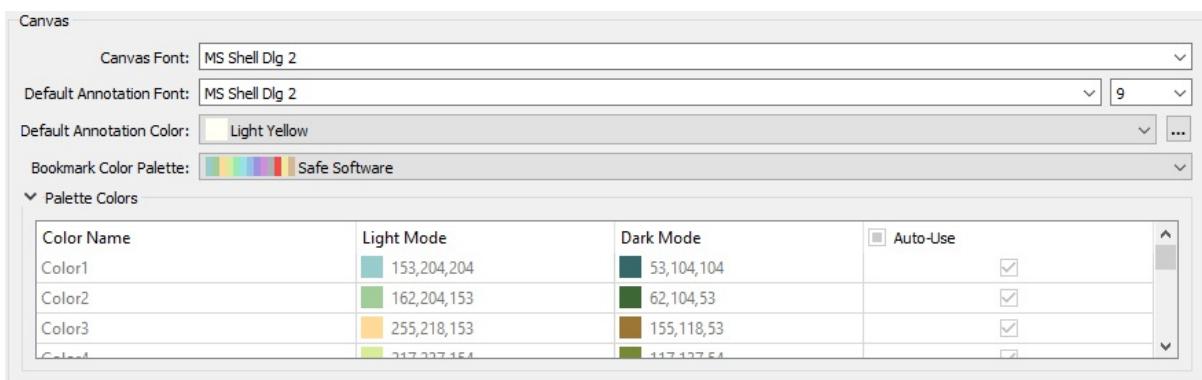
### Bookmark Properties

Click the cogwheel icon on a bookmark header to open the bookmark properties dialog:



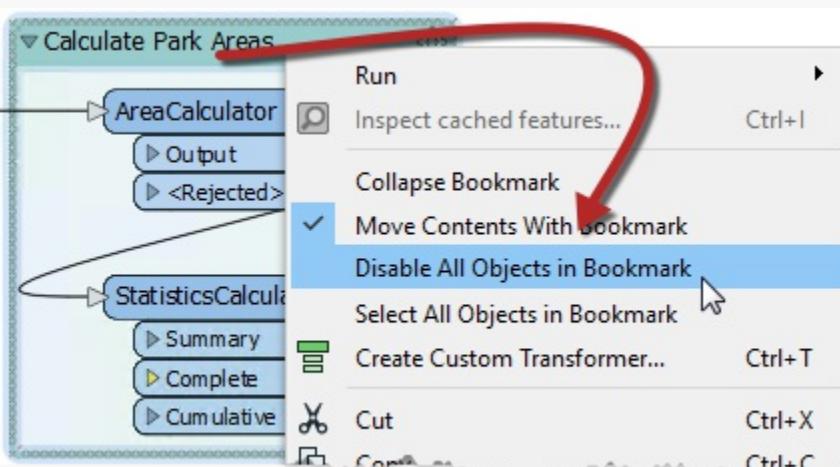
Here you can change both the name and color of the bookmark and decide about whether contents will move with it (more on that later).

The bookmark colors can be set to an existing color palette or custom colors can be used. Additionally, customized palettes can be created by going to Tools > FME Options... > Appearance:



## TIP

The context (right-click) menu for a bookmark reveals options to select all objects within the bookmark, or to disable all of those objects, making it useful for testing purposes:

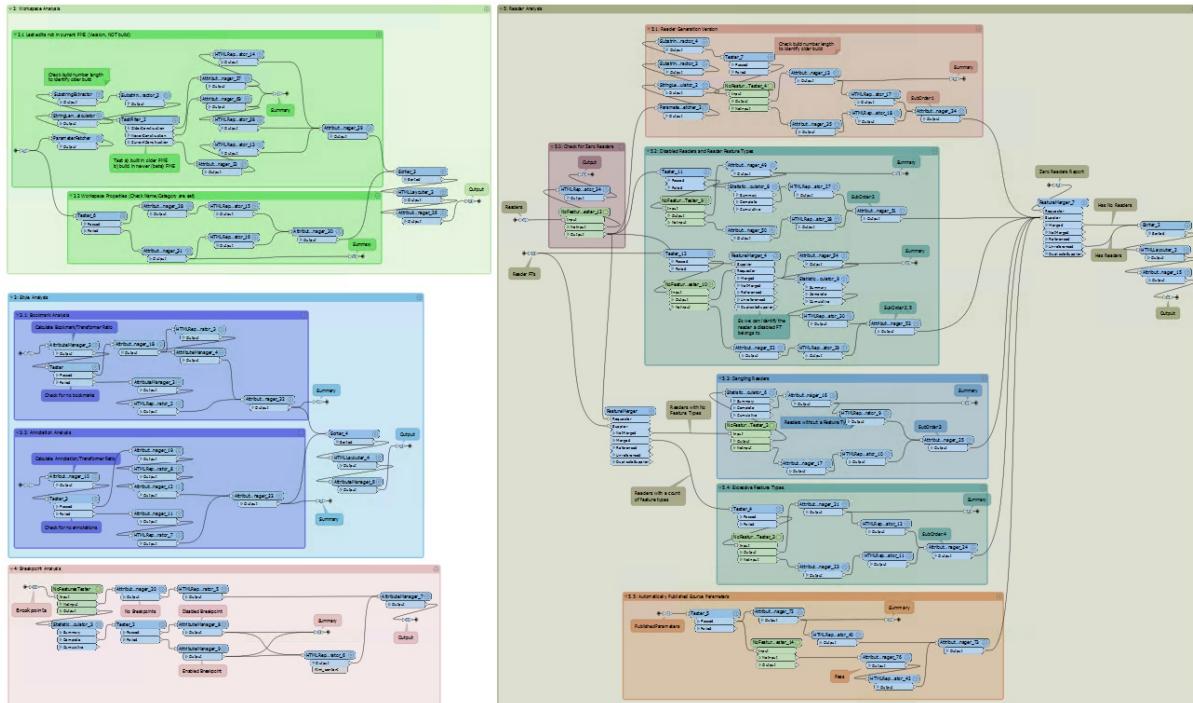




## Bookmarks for Design

A bookmark is a great way of indicating that a particular section of a workspace is for a particular purpose. By subdividing a workspace in this way, the layout is often a lot easier to follow.

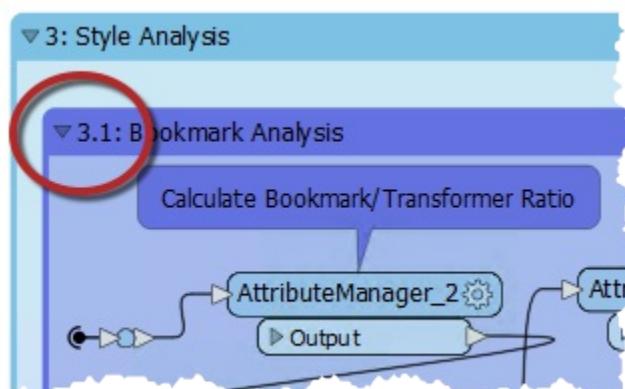
As one user has put it, bookmarks are like paragraphs for your workspace!



The above workspace illustrates how to mark up different sections of a workspace using bookmarks. As you can see, it's permitted to subdivide bookmarks further by *nesting* one bookmark inside another.

## Collapsible Bookmarks

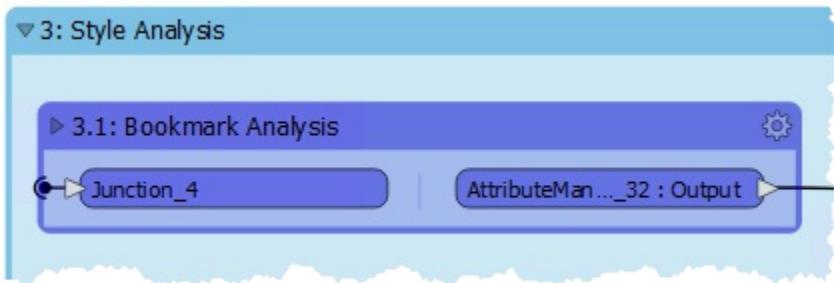
Each bookmark has a small icon in the top-left corner that allows it to be collapsed:



## FME Lizard says...

Did you notice the annotation in the image above? It is the same color as the bookmark. New for 2019, any annotations contained within a bookmark will take on the same color. This really helps to clean up the look of your workspace!

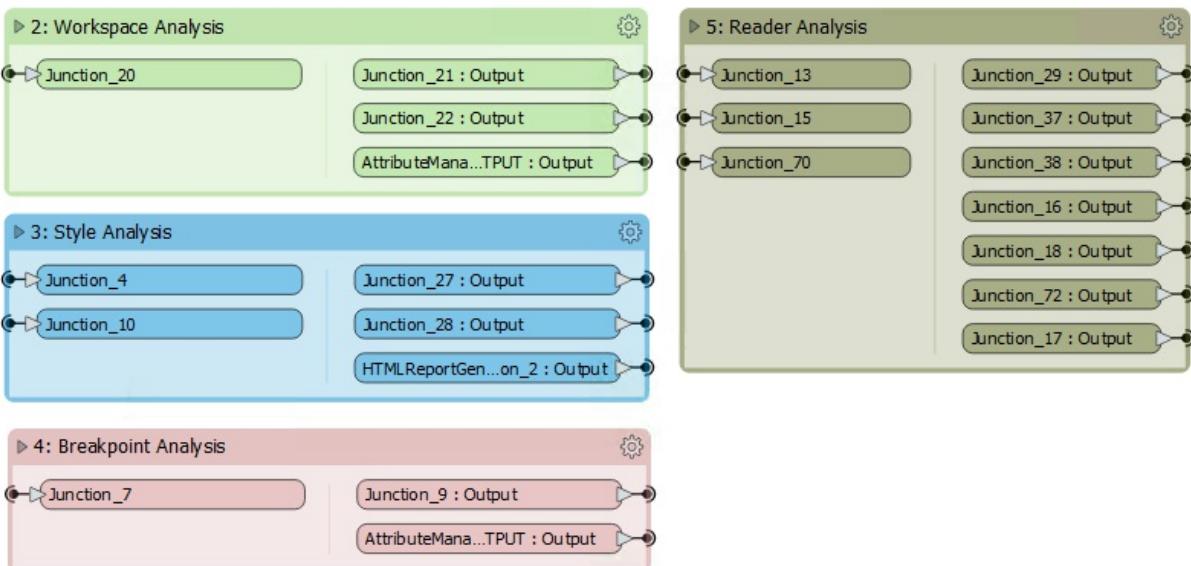
Collapsing a bookmark means it is compressed down to the size of a single transformer, displaying none of the contents except for where data enters or exits the bookmark:



Clicking the icon a second time re-opens the bookmark to its previous size.

This functionality allows large sections of workspace to be rendered in a much smaller area, and only opened up when editing is required.

For example, the section of the workspace displayed above might be reduced to this:

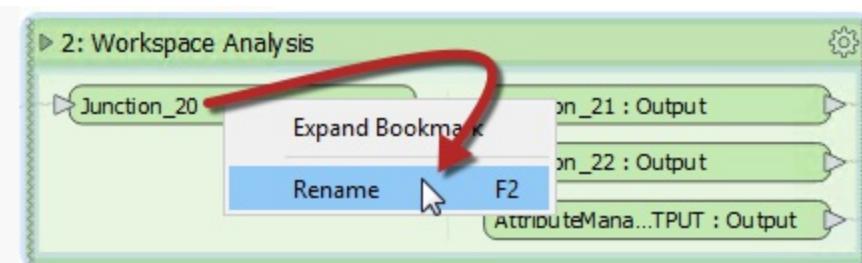


Re-opening a collapsed bookmark adjusts the layout of the workspace, moving other transformers or bookmarks out of the way so that its contents are shown without overlap. Re-closing the bookmark causes the opposite to occur.

For example, in the above screenshot if bookmark 3 (Style) is expanded, then bookmarks four and five are moved to one side to accommodate it. When bookmark three is collapsed again, the reverse takes place, to give the same compact layout as before.

## FME Lizard says...

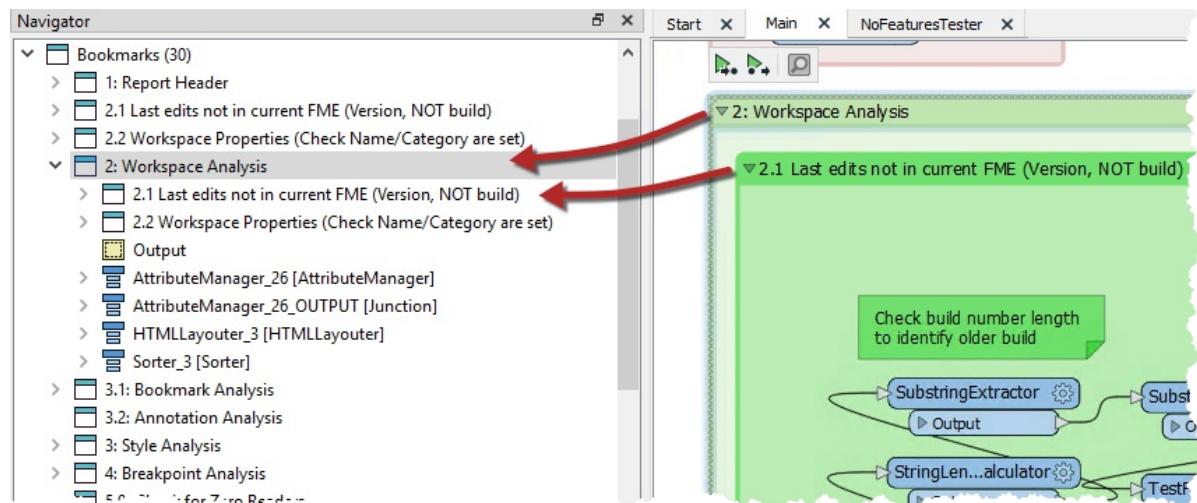
*The input and output ports on collapsed bookmarks can be renamed to help clarify the data entering and exiting:*



This can be achieved by either double-clicking the object, pressing F2, or using the Rename option on the context menu.

## Bookmarks for Quick Access

Bookmarks are listed in the Workbench Navigator window. Each bookmark is depicted as a folder and can be expanded to show its contents. It may include feature types, transformers, or other - nested - bookmarks:

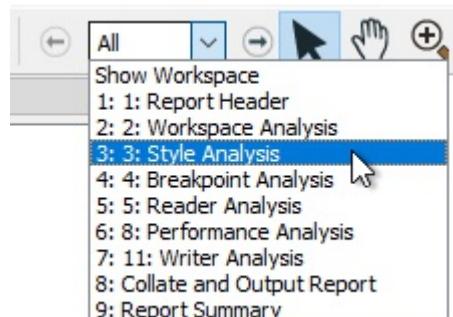


Clicking or double-clicking a bookmark in the Navigator selects that bookmark and brings it into view. So when bookmarks have been used to divide a workspace into sections, they can also be used to navigate between different parts of that workspace.

In this way, bookmarks are like the chapter headings in a book!

## Bookmark Navigator

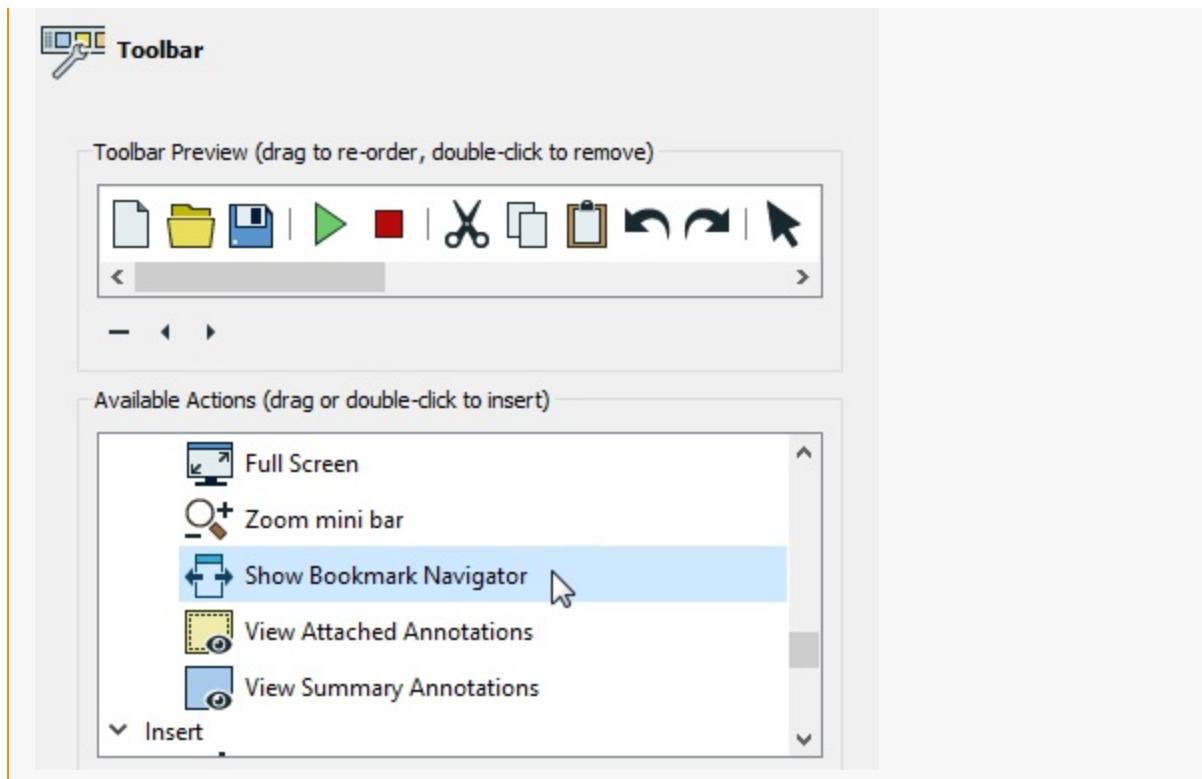
Bookmarks can also be navigated on the FME Workbench toolbar using the Bookmark Navigator:



Besides being a way to access bookmarks quickly, the Bookmark Navigator tool can be used to present your workspace. By clicking the arrow button (or pressing the keyboard spacebar), you flip from bookmark to bookmark using animation, in a way that would be very useful when showing the workspace as part of a presentation.

### TIP

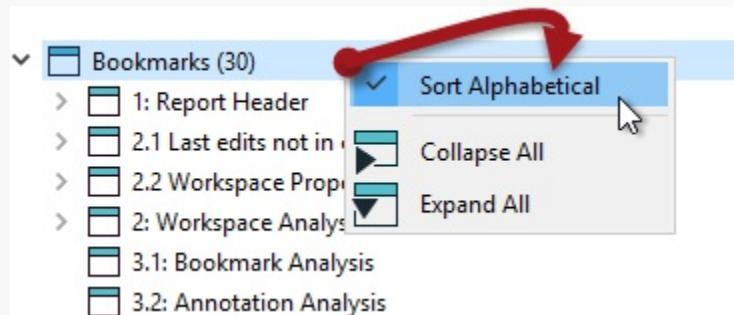
*To access the functionality you need to make sure it is added to the toolbar. You can do this by right-clicking on the toolbar and using the customize option.*



## FME Lizard says...

*The order of bookmarks in that window is alphabetical and that might not always be the same order that you wish to present a workspace.*

*In that case, right-click on Bookmarks in the Navigator window and turn off the default option to "Sort Alphabetically".*



*Bookmarks can then be dragged up and down in the Navigator window to give the correct order. Additionally, a new option on the bookmark Properties dialog allows you to exclude specific bookmarks from the Bookmark Navigator. Nested bookmarks are excluded by default, so must be turned on to be included in the navigator.*

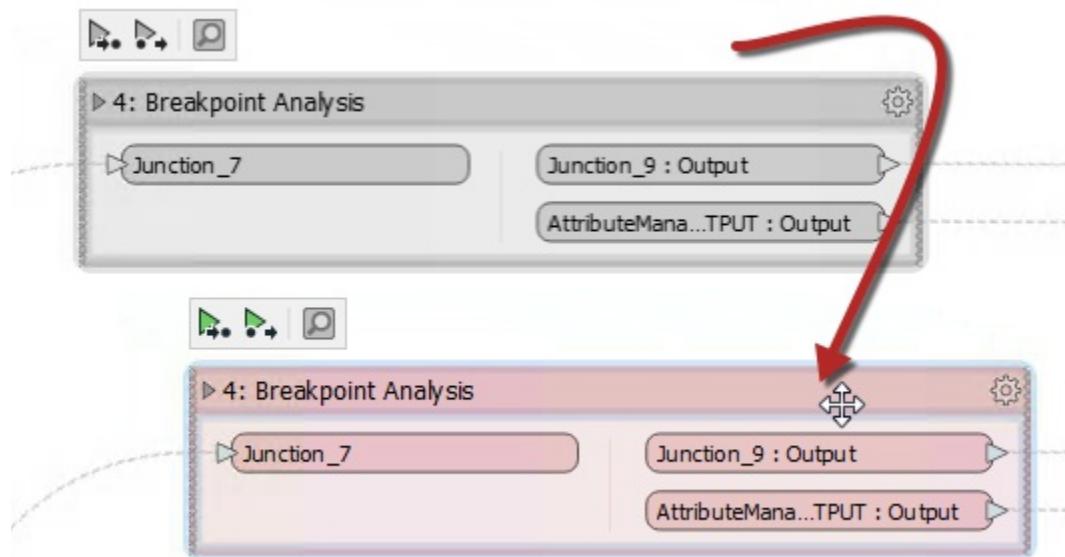
*For more information see this [article on bookmarks](#) on the Safe Software blog.*



## Bookmarks for Editing

Bookmarks define a section of the workspace containing a number of objects. When editing a workspace without bookmarks, moving objects is done by selecting the object or objects, and dragging them to a new position.

However, when a workspace is divided by bookmarks, objects can be moved by simply dragging the bookmark to a new position. When an object is located inside the bookmark, it moves as the bookmark does.



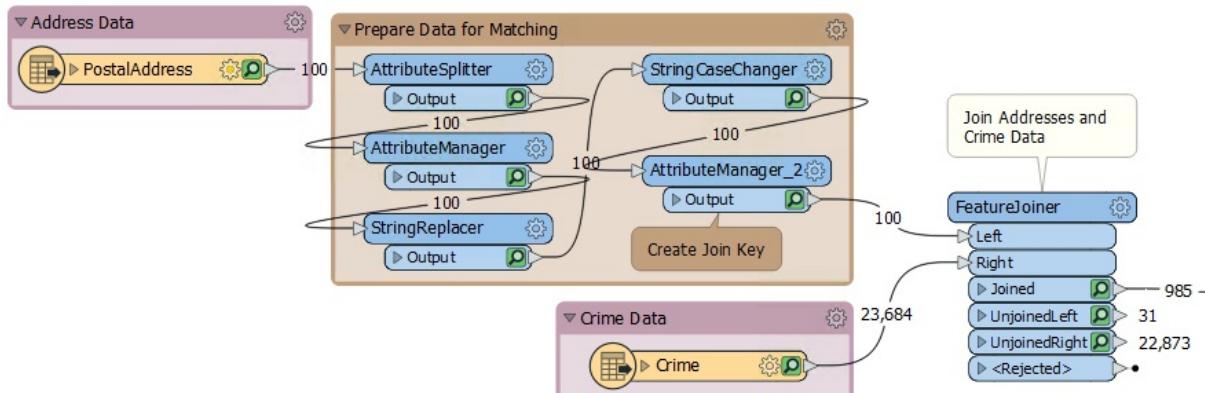
Using this technique, large groups of objects can be moved about the workspace canvas to create a clearer layout. Objects are moved inside a bookmark, whether the bookmark is collapsed or expanded.

### FME Lizard says...

*Remember that one of the bookmark properties is labelled **Contents** and has the value either "Move with Bookmark" or "Move Independently". Obviously the former value causes objects to move with the transformer; the latter value causes objects to remain in position when a bookmark is moved.*

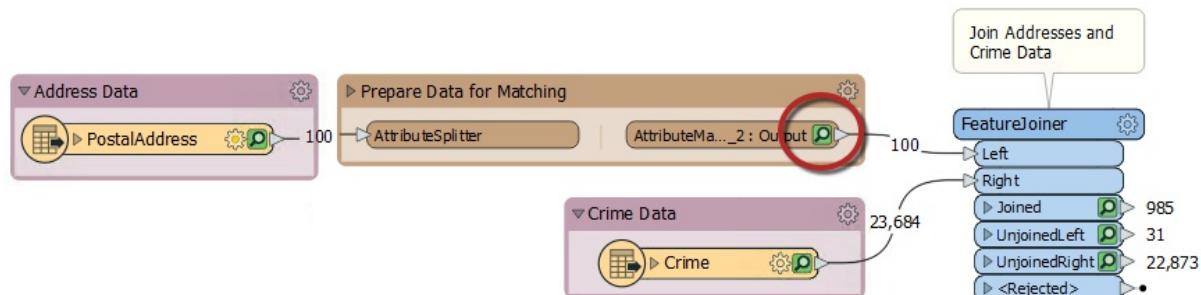
## Bookmarks for Performance

When a workspace is run with Data Caching turned on, then features are cached at every transformer. As you can imagine, in larger workspaces this leads to a lot of data being cached, sometimes unnecessarily:

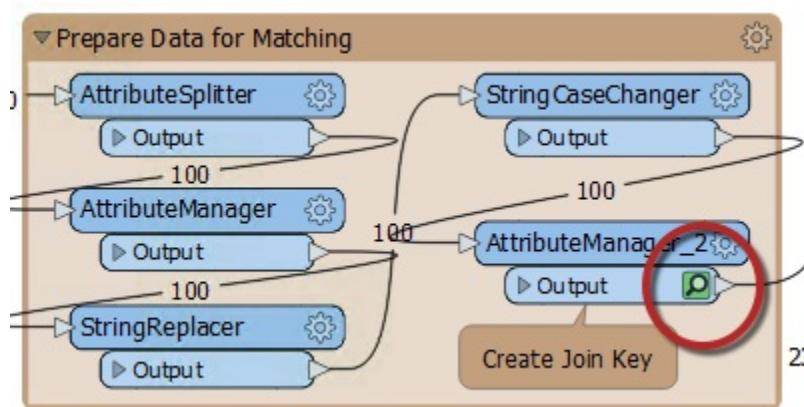


Notice in the above screenshot that every transformer in the Prepare Data for Matching bookmark is being cached.

However, when a bookmark is collapsed, then caching only occurs on the bookmark output objects:



This feature means that data is cached only for the final transformer in the bookmark, saving considerable time and resources:



### FME Lizard says...

Obviously you don't want to put a workspace into production when caching is turned on, regardless of whether your bookmarks are collapsed. This technique is only recommended for use in the design, authoring, and testing phases of workspace creation.

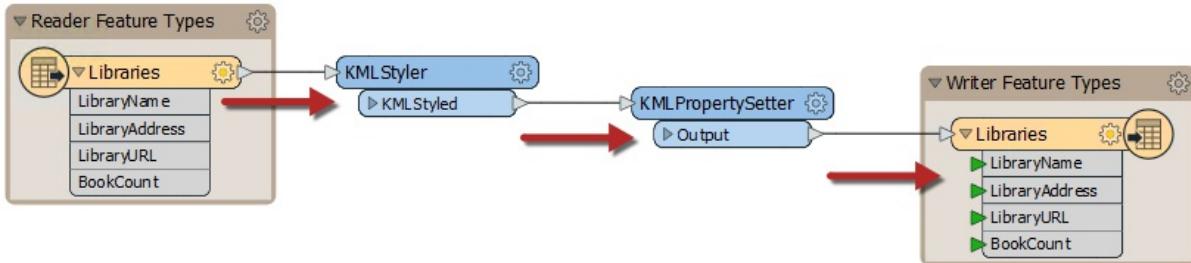


## Object Layout

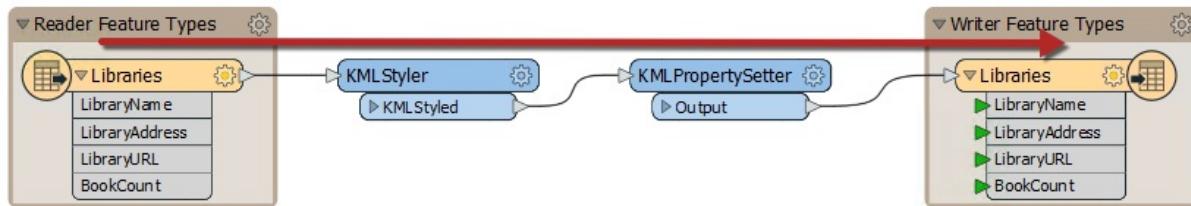
The positioning of workspace objects and the care taken in connecting them can make the difference between a poorly-designed workspace and one that is visually attractive and efficient.

### Object Layout

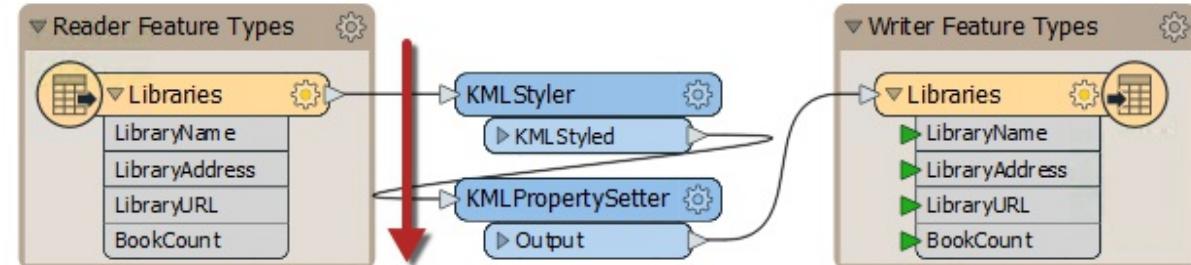
Layout methods vary from user to user. Some users like to line up objects so that all *connections* are horizontal:



Others prefer the tops of *objects* to be aligned horizontally, with angled connections:



Some prefer to align object edges vertically:

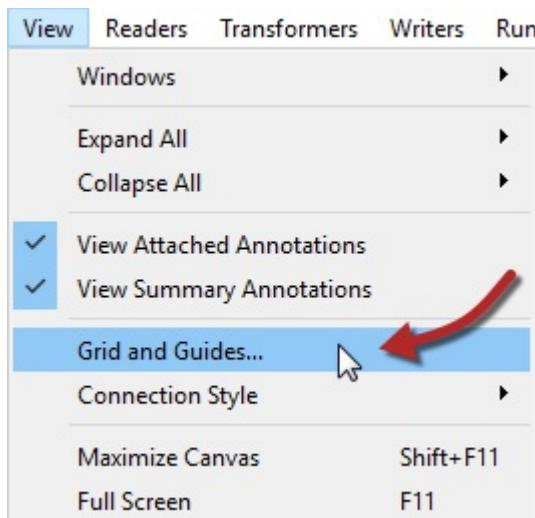


The style used is more of a personal preference than a definite rule, but what is important is consistency. A workspace that has no apparent layout style, or an inconsistent one, does not inspire confidence in the author's abilities!

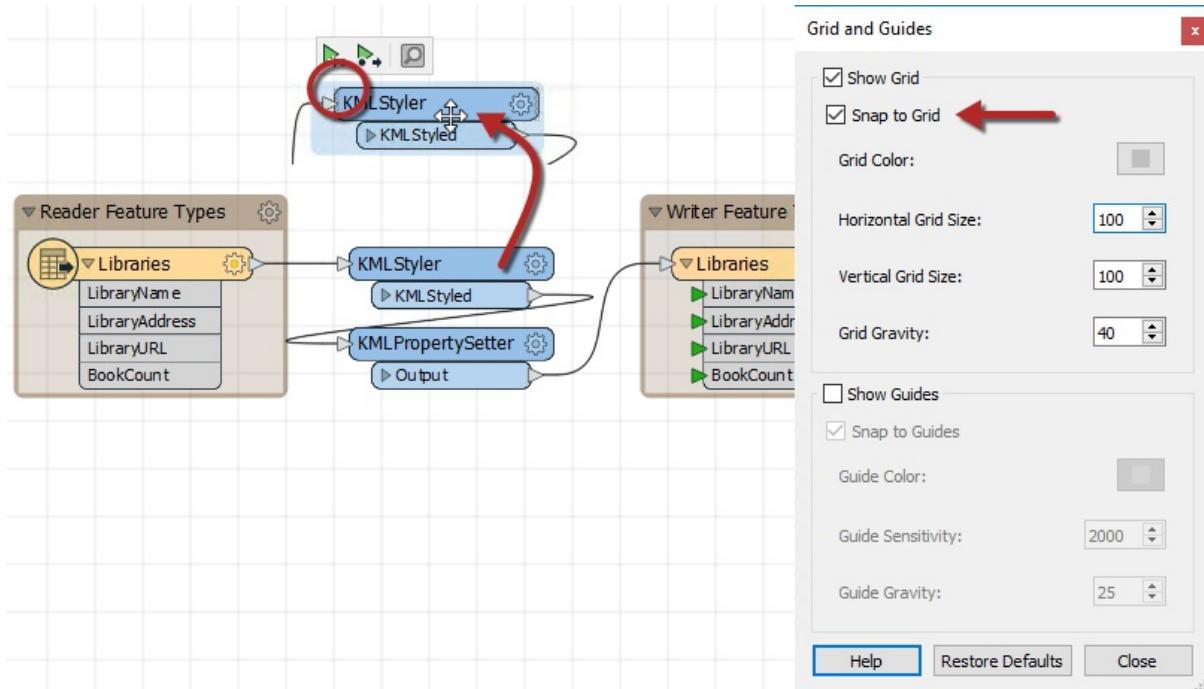
---

### Grid and Guides

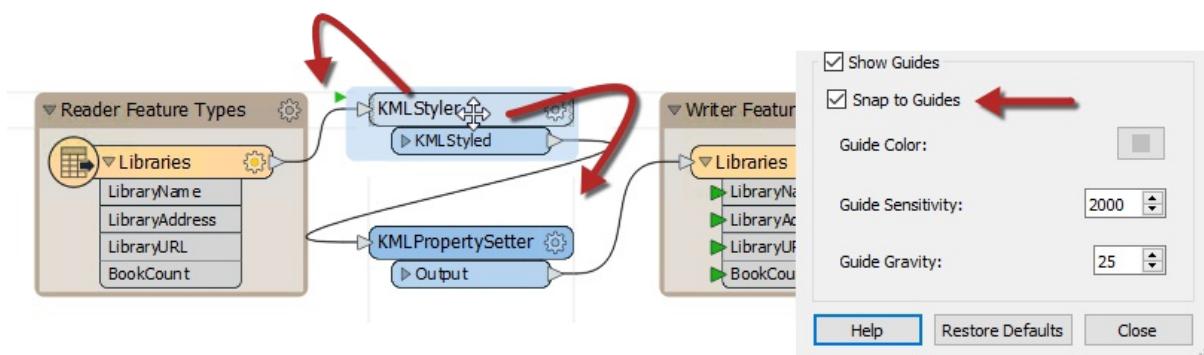
Grids and Guides are a tool to help align workspace objects neatly and tidily. This functionality is accessed through View > Grid and Guides on the Workbench menu bar.



**Show Grid** causes a grid of lines to be displayed on the Workbench canvas. Snap to Grid causes all objects - such as the summary annotation highlighted - to snap onto the intersection of grid lines when moved. In this way, objects can be more easily lined up.



**Show Guides** causes guidelines to be displayed on the Workbench canvas whenever an object is moved, and lines up approximately to another canvas object. Snap to Guides allows an object to be snapped onto a highlighted guideline.

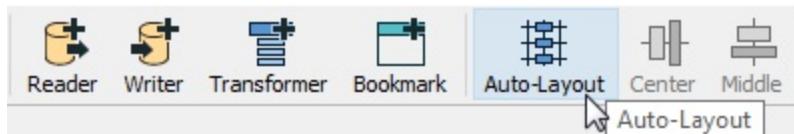


These two tools make it very simple for workspace objects to be aligned in a pleasing style.

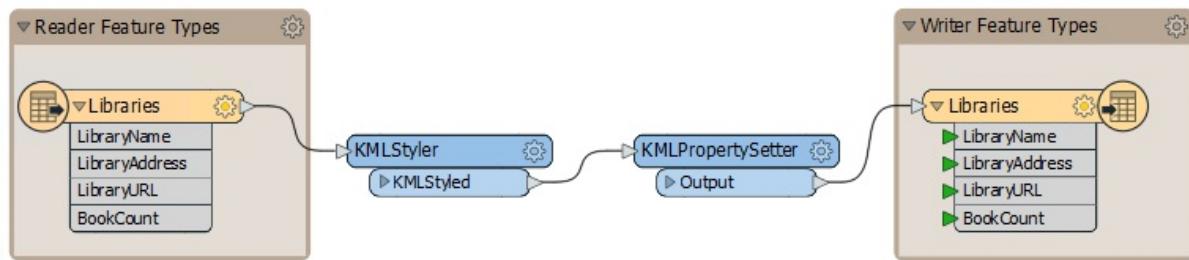
---

## Autolayout

The Autolayout tool appears on the toolbar of FME Workbench:



Clicking the toolbar button will layout either all of the workspace or just objects that are currently selected:



As you can see, the autolayout tends to use a horizontal pattern, with the tops of objects aligned. Therefore it's better to select groups of transformers at a time, when using this tool, rather than trying to lay out the entire workspace in a single action.

---

### FME Lizard says...

*In general, the autolayout algorithm is OK... but it still can't compare with taking the time and effort to manually organize your object layout.*

## Connection Style

It's also worth noting that object positioning is only part of a good layout. The other key part is the connection style.

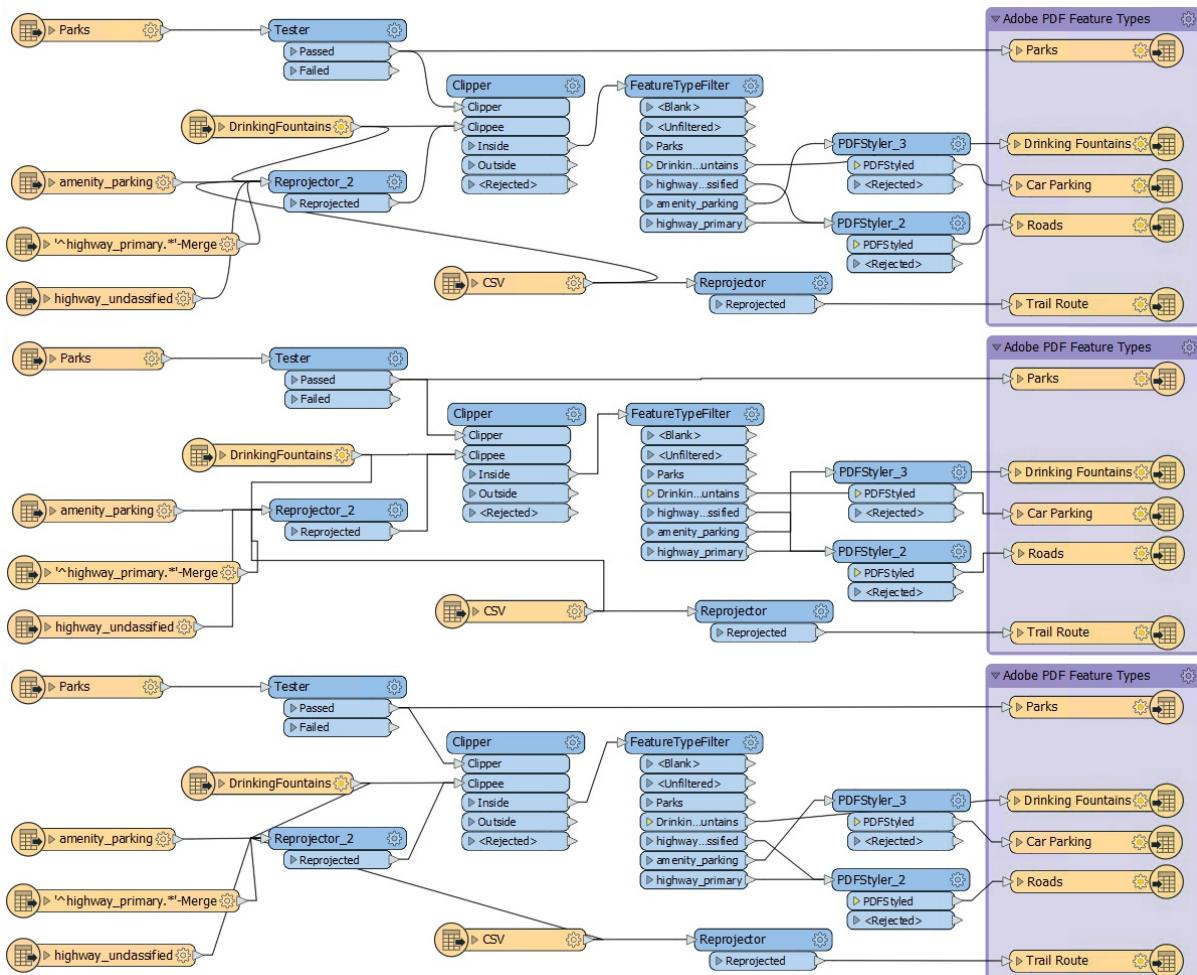
As with the positioning of workspace objects, the care taken in connecting them can make the difference between a poorly-designed workspace and one that is visually attractive and efficient.

### Connection Styles

Connections are the lines between objects on the workspace canvas. There are three different styles of connection that you can create in Workbench:

- Curved: The default style with curved line connections
- Squared: A style that evokes a Manhattan skyline through squared connections
- Straight: The original connection style; a straight line between two objects

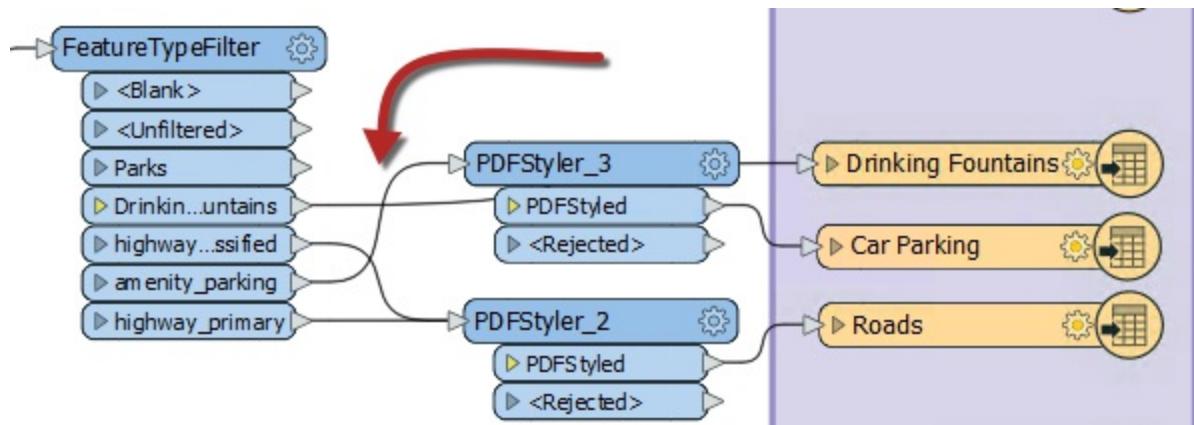
You can switch between styles using either the View menu, the FME Options menu, or the shortcut **Ctrl+Shift+C**. This image shows a comparison of the three styles:



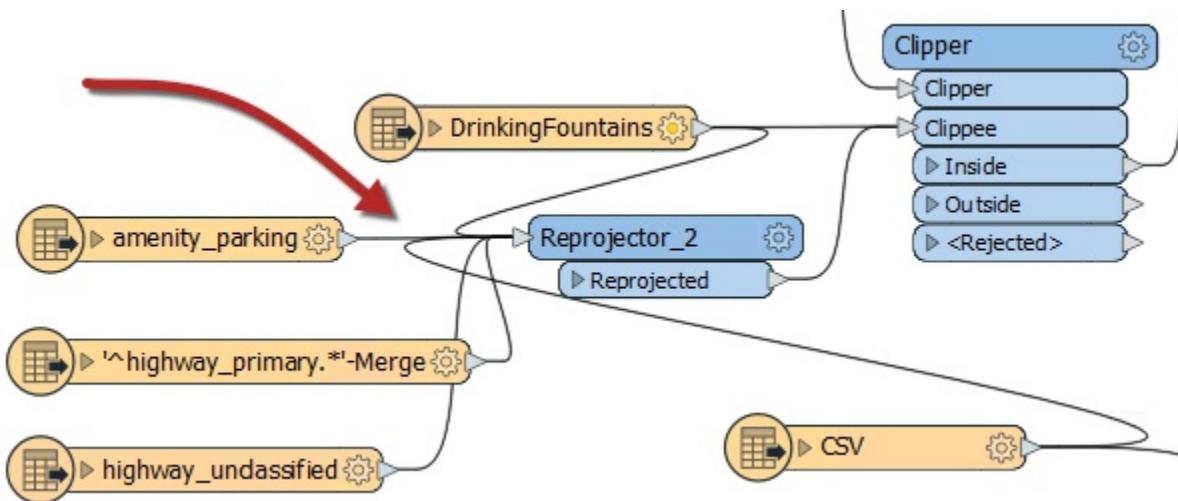
Once more, there is no right or wrong choice about which style to use; it is more a personal preference. However, object layout and connection style are related; the best FME authors will vary the position of objects according to the connection style used to avoid issues like overlapping connections.

### Overlapping Connections

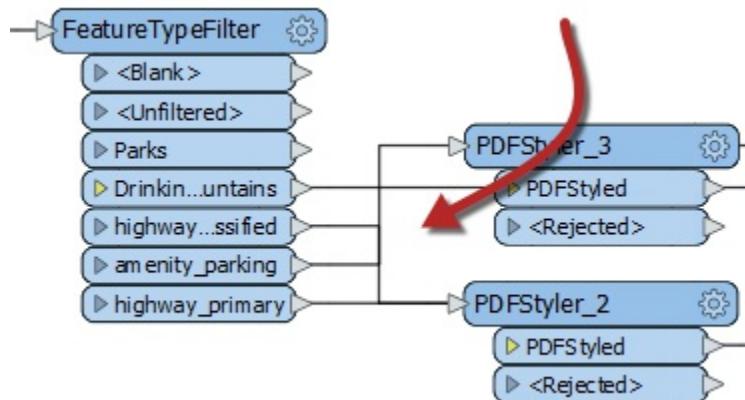
One of the most obvious failings of a workspace design is to have connections that cross over each other, for example like this:



The visibility and intent of a connection are always compromised when it overlaps with either another connection or another object on the canvas. However, the choice of connection style affects the possibility of overlap occurring. For example, curved connections tend to cross over more than straight ones:



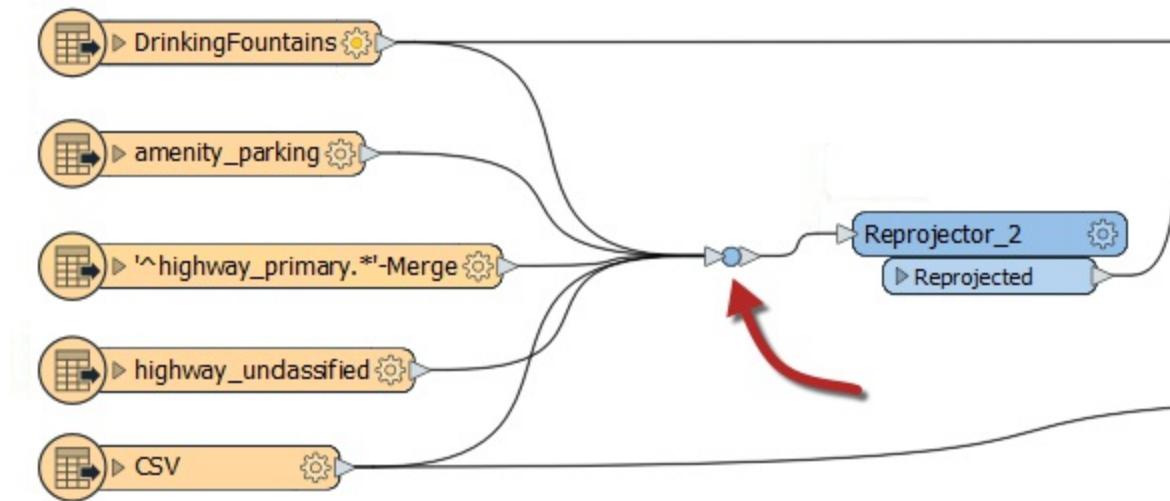
...and squared connections can sometimes cross in ways that are difficult to decipher:



Because these issues can spring up when you switch connection style, it's wise to choose a particular connection style and layout technique and stick with it. For example, in a curved connection workspace transformers could perhaps be spaced more widely to avoid overlaps.

## Junctions

There is one transformer in FME Workbench that is designed to be used to enhance the layout of objects and connections. That transformer is called the **Junction**.

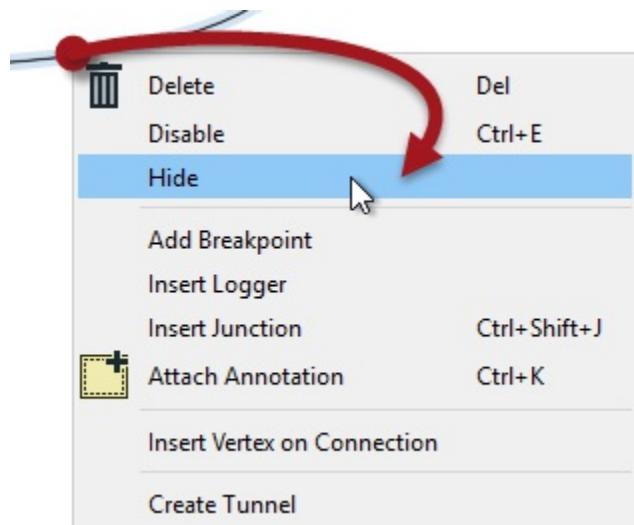


This transformer is a small, node-like object, that carries out no function on the data, but is instead used to tidy connections within a workspace - as in the above screenshot. This trait makes it an excellent tool for best practice.

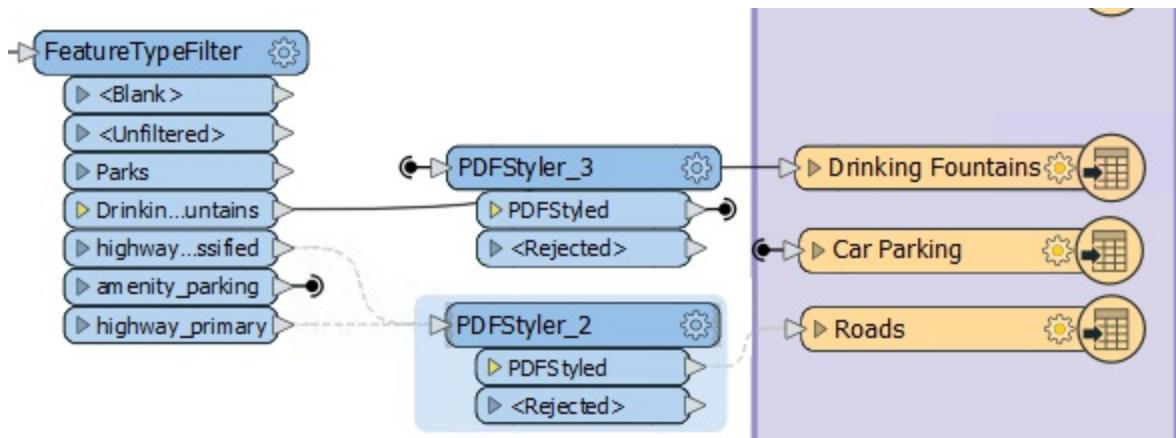
As with any other transformer, a junction can be connected to an Inspector or Logger, and it can have annotation objects attached to it. It also works with both Quick Add, Drag/Connect functionality, and Feature Caching.

## Hidden Connections and Tunnels

The ability to hide connections is especially useful for avoiding overlaps. To hide a connection, right-click on it and choose the option to Hide:



A hidden connection is represented by a 'transmitter' icon, or by a greyed-out dashed line when the object at one end of the connection is selected:



Here the object (transformer or feature type) must be selected for the connection to be visible.

The other available option is "Create Tunnel." This choice creates a hidden connection with the addition of an annotated junction transformer at each end:



A tunnel makes a hidden connection slightly more apparent, plus allows for annotation at each end.

## Revisualizing Hidden Connections

To view hidden connections, click on an object at either end. The connection is highlighted as a greyed-out dashed line.

To return a connection to view, right-click an object to which it is connected and choose Show Connection(s).

For more information on Tunnels and Junctions see [this blog post](#).

## Exercise 3

## The FME Style Guide

<b>Data</b>	Addresses (Esri Geodatabase) Crime Data (CSV - Comma Separated Value) Parks (MapInfo TAB)
<b>Overall Goal</b>	Work on Vancouver Walkability Project
<b>Demonstrates</b>	Style Best Practice
<b>Start Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\BestPractice-Ex3-Begin.fmw
<b>End Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\BestPractice-Ex3-Complete.fmw

Continuing from the previous exercise, you have been assigned to a project to calculate the "walkability" of each address in the city of Vancouver.

Your colleague wasn't aware of FME style best practice when they gave us the workspace, which made working with it a bit challenging. We need to present our workspace, so we want it to look neat, organized, and well-documented.

### 1) Start Workbench

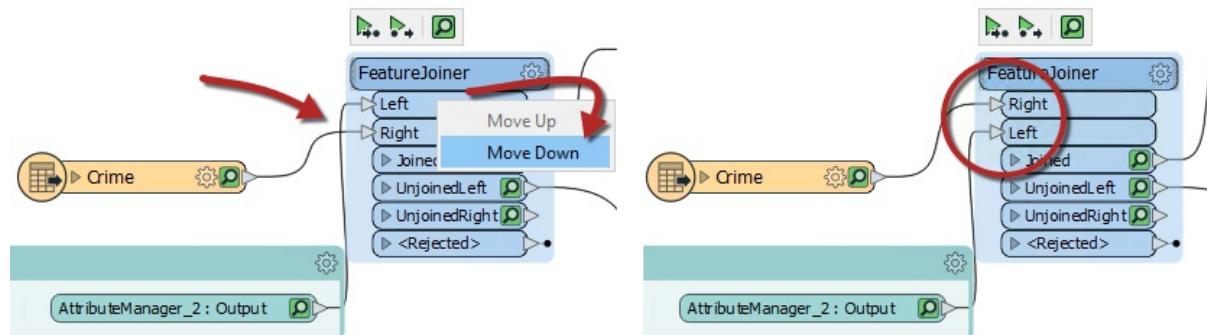
Start FME Workbench and open the workspace from the previous exercise. Alternatively, you can open C:\FMEData2019\Workspaces\DesktopBasic\BestPractice-Ex3-Begin.fmw.

**Note:** Remember, if you clicked the AutoLayout button in the first exercise, your workspace will look different. Pay close attention to transformer and port names.

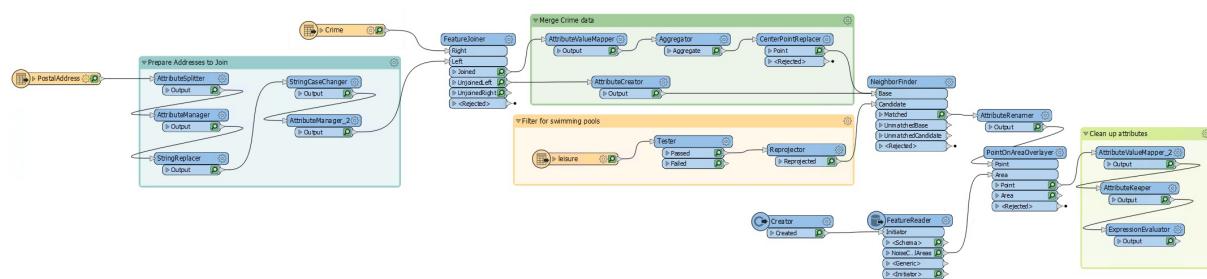
### 2) Rearrange Transformers

Firstly, let's clean up the transformers. Move the transformers around so that there are no overlapping connections.

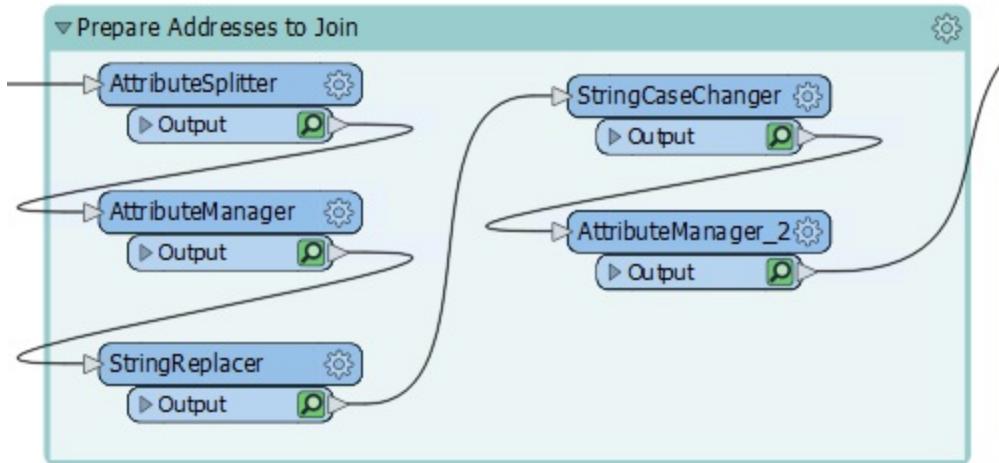
For the FeatureJoiner, you could move the Crimes reader below the Prepare Addresses to Join bookmark, or you can reorder the FeatureJoiner ports. Right click on the Left input port, and select Move Down. Now the two connection lines are not crossing:



Move the transformers into a logical order and add a bookmark around any logical groupings:

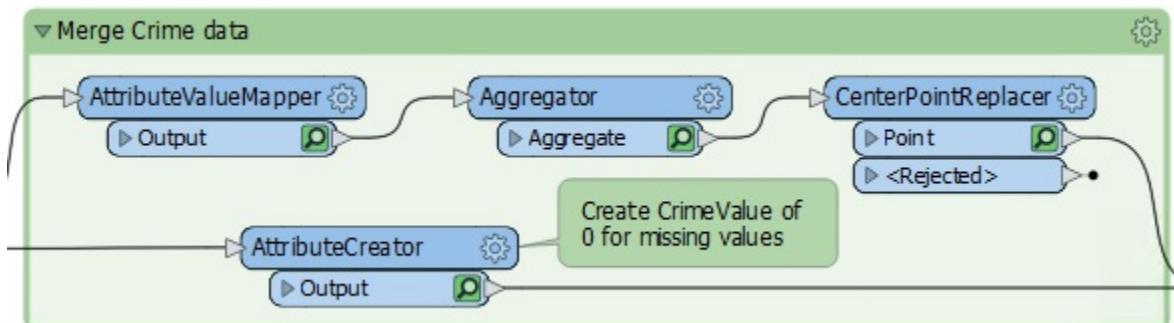


Don't forget to expand the Prepare Addresses to Join bookmark from the previous exercise and organize those transformers:

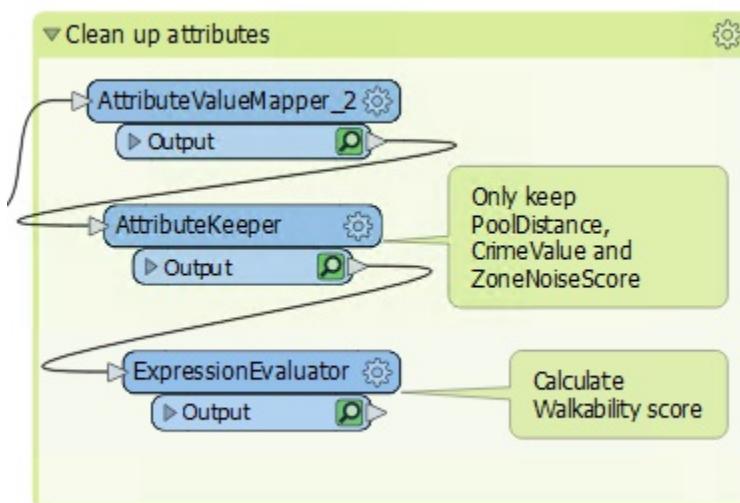


### 3) Add Style

Having rearranged the transformers and added bookmarks we can now add annotations and color to highlight what is going on. This step will require some inspection of the transformers to find out what they are doing as well as inspecting the readers to know which format they are in:

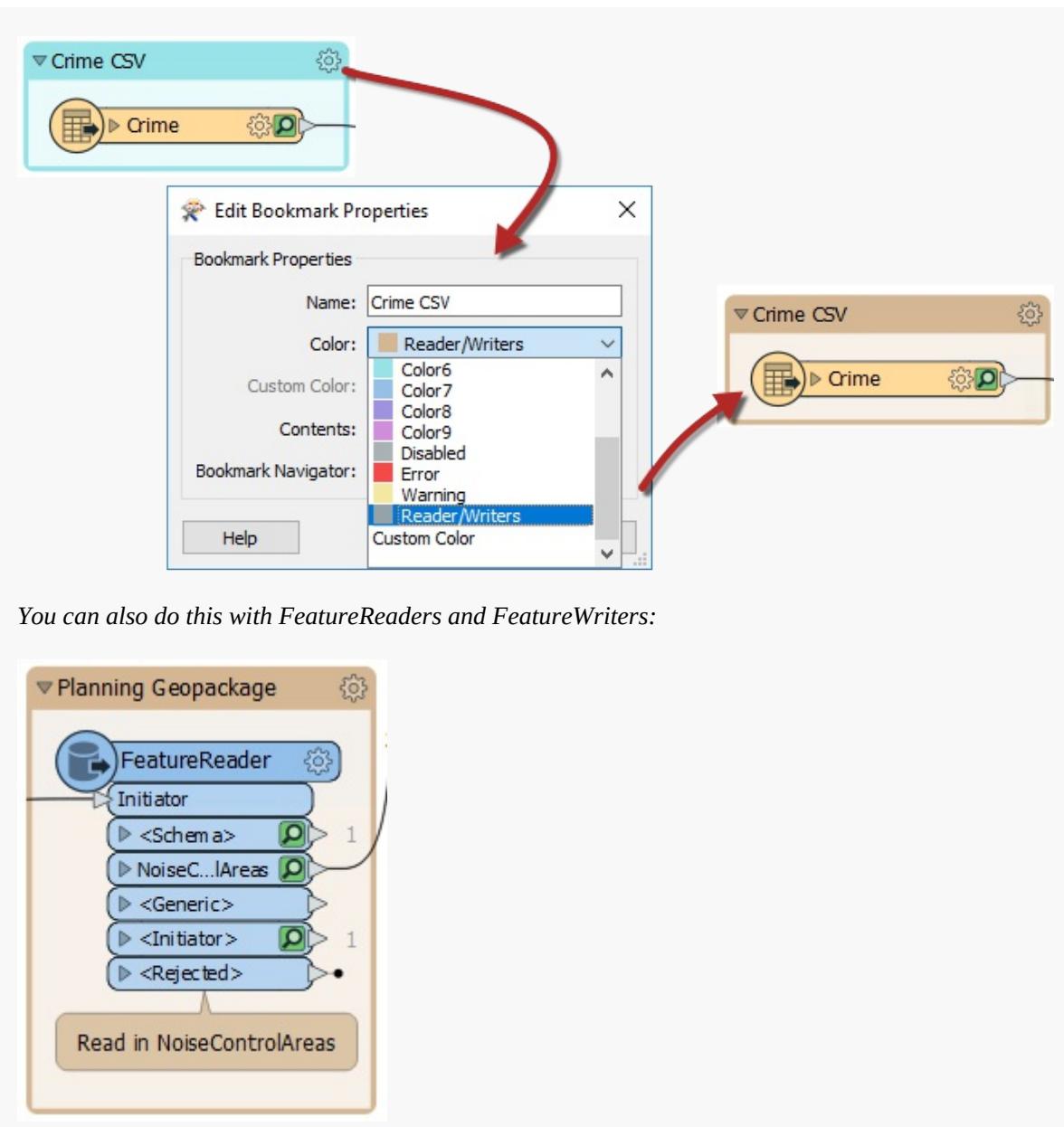


Adding good annotation where necessary will help determine what is going on in the workspace:

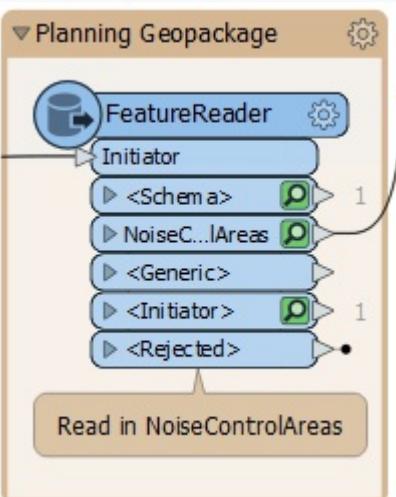


### TIP

*By adding a bookmark around a reader or writer and then setting the color to the preset Readers/Writers color, it is quick to see at a glance where your readers or writers are:*



You can also do this with FeatureReaders and FeatureWriters:



#### 4) Run Workspace

Collapse all the bookmarks if you wish and run the workspace once more to ensure all the caches are fresh. It might be a good idea to re-run the entire workspace.

#### 5) Save the Workspace

You can choose to save this workspace as a regular workspace or as a template workspace.

## CONGRATULATIONS

By completing this exercise you have learned how to:

- Rearrange transformers into a logical layout that groups those carrying out a single task
- Use annotations to clarify the processes taking place in a workspace
- Use bookmarks to turn a single workspace into defined sections

- *Avoid poor design choices like overlapping connections*

## **Module Review**

This module was designed to help you use FME Workbench efficiently, to effectively manage FME related projects, and to ensure those projects are scalable and portable.

## **What You Should Have Learned from this Module**

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

### **Theory**

- Best Practice is the act of using FME in a manner that is efficient, but also easily comprehensible by other FME users.
- Best Practice in FME can be divided into Style, Methodology, and Debugging.
- Debugging is the act of locating and fixing defects within a workspace.
- Methodology is a practice that defines the optimum transformers to use for maintenance and performance purposes.
- Style is a practice that makes it easier to navigate and understand an existing workspace.

### **FME Skills**

- The ability to use bookmarks and annotation to create a well-designed workspace style.
- The ability to create workspaces that are easier to maintain, edit, and scale.
- The ability to create workspaces that perform efficiently.
- The ability to interpret an FME log file.
- The ability to inspect output and feature counts to locate errors.
- The ability to use feature debugging to trace individual features.

### **Further Reading**

For further reading why not browse [articles tagged with Best Practice](#) on our blog?

## Best Practice Quiz

Each section ends with a quiz to test your new knowledge. Make your selection and click "Check my answers" to check each individual question. If you want an explanation for the answer, click "Explain".

**Note:** your score won't be tallied; this is just for review purposes.

**1)** Which of the following are methods of creating a bookmark (Select All That Apply)?

- A. Click the Insert Bookmark button on the toolbar
- B. Select a transformer, right click, choose Create Bookmark
- C. Select multiple transformers, right click, choose Create Bookmark
- D. Use the Ctrl+B shortcut

**2)** It's possible to disable other objects besides connections. Can you pick out which of these objects (there may be more than one) can be disabled in Workbench?

- A. Transformers
- B. Feature Types
- C. Annotation
- D. Bookmarks

## Exercise 4

## FME Hackathon

<b>Data</b>	Roads (Autodesk AutoCAD DWG and/or PostGIS)
<b>Overall Goal</b>	Find the shortest route from the hackathon to an Italian Cafe
<b>Demonstrates</b>	Data Translation, Transformation, and Best Practice
<b>Start Workspace</b>	None
<b>End Workspace</b>	C:\FMEData2019\Workspaces\DesktopBasic\BestPractice-Ex4-Complete.fmw

A regional GIS group is holding an FME Hackathon, and you have been invited to take part.

You have been provided with a set of source data and asked to create a useful project from it. You decide that it would be interesting to produce a tool that maps the route from the hackathon venue to a cafe where a group get-together will be held that evening.

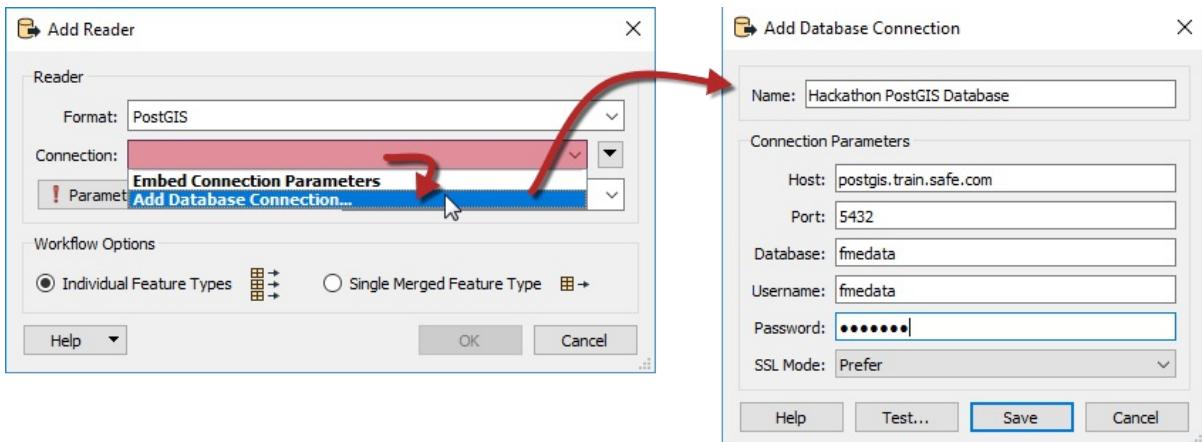
So, your task is to use the data available to you to calculate the best route from the convention center to the cafe and to write out that data to GPX format so people can use it in their GPS/mobile device.

### 1) Create Database Connection

The source data has been provided in a PostGIS database; therefore our first task should be to create a connection to it. That way we can use the connection instead of having to enter connection parameters.

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

In FME Workbench, add a PostGIS reader, then for the Connection, click the drop-down and select Add Database Connection. Then in the Add Database Connection dialog, enter the connection parameters obtained through the web browser for the PostGIS on Amazon RDS database. For Name enter Hackathon PostGIS Database:



Click Save. Click on Parameters to select the table we want to read from the database, select the public.CompleteRoads table. Then click OK twice to add the reader to the canvas.

<b>Reader Format</b>	PostGIS
<b>Reader Dataset</b>	Hackathon PostGIS Database
<b>Parameters</b>	Table List: public.CompleteRoads

### TIP

To add a connection without first adding a reader, select Tools > FME Options from the menu bar.

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

The completed workspace for this exercise uses a database connection called **Hackathon PostGIS Database**

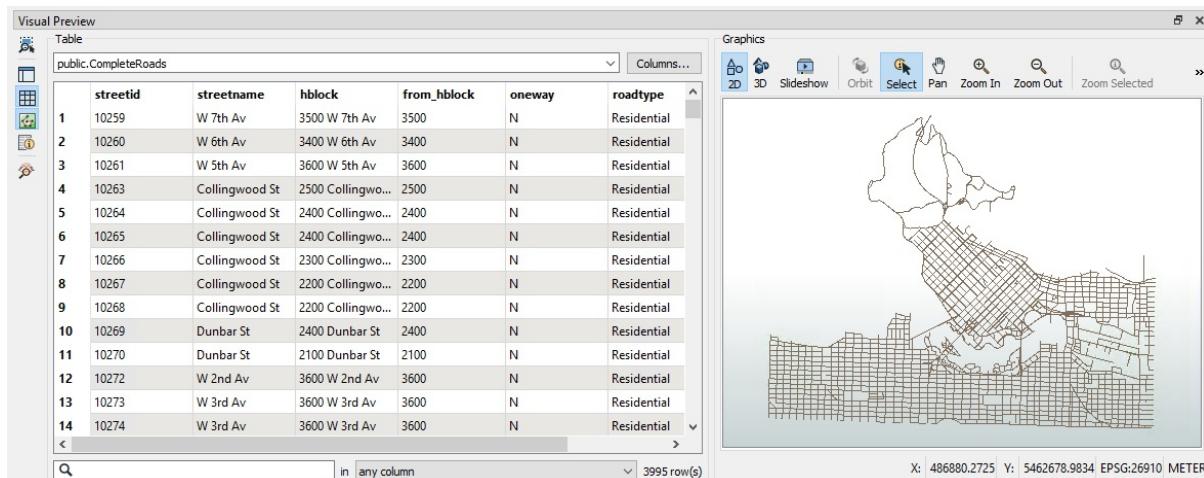
If you wish to open/use this workspace, you should create your connection with the same name. That way when you open the workspace, it will automatically find a matching connection.

This is a good illustration of the importance of naming conventions for database (and web) connections.

## 2) View the Data

Click on public.CompleteRoads feature type to open the popup menu. Then click the View Source Data button to view the data in Visual Preview.

You will see a set of road features each of which has a set of attributes. One attribute specifies whether a feature represents a one-way street. It's important to know this if we want to calculate a route that is actually legal!



**Note:** If you have any problems using the PostGIS database - for example connectivity problems with a firewall - then the following AutoCAD dataset can be substituted with very few changes required:

<b>Reader Format</b>	Autodesk AutoCAD DWG/DXF
<b>Reader Dataset</b>	C:\FMEData2019\Data\Transportation\CompleteRoads.dwg

## 3) Add GPX Writer

Now that we know that we read in the correct table, let's prepare the writer. Add a GPX Writer to the canvas with the following parameters:

<b>Writer Format</b>	GPS eXchange Format (GPX)
<b>Writer Dataset</b>	C:\FMEData2019\Output\Training\Route.gpx
<b>Feature Types</b>	Select All

Following the best practices that were covered in this chapter, add bookmarks around the reader feature type and all of the writer feature types. The workspace will look like this:



GPX is a fixed-schema format, hence the six different writer feature types that are automatically created.

#### 4) Add ShortestPathFinder

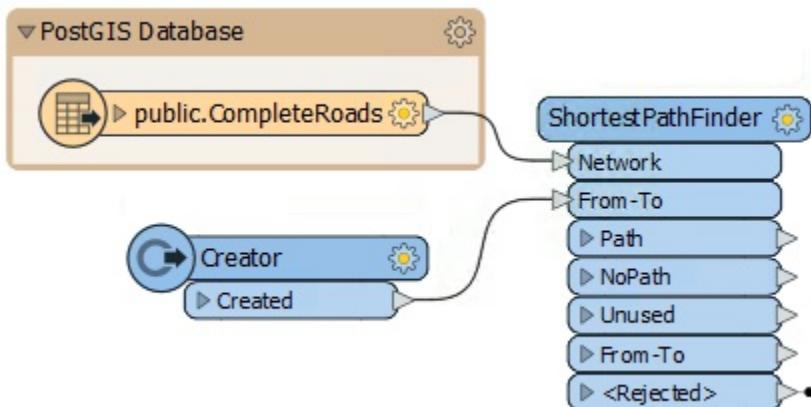
Now we need to start calculating a route. The obvious first step is to add a ShortestPathFinder transformer, which is how we can calculate our route.

So, add a ShortestPathFinder transformer. Connect public.CompleteRoads to the Network port.

#### 5) Add Creator

The other input port on the ShortestPathFinder is for the From-To path (the start and end points of our journey). There are many ways to create this - or even accept input from a web map - but here we'll manually create a feature with the Creator transformer.

So, add a Creator transformer and connect it to the From-To port:

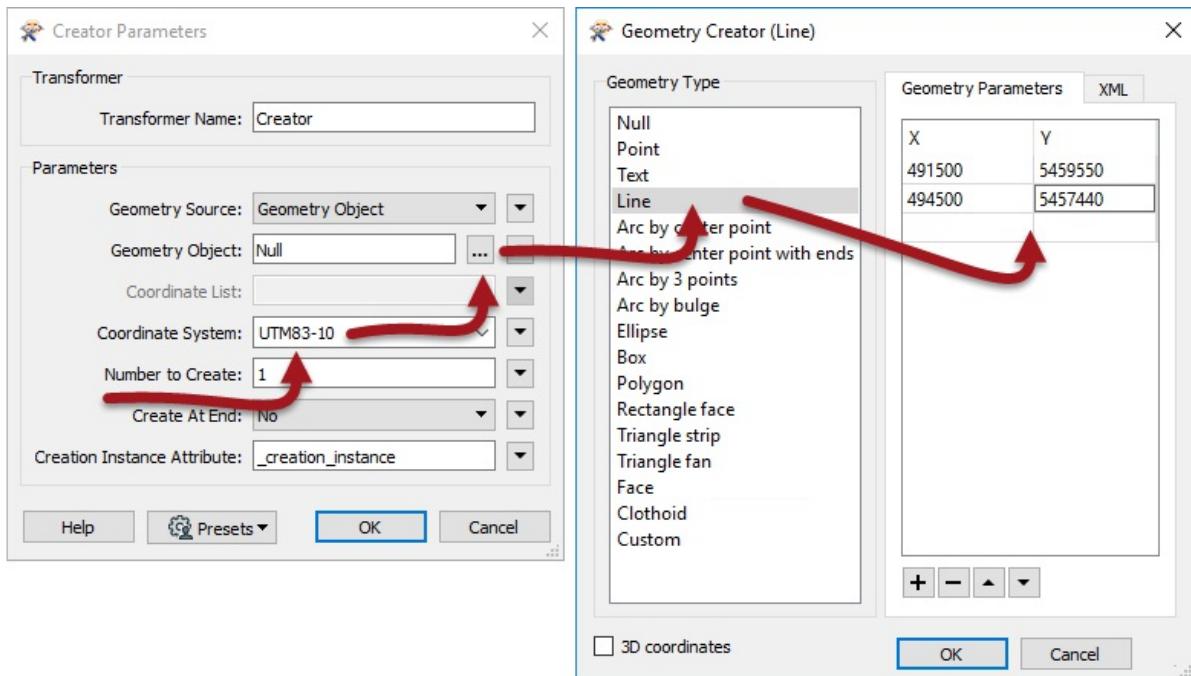


Inspect the Creator's parameters.

Firstly enter UTM83-10 as the coordinate system of the data we are about to create. For the Geometry Object parameter, click the [...] browse button to the right to open a geometry-creation dialog. Select Line as the geometry type and enter the following coordinates:

X	Y
491500	5459550
494500	5457440

The first coordinate is that of the hackathon venue and the second is the closest point in our network to the cafe we're going to visit.

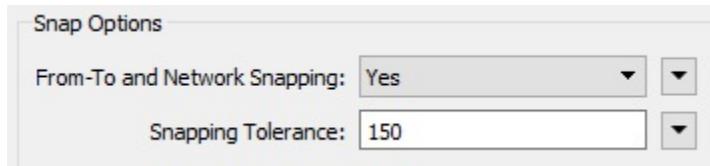


Click the OK button to confirm the changes.

## 6) Check ShortestPathFinder Parameters

The coordinates of the feature we've added might not sit exactly on the road network. To get around this issue, there are parameters we can use in the ShortestPathFinder.

So, inspect the ShortestPathFinder parameters. Under Snap Options set **From-To and Network Snapping** to Yes and enter 150 as the **Snapping Tolerance**:



Also, notice that there are parameters for network costs - we'll be making use of those later.

## 7) Run Workspace

Ensure feature caching is turned on and run the workspace. Check the log and then inspect the ShortestPathFinder:Path cache. If all has gone well, the output will look like this, with a route defined:



Map tiles by [Stamen Design](#), under [CC-BY-3.0](#). Data by [OpenStreetMap](#), under [CC-BY-SA](#).

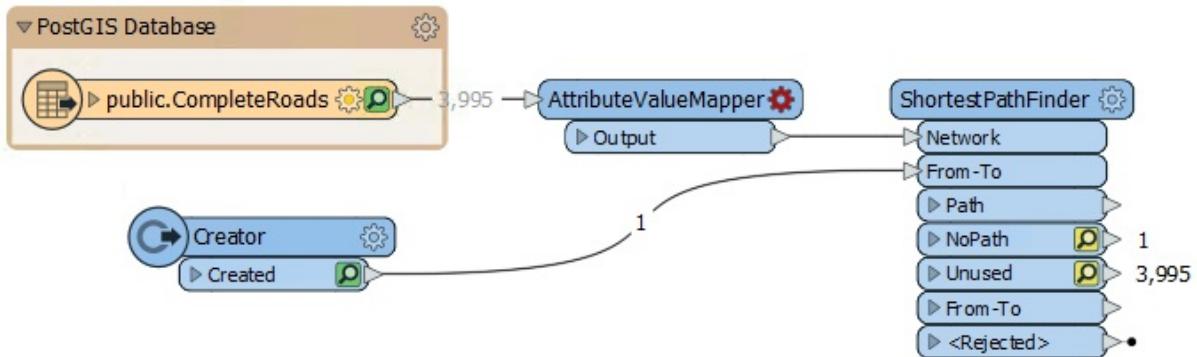
If you don't see a path like this one, you must make use of your debugging skills to try and locate the error!

### 8) Add AttributeValueMapper

The result looks fine, but there are some things we are still uncertain about: for example, what if the route uses slower, residential roads? We can force the route to prefer arterial routes by applying a different cost to each road feature.

The cost will depend on the road type. We need to map road type to cost, and the way to do that is with an AttributeValueMapper transformer.

Add an AttributeValueMapper transformer to the workspace between the CompleteRoads feature type and the ShortestPathFinder:Network port:



### 9) Edit AttributeValueMapper

Inspect the AttributeValueMapper's parameters. Enter the following values:

Source Attribute	Value
roadtype	

Destination Attribute	Cost
Default Value	2
<b>Attribute Selection</b>	
Source Attribute:	<input type="button" value="roadtype"/>
Destination Attribute:	<input type="text" value="Cost"/>
Default Value:	<input type="text" value="2"/> <input type="button" value="..."/> <input type="button" value="▼"/>

Now, underneath those parameters, we'll map some data.

Source Value	Destination Value
Arterial	1
Residential	3

**Value Map**

Mapping Direction:

Source Value	Destination Value
<input type="checkbox"/> Arterial	<input type="checkbox"/> 1
<input type="checkbox"/> Residential	<input type="checkbox"/> 3

If the route is arterial (a main road) it will get a cost of 1, residential routes will get a cost of 3, and all other types will get a cost of 2 (because that's the default value). Click Accept/OK to confirm the parameters.

## 10) Apply Costs

Now we have to apply the costs we have just created. Inspect the ShortestPathFinder's parameters. Enter the following values:

Cost Type	By Two Attributes
Forward Cost Attribute	Cost
Reverse Cost Attribute	Cost

### TIP

*Why "Two Attributes"? That's because with only a forward cost, I could only ever travel along a stretch of road in the same direction as the vertices are arranged in. Since I don't want to avoid roads based on their vertex direction, using two attributes tells FME the cost is the same in both directions.*

Now re-run the workspace to see if there is any difference in the result. It should look like this:



Map tiles by [Stamen Design](#), under CC-BY-3.0. Data by [OpenStreetMap](#), under CC-BY-SA.

So the cost weighting has made a difference. But there is a problem with this result...

### FME Lizard says...

*The route is taking a longer path this time, and I can see a reason why that might be: cost is being used to weight routes **instead** of the distance, not as well as.*

*To explain this, let's say I want to travel from A to B. There is a single residential road feature that starts at A and ends at B, with a route distance of 1.5 kilometres.*

*There is also a single arterial road feature that starts at A and ends at B. Rather extremely, it loops around the dark side of the moon with a route distance of 768,000 kilometres.*

*Currently our solution would choose the 768,000 kilometre trip, because it has a cost of "1" compared to the residential route cost of "3"!*

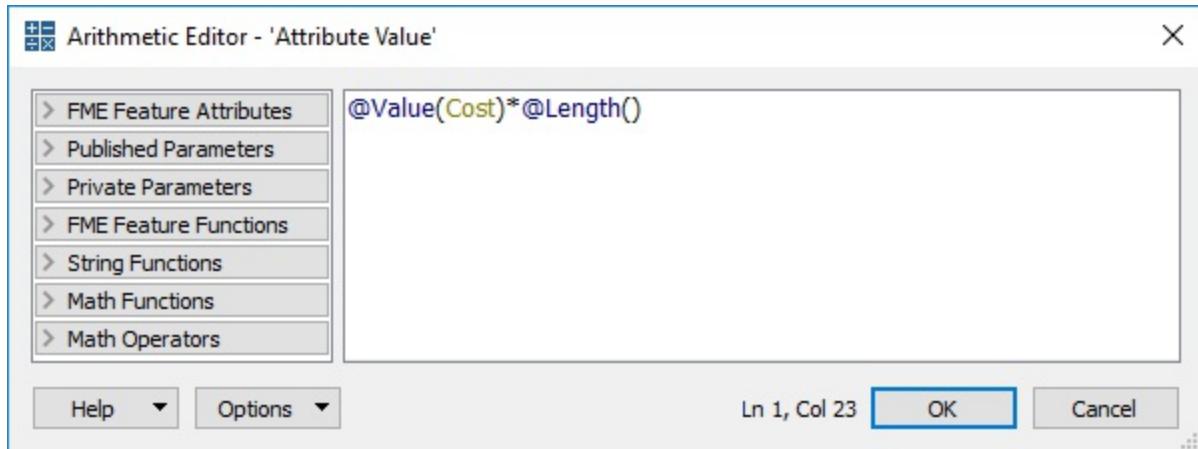
*Plainly, the exercise data here is nowhere near as ridiculous, but it's equally plain that the route might not be the optimum until distance is factored back into the result.*

### 11) Add AttributeManager

Add an AttributeManager transformer between the AttributeValueMapper:Output port and the ShortestPathFinder:Network port and view the parameters.

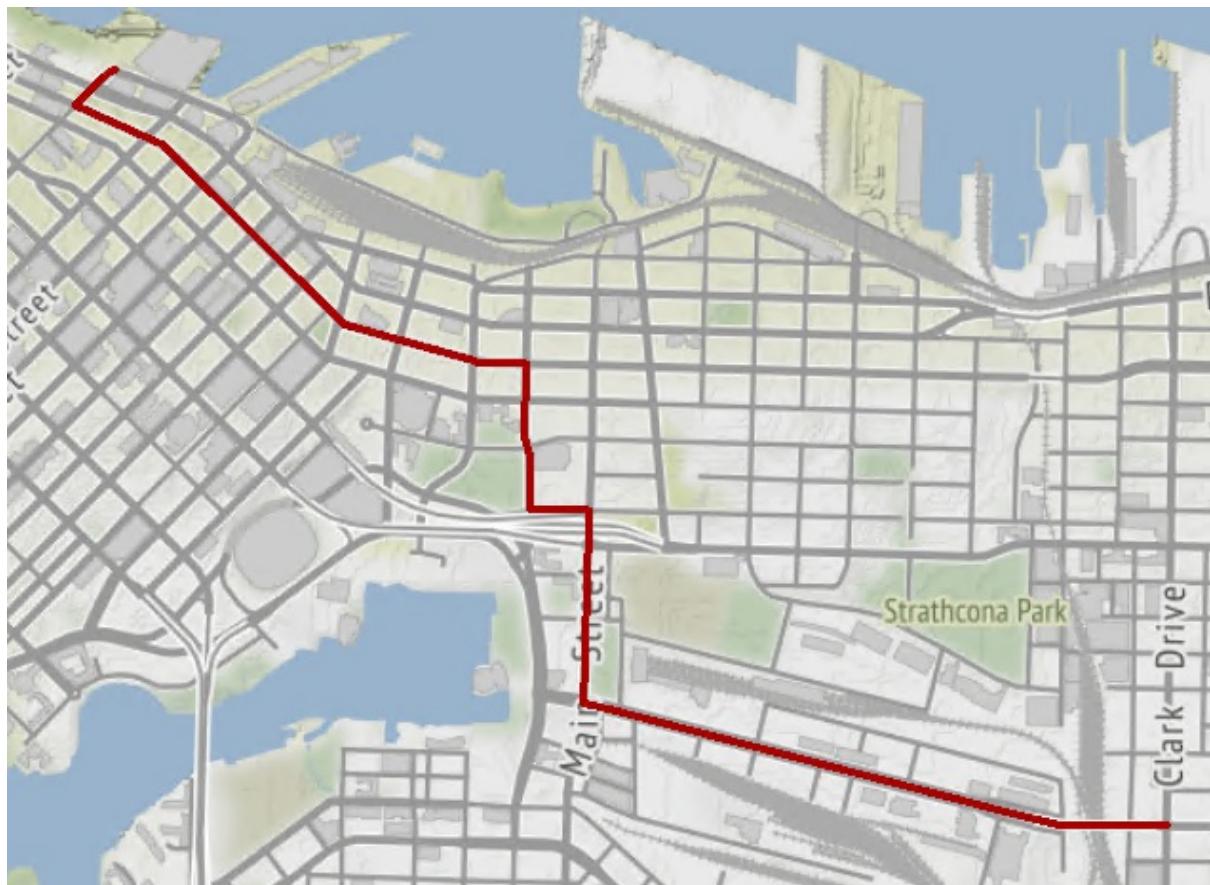
In the value field for the Cost attribute, click the drop-down arrow and select Open Arithmetic Editor. In that dialog enter the expression:

```
@Value(Cost)*@Length()
```



In short, we're now multiplying cost by road length to give us a combined weighting.

Re-run the translation to see if it makes a difference:



Map tiles by [Stamen Design](#), under [CC-BY-3.0](#). Data by [OpenStreetMap](#), under [CC-BY-SA](#).

Yes, it does, proving that the route was longer than necessary. Of course, the expression we've used is subjective and could be made more complex to give a better result. For example, we could try a logarithmic scale instead to see what that produced.

## 12) Edit AttributeManager

Regardless of our expression, there's another last problem to investigate: one-way streets. Currently, we have no

solution in place to prevent us from driving the wrong direction.

Luckily each one-way street is flagged with an attribute and has its vertices ordered in the direction of permitted travel, so we know which way to avoid. Let's use this information to solve that problem.

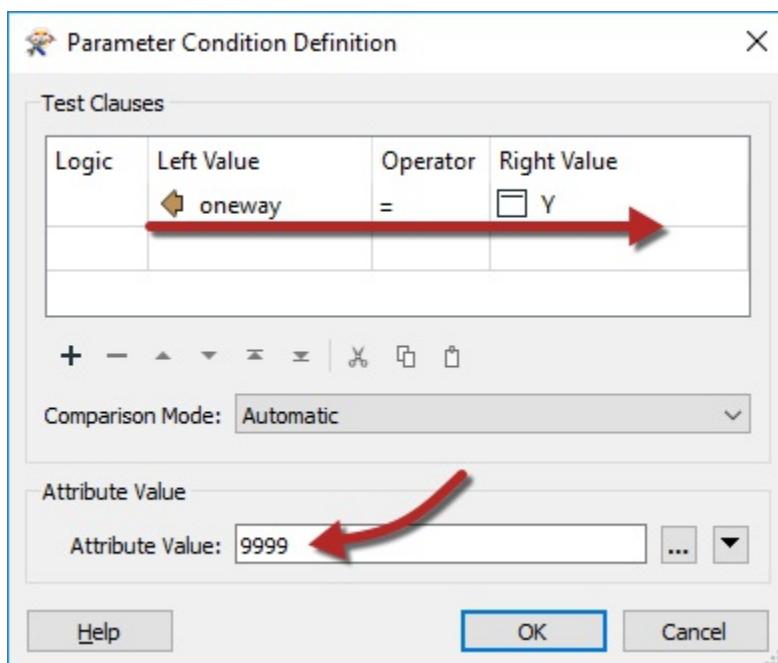
We'll need to calculate different cost attributes for each direction now, although the value will only differ when a one-way street is involved. There are - as usual - multiple ways to handle this in FME; let's go with a moderately easy one.

View the AttributeManager parameters again. This time create a new Output Attribute called ReverseCost. In the value, field click the drop-down arrow and choose Conditional Value.

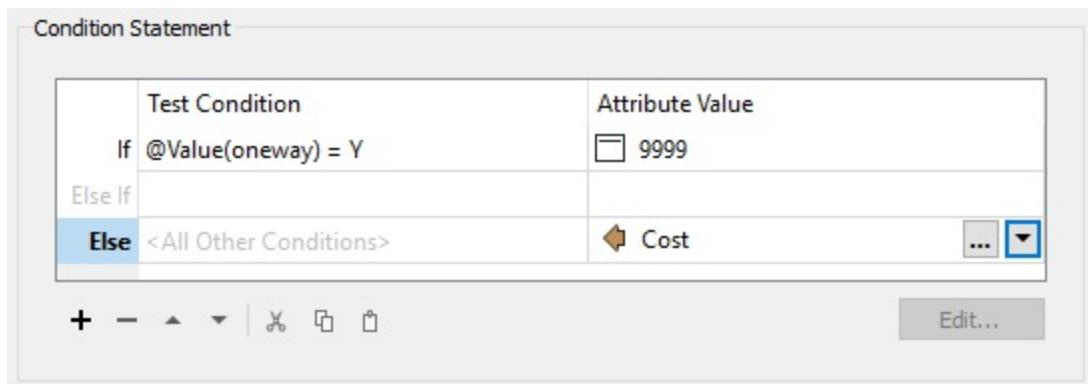
### TIP

*Conditional Values are those that are set dependent on a test condition. It's like incorporating a Tester into the AttributeManager. These are covered in more detail in the FME Desktop Advanced training course.*

In the definition dialog that opens, double-click in the first "If" row and a Test Condition dialog opens. In here set up a test for oneway = Y. For the Output Value (bottom of the dialog) enter the value 9999 (for example, the cost of traveling the wrong way is *really* expensive)!



Click OK to close that dialog. Back in the previous dialog, double-click where it says <No Action> choose the dropdown arrow and select Attribute Value > Cost:

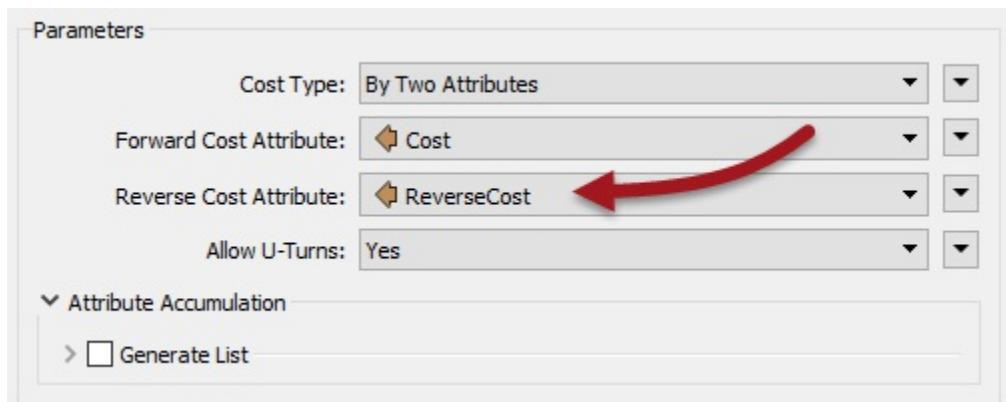


Click OK again twice more to close these dialogs.

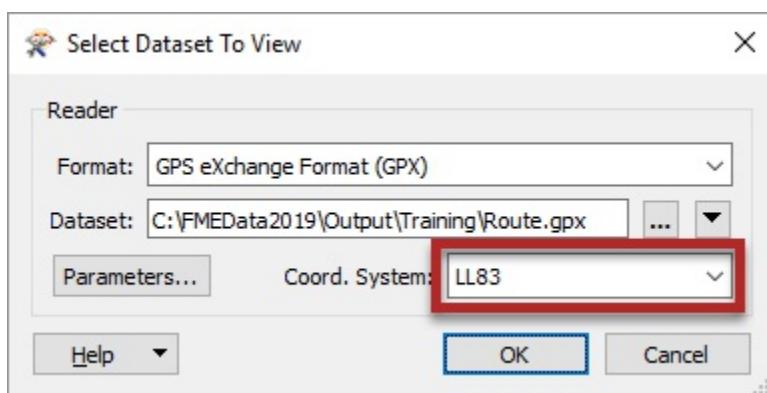
Now one-way streets have a prohibitively high reverse cost, while other streets just receive the usual forward cost.

### 13) Apply Cost

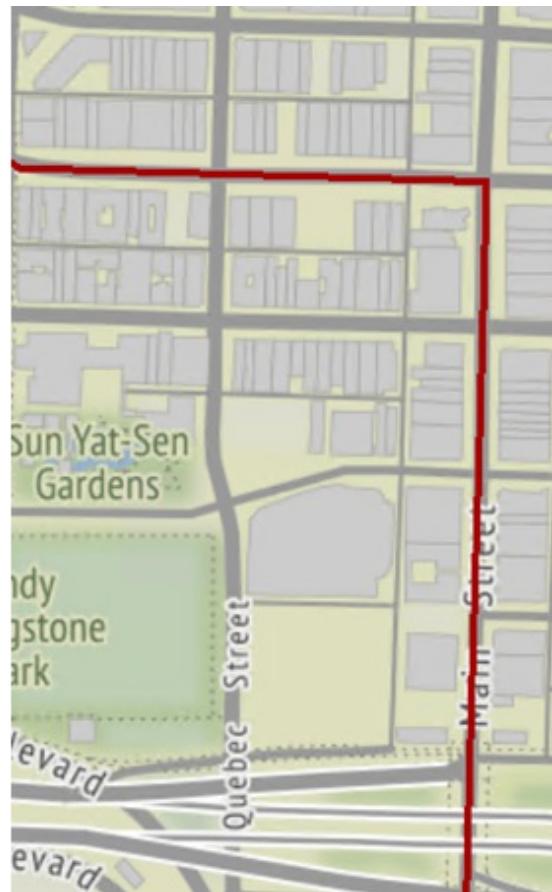
We have created a ReverseCost attribute, but we still need to use it. Check the ShortestPathFinder parameters and change the ReverseCost attribute from Cost to ReverseCost:



Now we're ready to go. Re-run the workspace. You could inspect the ShortestPathFinder:Path output port to see the results. However, if you want to see the **actual** written output, you will have to drag and drop the .gpx file onto Visual Preview. You can then specify to read the data with an LL83 coordinate system, as that information is not stored in GPX files:



In either case, you should see the correct output:

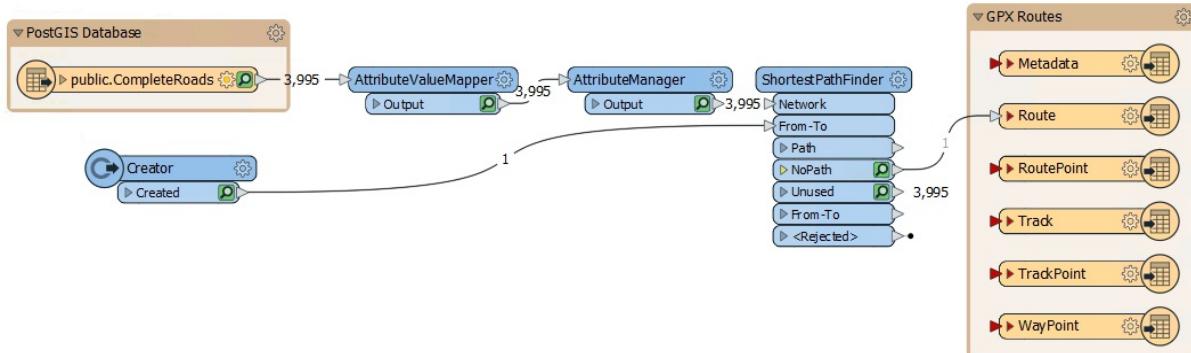


There is at one change (above) caused by a route that previously traveled the wrong-way (west-east) along a street that is one way (east-west)!

#### 14) Connect Schema

Connect the Path port to the Route output feature type:

**FIX PORT IMAGE**



Now run the workspace, upload the data to your GPS device, and you are ready to go!

#### Advanced Exercise

*Not really advanced, but you did use Best Practice throughout, right? I mean, you have bookmarks and annotations where needed, and no overlapping connections? As well, did you remove the additional GPX feature types we ended up not using? If not, well, you might want to fix that!*

---

## CONGRATULATIONS

*By completing this exercise you proved you know how to:*

- *Create and use an FME database connection*
- *Create a prototype FME workspace using a variety of transformers*
- *Use debugging techniques to find any problems encountered in an exercise*
- *Use good style for developing workspaces*

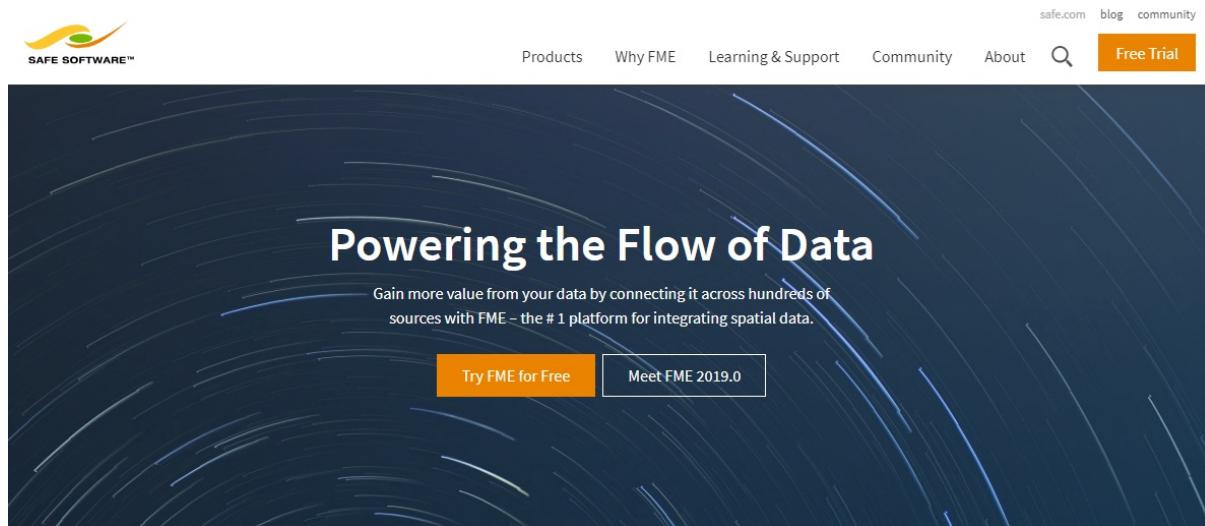
## **Course Wrap-Up**

Although your FME training is now at an end, there is a good supply of expert information available for future assistance.

## Product Information and Resources

### Safe Software Web Site

The [Safe Software web site](#) is the official information source for all things FME. It includes information on FME products, Safe Software services, FME solutions, FME support and Safe Software itself.



### Safe Support Team

Behind FME are passionate, fun, and knowledgeable experts, ready to help you succeed, with a [support team](#) philosophy built on the principle of knowledge transfer.



You can request product support through a Support Case (web/email) or using a Live Chat.

### Your Local Partner

Safe Software has partners and resellers around the world to provide expertise and services in your region and your language.

You can find a list of official partners on the [Safe Software Partners Page](#).



## Safe Software Blog

The [Safe Software blog](#) provides technical information and general thoughts about FME, customers' use cases, and spatial data interoperability. It includes articles, videos, and podcasts.

The screenshot shows the homepage of the Safe Software blog. At the top, there's a navigation bar with links for "About Data", "About FME", and "About Our Customers", along with a search icon. Below the header, there's a form to "Enter your email address" with a "Subscribe" button. A note says "Delivered by MailChimp".

The first post, titled "Spatial Asset Tracking: The Return of the QR Code" by Mark Ireland on December 24, 2017, features an image of a QR code with a green arrow pointing to a specific location. It discusses using QR codes for tracking assets like recycling bins or utility equipment. A "READ MORE" button is at the bottom.

The second post, titled "Podcast: Automating Data for Smart Cities" by Tiana Warner on December 12, 2017, features an image of a 3D rendering of a modern city with a river. It explains what a "smart city" is and how it gathers data from various sources. A "READ MORE" button is at the bottom.

## FME Manuals and Documentation

Use the Help function in FME Workbench to access help and other documentation for FME Desktop. Alternatively, look on our website under the [Learning & Support section](#).

# FME Documentation

Browse official FME Documentation.

## FME Desktop

### [FME Desktop Administrator's Guide](#)

Find out how to install and license your version of FME Desktop, and perform other administrative tasks. (PDF Version)

### [FME Readers and Writers](#)

A detailed technical guide to the many reader and writer formats available in FME.

### [FME Workbench](#)

A guide to FME's primary graphical tool for creating and running data transformations.

### [FME Transformer Reference Guide \(PDF\)](#)

A quick reference describing each transformer's functionality.

### [FME Workbench Transformers](#)

A detailed technical guide to the many transformers available in FME Workbench.

### [FME Data Inspector](#)

A guide to FME's graphical tool for inspecting transformation results and other datasets.

### [FME Integration Console](#)

Find out how to extend your "FME-ready" third-party applications so they will integrate with FME Desktop.

### [FME Quick Translator](#)

Use this tool to perform simple, automatic data conversions.

## FME Server

### [FME Server Documentation](#)

The FME Server Administrator's Guide, Reference Manual, Developer's Guide, and Web User Interface Help all in one place.

### [FME Server REST API Documentation](#)

The FME Server REST API provides a simple, open web interface for accessing core FME Server functionality. Use this technical reference to do anything from running a workspace to canceling a running job.

### [FME Server Tutorial](#)

A series of exercises to help new users get started with key tasks in FME Server.

## FME Cloud

### [FME Cloud Documentation](#)

Discover just how easy it is to get up and running with FME Cloud by browsing through this handy guide.

## Free Trial

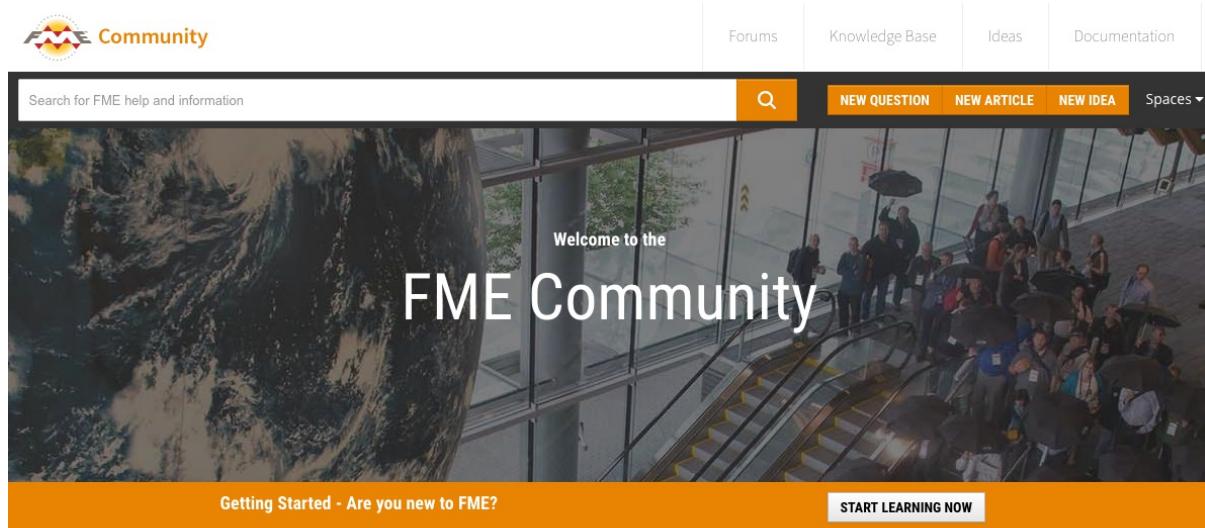
Safe Software offers a free 30-day trial of FME Desktop so you can learn at your own pace and develop new skills. Click on the [Download Free Trial](#) button to get started.

## Community Information and Resources

Safe Software actively promotes users of FME to become part of the FME Community.

### The FME Community

The [FME Community](#) is a one-stop shop for all community resources, plus tools for browsing documentation and downloads.



#### Forums

Browse topics, ask questions, and share your insights.

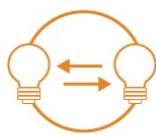
[JOIN IN](#)



#### Knowledge Base

Visit our library of authoritative how-to's, FAQs, demos, and more.

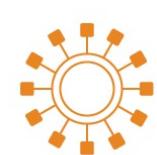
[BROWSE ARTICLES](#)



#### Ideas Exchange

Help improve FME. Suggest, vote, and comment on new features.

[SUGGEST IDEAS](#)



#### FME Hub

Discover new transformers, web connections, and more built by the FME community.

[BROWSE FME HUB](#)

## 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.

## Forums

FME community members post FME-related messages, ask questions, and share in answering other users' questions. Members earn "reputation" and "badges," and there is a leaderboard of the top-participating users. Join the conversation to see how the community helps each other with their FME projects!

## Ideas Exchange

FME development is very much user-driven. The Ideas Exchange gives users the chance to post their ideas for new FME functionality, or improvements to existing functionality, and allows everyone to vote on the proposed ideas. The more votes an idea gets, the more likely it is to be implemented!

## The FME Channel

This [FME YouTube channel](#) is for those demos that can only be properly appreciated through a screencast or movie. Besides this, there are a host of explanatory and helpful videos, including recordings of most training and tutorials.

CONNECT. TRANSFORM. AUTOMATE.

FME Channel  
4,209 subscribers

SUBSCRIBE 4.2K

HOME VIDEOS PLAYLISTS COMMUNITY CHANNELS ABOUT

What is FME

424 views • 2 weeks ago

FME is the data integration platform with the best support for spatial data worldwide.

It enables you to remove data silos by connecting data between 400+ sources. Increase the value of your data by making it usable where, when, and how it's needed. Then, eliminate the manual effort of performing complex, repetitive tasks by

READ MORE

FEATURED CHANNELS

FMEGuru

Dmitri Bagh

Learning and Tutorials

Automating Enterprise Web Services with FME Server	96	16	Tutorial: Getting Started with FME Desktop	4	Getting Started with FME Server
FME Channel	VIEW FULL PLAYLIST	FME Channel	Translate Data Between	FME Channel	Take a Tour of the Web

## Feedback and Certificates

The format of this training course undergoes regular changes prompted by comments and feedback from previous courses.

---

### Course Feedback

#### FME Lizard says...

*There's one final set of questions – and this time you'll be telling me if the answers are correct or not!*

---

Safe Software values feedback from training course attendees, and our feedback form is your chance to tell us what you think about how well we're meeting your training goals.

You can fill in [the feedback form](#) now, but you'll also be reminded by email shortly after your course. Safe Software's partners who carry out training may ask that you fill in a separate form, but you can also use the official Safe Software form if you wish.

---

### Certificates

#### FME Lizard says...

*In order to prove you have taken this training course, a certificate will be emailed automatically to anyone who was logged on for the duration of Safe Software hosted courses.*

## Continue Your Education

Your FME education doesn't stop here! Sign up for another course or follow along with a recorded version. Check our [website](#) often for new live online courses

### FME Desktop Advanced

[FME Desktop Advanced](#) training course builds upon the basic framework of workspace creation in FME Desktop. The course covers topics that are commonly used by all workspace authors who wish to take their FME skills to the next level. Requires FME Desktop Basic or equivalent experience.

### FME Server Authoring

[FME Server Authoring](#) is a key course for FME users intending to author workspaces for use on FME Server. Through hands on exercises, learn the advanced skills, knowledge and terminology that users need to know in order to produce workspaces optimized for FME Server. The training will introduce basic concepts and terminology, help you become an efficient FME Server user, and direct you to resources to help apply the product to your own needs. Requires basic experience with FME Desktop.

### FME Server REST API

[FME Server REST API](#) is a brand new course that teaches FME users how to master the FME Server REST API and build their own web applications to solve data challenges. This course will teach users what the REST API is, how to use it, and how to build custom web applications that leverage the power of FME. Requires basic experience with FME Desktop and FME Server, HTML and Javascript knowledge an asset but not required.

### FME Server Administration

[FME Server Administration](#) involves managing important tasks like optimization, security, and installation. Learn what administration options are available in FME Server and how to modify them to design the most effective deployment for your needs. You'll learn how to monitor the health of FME Server and its components, plus how to troubleshoot problems. Requires advanced FME Server experience.

## Thank You

Thank you for attending this FME training course.

