



FME<sup>®</sup> Desktop  
XML Pathway Training Manual  
Reading XML with FME

FME 2013-SP1 Edition



Safe Software Inc. makes no warranty either expressed or implied, including, but not limited to, any implied warranties of merchantability or fitness for a particular purpose regarding these materials, and makes such materials available solely on an “as-is” basis.

In no event shall Safe Software Inc. be liable to anyone for special, collateral, incidental, or consequential damages in connection with or arising out of purchase or use of these materials. The sole and exclusive liability of Safe Software Inc., regardless of the form or action, shall not exceed the purchase price of the materials described herein.

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.

## **Copyright**

© 1994 – 2013 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.  
Suite 2017, 7445 – 132<sup>nd</sup> Street  
Surrey, BC  
Canada  
V3W1J8

[www.safe.com](http://www.safe.com)

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

## **Trademarks**

FME is a registered trademark of Safe Software Inc.

All brand or product names mentioned herein may be trademarks or registered trademarks of their respective holders and should be noted as such.

## **Documentation Information**

The following is information about this document and the systems used to create it.

Document Name: FME Desktop XML (Reading) Pathway Training  
Updated: May 2013  
FME Version: FME 2013-SP1, Build 13448, WIN32  
Operating System: Windows 7 SP-1, 64-bit.  
Other Applications: Notepad++ v6.3.2, Internet Explorer v10



Introduction.....	5
XML Pathway.....	5
FME Version.....	5
Sample Data.....	5
XML Basics.....	6
What is XML?.....	6
XML Data Structures.....	6
Schemas.....	8
GML.....	12
Supported GML Formats.....	14
GML Schema Documents.....	15
Advanced XML Reading.....	19
XML Reading Methods.....	19
GML Extraction.....	19
XQuery.....	23
xfMaps.....	29
Reading XML Fragments.....	30
Text File Reader.....	30
Data File Reader.....	30
AttributeFileReader Transformer.....	30
Session Review.....	35
What You Should Have Learned from this Session.....	35



## Introduction



*This training material is part of the FME Training Pathway system.*

### XML Pathway

This training material is part of the FME Training XML Pathway.

It contains advanced content and assumes the user is familiar with all concepts and practices covered by the FME XML Pathway Tutorial, and the FME Desktop Basic Training Course.

The course looks at the methods by which XML documents and datasets can be translated and transformed using FME. The focus is on spatial formats – such as GML and GML profiles – and related spatial data XML topics such as metadata.

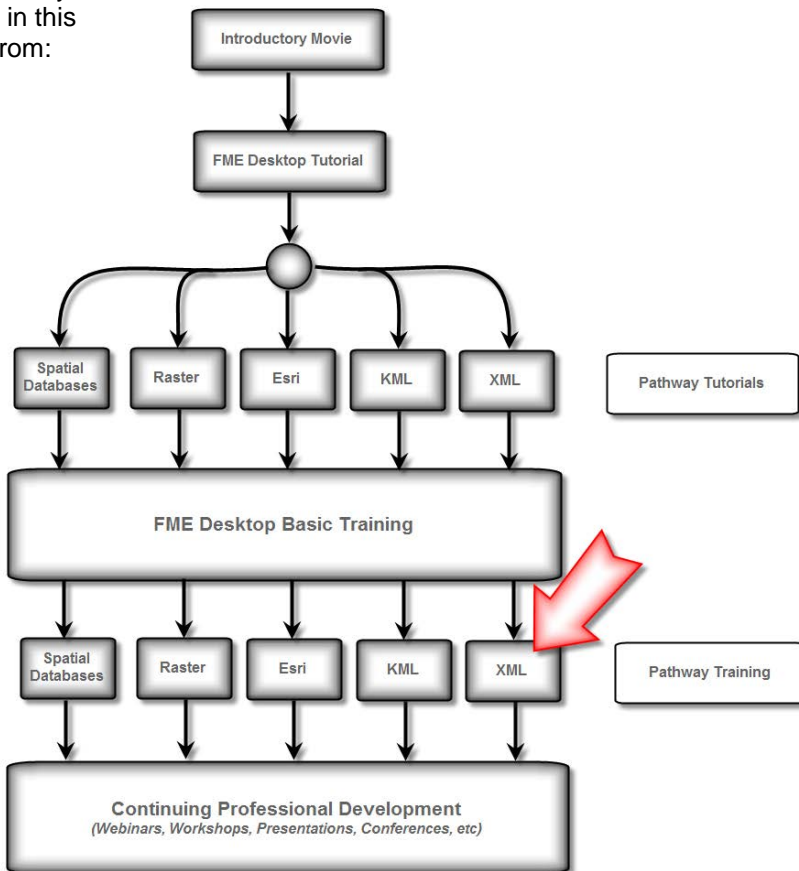
### FME Version

This training material is designed specifically for use with FME2013-SP1. You may not have some of the functionality described if you use an older version of FME.

### Sample Data

The sample data required to carry out the examples and exercises in this document can be obtained from:

[www.safe.com/fmetadata](http://www.safe.com/fmetadata)



## XML Basics



*A review of XML basics*

XML is growing as a language for defining spatial formats and metadata. Because it is such a flexible language any number of differing formats can be – and are – based on XML. It's relatively simple to use because XML documents (datasets) are generally open and self-documenting.

### What is XML?

XML is a markup language. In other words it is used to annotate (mark-up) the contents of a document.

In this simple example the data is a simple text string – Joan – and the XML markup is a set of tags that tell us this is the name of an FME user:

```
<FMEUser>  
  <name>Joan</name>  
</FMEUser>
```

The official specification for XML defines not the names of the tags (how could they possibly know to have an “FMEUser” tag?) but simply how these tags should be structured.



Professor Lynn Guistic says...

*“Think of the XML specification as defining the **grammar** of a language, not the words”*

### XML Data Structures

One challenge to using XML data with GIS (and other spatial archetypes) is that spatial systems are commonly geared to working with relational and “flat” data structures. However, XML documents are object-oriented and often nested to a high degree.

Therefore, much of the challenge of handling XML data is in either:

- Reading an XML document and converting it to a GIS relationship-type structure
- Converting a GIS relationship-type structure and writing it as an XML document.



Professor Lynn Guistic says...

*“Converting XML to a GIS format is an exercise in restructuring the data. For known formats FME does this semantically, without the user needing to be aware”*



Example 1: Reading Pre-defined XML Datasets	
<b>Scenario</b>	FME user; City of Interopolis, Planning Department
<b>Data</b>	GPS data (GPX format); Building Information (CityGML format)
<b>Overall Goal</b>	Inspect XML dataset contents
<b>Demonstrates</b>	Reading XML data with pre-defined FME formats
<b>Starting Workspace</b>	None
<b>Finished Workspace</b>	None

To get started, this example merely calls for reading XML datasets using a pre-defined format.

### 1) Start FME Data Inspector

To inspect this data we'll use the FME Data Inspector, which allows us to visualize both simple and complex XML datasets.

So, start the FME Data Inspector from the start menu.

### 2) Open Dataset

Open the following XML dataset:

**Reader Format**            GPS eXchange Format (GPX)  
**Reader Dataset**        [C:\FMEData\Data\GPS\gps\\_control.gpx](C:\FMEData\Data\GPS\gps_control.gpx)

Query a feature. Notice how it is a simple point feature with a small number of attributes.

Geometry	
IFMEPoint	(-97.6939134831474, 30.3116541467477)
x	-97.6939134831474
y	30.3116541467477

### 3) Open Dataset

Now open the following XML dataset:

**Reader Format**            CityGML  
**Reader Dataset**        <C:\FMEData\Resources\XML\building.gml>

Turn off the display of all feature types except "Building". You can switch to 3D mode and rotate the data to get a proper view of it. Query the building feature.




Notice how it is a lot more complex. It is made up of a BRep (Boundary Representation) solid shape with multiple surfaces and information stored in what are known as FME "traits". This is how FME has handled the conversion between object-oriented and relational.

Whereas the GPS data is virtually "ready to use", the workspace author may have to transform the contents of the CityGML features in order to extract useful information.

**Schemas**

XML documents are often accompanied by a schema document. An XML schema document defines the names and content of the elements used in the XML.



*Professor Lynn Guistic says...*

*“The names of elements can be called the **vocabulary** of the schema“*

In the previously described example:

```
<FMEUser>
  <name>Joan</name>
</FMEUser>
```

...the schema document will explain that the tag “name” is a text string that belongs to the type of feature called “FMEUser”. As an XSD file, it will look something like this:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="FMEUser">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

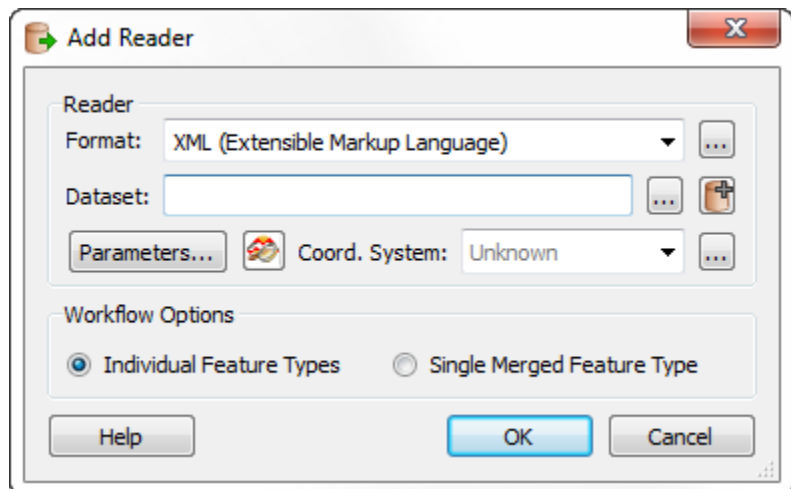
</xs:schema>
```

With this schema a person or application is able to:

- Validate the names (but not the meaning) of the XML document
- Discover the data types used in attributes (string, decimal, integer, date, etc)

**No Schema**

When a particular XML document has no schema (and there is no specific reader or writer for it in FME) there is a basic XML format for handling XML in its raw state.



However, handling schema-less XML is more complex and requires greater input from the workspace author on how features are to be mapped from object-oriented to a relational structure.





Example 2: Reading Schema-less XML	
<b>Scenario</b>	FME user; City of Interopolis, Planning Department
<b>Data</b>	FME Course locations (XML)
<b>Overall Goal</b>	Read XML dataset contents
<b>Demonstrates</b>	Extracting data from a Feature Path
<b>Starting Workspace</b>	None
<b>Finished Workspace</b>	None

When elements are nested inside an XML document, the nesting to a particular element is called a **path**. FME is able to extract features from an XML document by defining the path name of those features.

### 1) Inspect Source Data

Start a text editor application (or other tool for inspecting XML documents).

Open the file: <C:\FMEData\Resources\XML\FMECourses.xml>

Notice how it is an XML document recording the location and dates of upcoming FME training courses. In particular note that the path for an individual training course is called "Course".

```

<?xml version="1.0" encoding="UTF-8"?>
  <FeatureCollection>
    <Course>
      <type>Desktop</type>
      <location>Vancouver, BC</location>
      <activeDate>
        <from>2014-05-22</from>
        <to>2014-05-23</to>
      </activeDate>
      <coord lat="49.2" long="-123.0"/>
    </Course>
    <Course>
      <type>Server</type>
      <location>Austin, TX</location>
      <activeDate>
        <from>2014-11-13</from>
        <to>2014-11-14</to>
      </activeDate>
      <coord lat="30.25" long="-97.75"/>
    </Course>
  </FeatureCollection>

```

### 2) Start FME Data Inspector

Start the FME Data Inspector. Select File > Open Dataset and enter the reader format and dataset as follows:

**Reader Format** XML (Extensible Markup Language)  
**Reader Dataset** C:\FMEData\Resources\XML\FMECourses.xml

Click OK to open the dataset. An error will appear reporting that the XML module in FME halted with an error. The log window will elaborate:

```
The XML format could not be determined by examination.
Try entering a feature path into the "Elements to Match" parameter, specifying
an xfMap, selecting an XRS, or using a more specific reader
```

The problem is that FME does not have any way to know what sort of XML document this file is, nor how to interpret its schema.

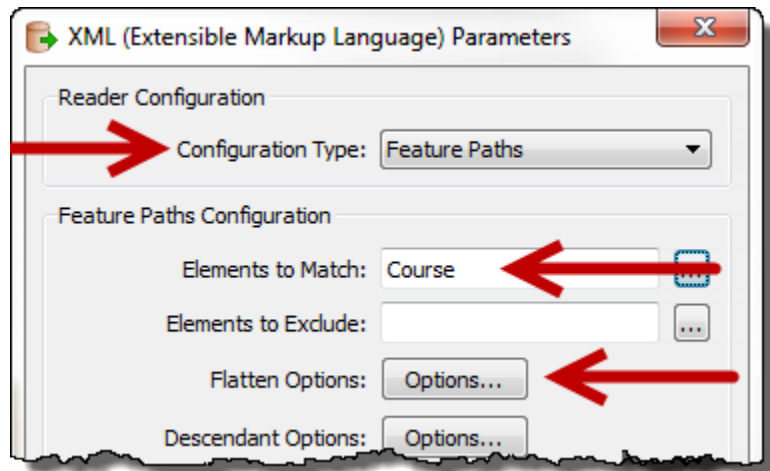
### 3) Re-Open Dataset

Select File > Open Dataset a second time. The format and dataset parameters should exist from before. This time click the button marked **Parameters...**

In the parameters dialog, ensure the Configuration Type is set to Feature Paths.

Enter Course into the Elements to Match field.

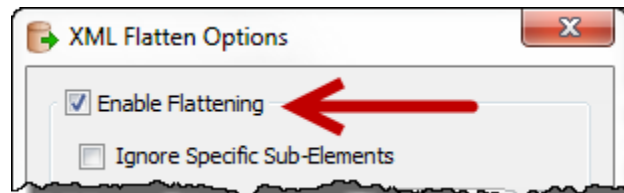
This will cause FME to create a feature for every instance of the Course element.



*This field is case sensitive, so "course" or "COURSE" will not give the correct result!*

Now click the button for Flatten Options.

Flattening is the term used for extracting a set of nested values into FME attributes.

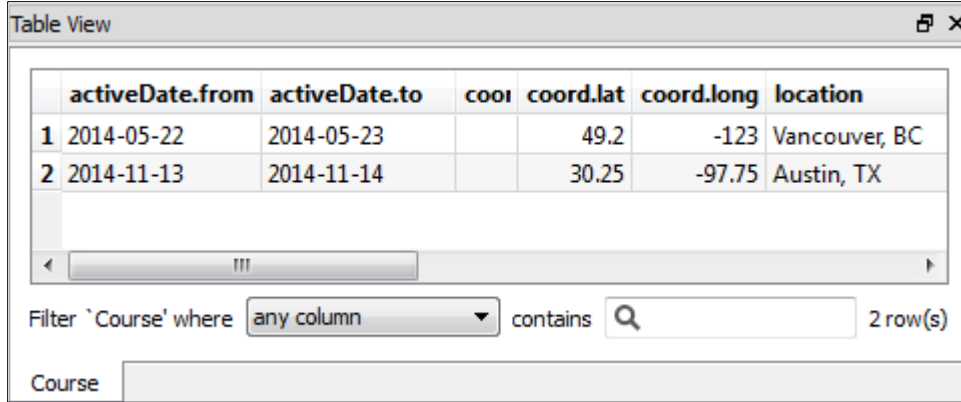


In this dialog simply put a check mark against the parameter labelled Enable Flattening.

Click OK on this dialog, then twice more, to close all dialogs and read the data.

#### 4) Query Data

Being plain XML the data will have been read without geometry, but you'll be able to see the associated attributes in the Table View window.



	activeDate.from	activeDate.to	cool	coord.lat	coord.long	location
1	2014-05-22	2014-05-23		49.2	-123	Vancouver, BC
2	2014-11-13	2014-11-14		30.25	-97.75	Austin, TX

Filter 'Course' where any column contains 2 row(s)

Course



#### Advanced Task

Notice that two of the fields in the data are coord.lat and coord.long.

If you have time, read this XML document with FME Workbench, using a 2DPointReplacer transformer to convert these attributes to true geometry.

Write the data to KML format and inspect the output in Google Earth to confirm it is in the correct location.



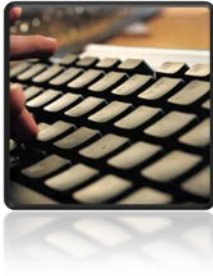
*Professor Lynn Guistic says...*

*"An alternative method for handling XML data like this is to read the XML content into an attribute and then use a transformer to process it.*

*You might do this when the XML is stored in a way that the XML reader can't access it - for example when it is in a database field or a user attribute in a regular spatial dataset, or when the XML is the result of a web query, maybe one carried out by an HTTPFetcher transformer.*

*You could use the XMLFlattener transformer to process it, but this will merely flatten the attributes and not also split the data up into different types. The transformer to use is the XMLFragmenter, which will both divide the data into different features AND flatten the attributes."*

## GML



*A review of GML basics*

The OGC (Open Geospatial Consortium) defines GML as:

*“...an XML grammar defined to express geographical features.”*

In other words, while conforming to the XML specification in terms of the structure used, it defines a unique – and very large – set of elements to cover all spatial geometries.

The GML grammar specifies how to define geographical entities such as:

- Arcs
- Curves
- Linestrings
- Points
- Polygons
- Splines

Like XML, a GML dataset usually includes a schema document, which defines the vocabulary and structure of the dataset.

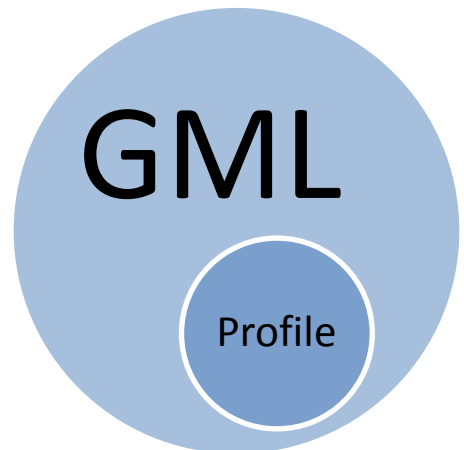
### **GML Profiles**

A GML Profile is a subset of the overall GML specification.

For example, a Point Profile exists in which the supported geometry is limited to points.

Another common profile is the GML Simple Features Profile. This allows a wider range of geometry than just point features, but excludes support for items such as coverages and topology.

Data might be restricted to a profile when its intended use doesn't need to employ the full GML specification.



### **GML Application Schemas**

An Application Schema is an XML schema that describes object (or feature) types that it supports.

Wikipedia notes:

*“For example, an application for tourism may define object types including monuments, places of interest, museums, road exits, and viewpoints in its application schema”*

So the application schema defines real-world items that – in turn – make reference to the geometry types defined in the full GML definition ([or a Profile of GML](#)).

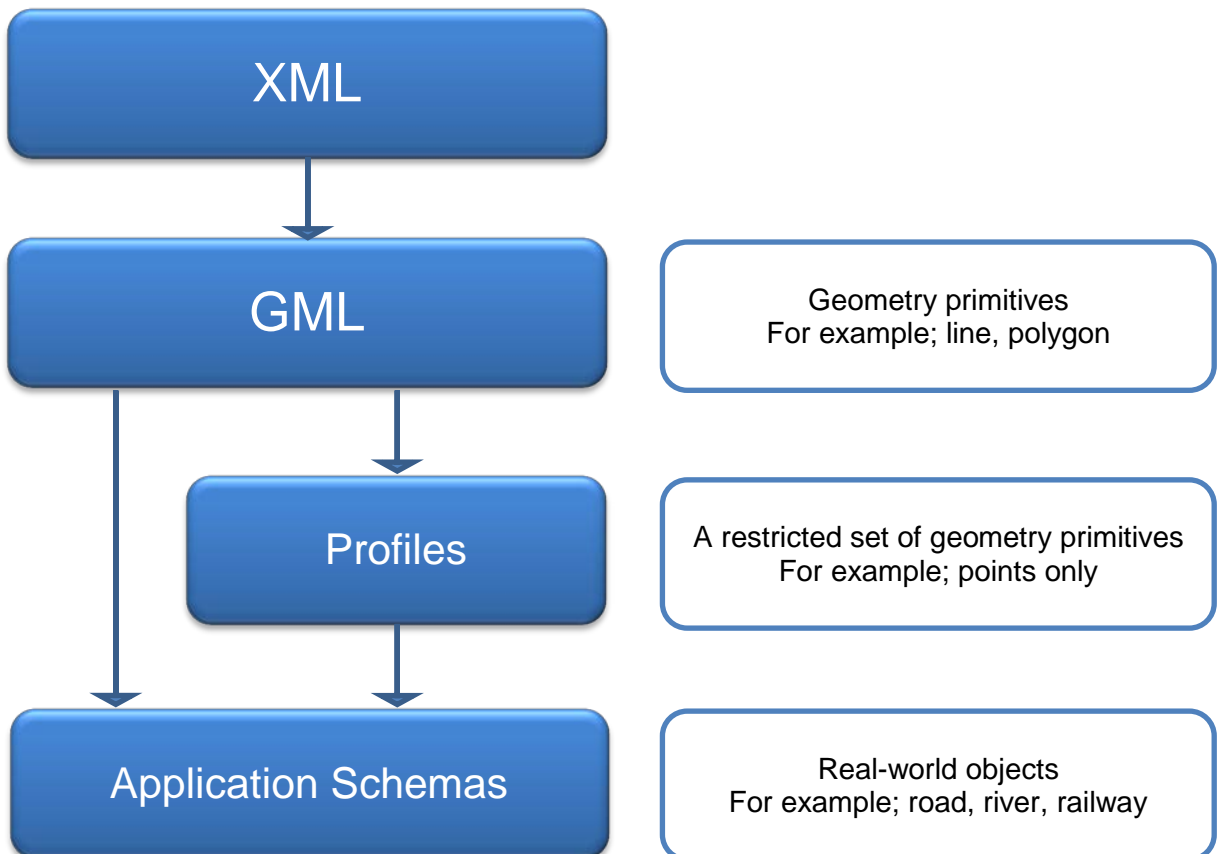
Examples of application schemas are:

- AIXM (Aeronautical Information Exchange Model)
- CityGML (GML for 3D City/Building Information Models)
- Ordnance Survey MasterMap GML (UK National Mapping Agency data)
- TigerGML (US Census data)



*Professor Lynn Guistic says...*

*“You might think of the whole XML/GML terminology like this diagram:”*



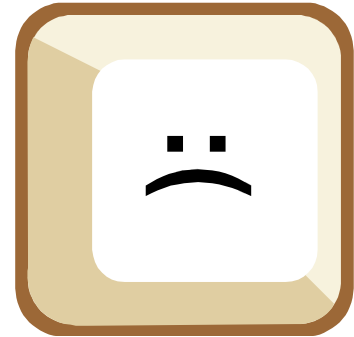
## Supported GML Formats

So, often a GML document is just GML, pure and simple.

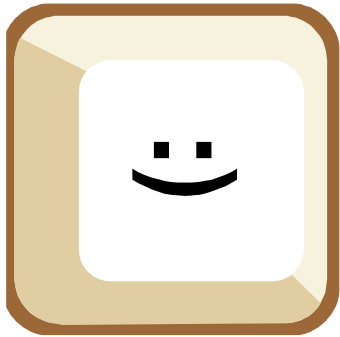
However, sometimes a GML document adheres to a defined Application Schema; where real-world objects such as traffic lights, roads, and parks are defined and mapped to GML primitives such as points, lines, and polygons.

And sometimes a dataset that you think would be GML, can be a hybrid format that just embeds geometry using GML, but also uses aspects of XML outside of the GML vocabulary!

Throw in the ability to base a document on a sub-set (profile) of GML types, and it becomes very easy to be overwhelmed by all these differences.



Happily FME resolves much of this confusion.



For the most well-known and well-documented formats it provides a set of pre-made readers and writers, so that the workspace author does not need to inspect the raw XML to understand how the format operates.

This is possible because, when there is a known application schema, FME has the information required to turn the raw XML data into proper spatial features; with the correct geometry and feature type names.

Similarly, for known hybrid formats, FME understands the specification and can interpret it without user intervention.

For plain GML, or where the application schema is not known, FME offers the following format:

- GML (Geography Markup Language)

Some examples of specific GML Application Schema formats are as follows:

- Aeronautical Information Exchange Model 5 (AIXM 5)
- CityGML
- Dutch TOP50NL GML
- German AAA GML Exchange Format (NAS)
- INSPIRE GML

Some examples of hybrid “GML” formats are as follows:

- GeoRSS/RSS Feed
- Google Earth KML
- GPS eXchange Format (GPX)
- OpenStreetMap (OSM) XML
- Trimble JobXML

**GML Schema Documents**

The name and location of the schema that relates to a GML dataset is usually stored in the document header. The schema may be defined as just a name, as a full URL, or it may not be specified at all. FME's 'vanilla' GML reader will examine the schema setting in the file.

**Fetching a Schema**

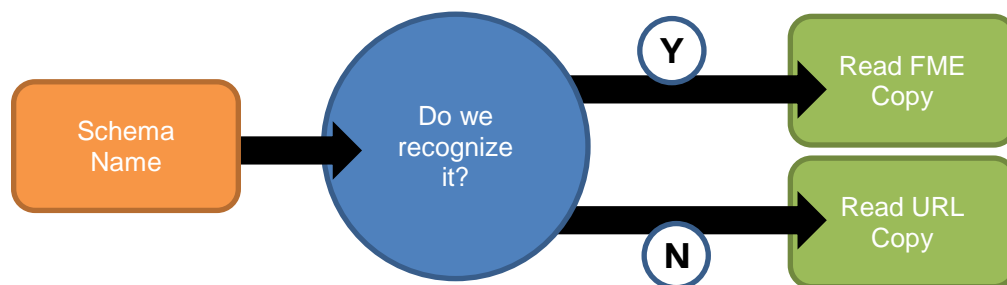
If the document contains a schema declaration then FME knows the schema to use.

For a schema defined purely by a name (no path) then FME expects that schema to be present locally to the GML document; for instance, an XSD file with the same rootname as the GML.

For a schema defined with a full path (such as a URL) then FME can fetch the schema document from that location.

However, FME does have some schemas built into the product, so it will first check to see if the type of GML is known to it and, if so, it uses an inbuilt copy of the schema.

Only when FME does not recognize the type of GML, will it fetch the schema from the URL.

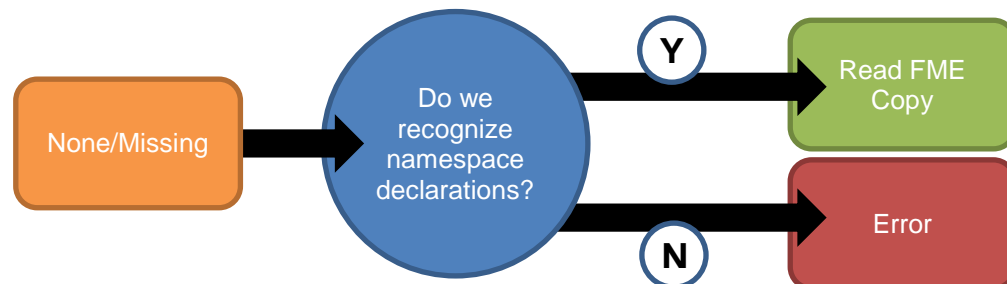


**No Schema**

If the dataset does not have a schema declaration, or it does but the defined schema is not available then, once more, FME will try to identify the dataset from the namespace declarations.

If the type of data can be identified then FME will use a built-in schema to read it.

If the type cannot be identified then the process will end with an error.





Example 3: Reading GML Datasets	
<b>Scenario</b>	FME user; City of Interopolis, Planning Department
<b>Data</b>	EPA Facilities, Inspire Geographic Names
<b>Overall Goal</b>	Read GML dataset contents
<b>Demonstrates</b>	Schemas and GML Schema parameters
<b>Starting Workspace</b>	None
<b>Finished Workspace</b>	C:\FMEData\Workspaces\PathwayManuals\XML3a-Complete.fmw C:\FMEData\Workspaces\PathwayManuals\XML3b-Complete.fmw

In this example we'll read some basic GML datasets.

**1) Start FME Data Inspector**

Start the FME Data Inspector from the start menu. This time we'll open the 3D dataset from example 1 in this manual. However, instead of CityGML as the source format, simply set GML:

**Reader Format**            GML (Geography Markup Language)  
**Reader Dataset**        C:\FMEData\Resources\XML\building.gml

Notice that FME reads the data correctly. The log window tells us that FME has identified this type of data from its schema and is therefore using the correct application schema.

Opening the CITYGML reader with source dataset  
 'C:/FMEData/Resources/XML/building.gml'

Furthermore FME is using a local copy of the schema, and does not need the remote version:

URI 'http://schemas.opengis.net/citygml/1.0/cityGMLBase.xsd' mapped to  
 'file:///C:/apps/FME2013/xml/schemas/schemas.zip/CityGML/CityGML/1.0.0/CityGML/cityGMLBase.xsd'

**2) Start FME Workbench**

Let's look at a second dataset. This time start FME Workbench and begin with a blank canvas. Add a reader to read the following dataset:

**Reader Format**            GML (Geography Markup Language)  
**Reader Dataset**        C:\FMEData\Data\EPA\EPAMonitoredFacilities.gml

A dialog will pop up asking you which Feature Types (layers) to include. Select all of them and click **OK**.



### 3) Connect Inspector

Connect Inspector transformers to the GML reader's four Feature Types and run the workspace.

If you look in the log file you will see a message like:

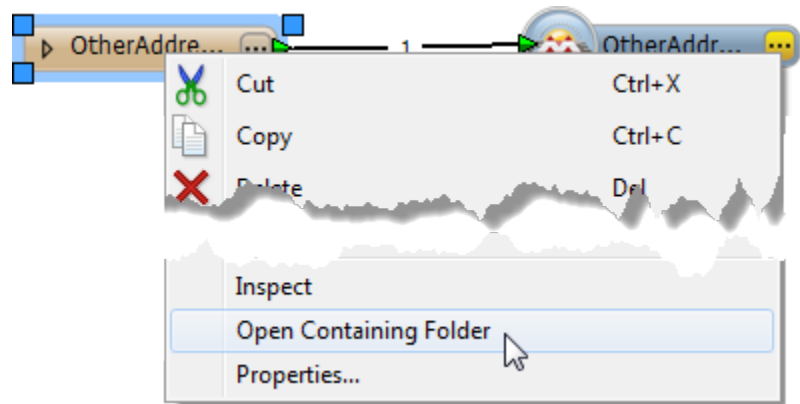
```
Parsing schema document 'file:///C:/FMEDData/Data/EPA/EPAMonitoredFacilities.xsd'
```

This shows that FME knows that the schema is stored in an XSD file with the same filename as the GML document.

### 4) Open Directory

Right-click on one of the reader feature types and choose the option "Open Containing Folder".

You will see the GML document and its schema (XSD) file.









Rename the XSD file to EPAMonitoredFacilitiesSchema.XSD and re-run the workspace.

This time the translation will fail. FME cannot find the schema document – because the name doesn't match – and is unable to otherwise identify the GML document as a known application schema.

### 5) Set Schema Manually

In the Navigator window expand the parameters for the GML reader. One of the parameters will be called "Application Schema".

Double-click the parameter and use its file browser to select EPAMonitoredFacilitiesSchema.XSD. Re-run the workspace. It will now run to completion again.

-  Substitute Close List Brace: <not set>
-  Substitute Element List Separator: <not set>
-  Application Schema: C:\FMEDData\Data\EPA\EPAMonitoredFacilitiesSchema.xsd 
-  Validate GML Dataset File: No
-  Ignore xsi:schemaLocation in Dataset: No

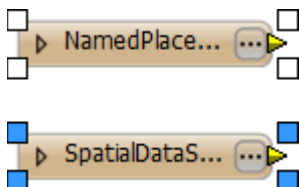
This demonstrated how to set a schema document when it is not included with the GML document. This parameter can also be set as a URL if necessary.

### 6) Create Workspace

For our final dataset, close the previous workspace and start again with an empty canvas.

Add a reader to read the following dataset:

**Reader Format** GML (Geography Markup Language)  
**Reader Dataset** C:\FMEData\Resources\XML\geographicNames.gml



### 7) Connect Inspector

Connect Inspector transformers to the GML reader's two Feature Types and run the workspace.

The workspace will run to completion, despite the fact that there is no related XSD file. In fact, if you open the GML document in a text editor you will see that no schema is defined.


This works because FME has recognized the type of GML document from its contents, and applied copies of the schema that it has locally:

```
GML Reader> - No explicit XML Schema specified (through the XSD_DOC keyword)
nor an xsi:schemaLocation or xsi:noNamespaceSchemaLocation were found in dataset
'C:\FMEData\Resources\XML\geographicNames.gml',

determining schema files from the dataset XML namespace declarations...

<GML Reader> - XML Namespace 'urn:x-inspire:specification:gmlas:BaseTypes:3.2' mapped to
schema 'BaseTypes.xsd'

<GML Reader> - XML Namespace 'urn:x-inspire:specification:gmlas:GeographicalNames:3.0'
mapped to schema 'GeographicalNames.xsd'
```



*Professor Lynn Guistic says...*

*“The lesson here is that a schema is ALWAYS needed to be able to read a GML dataset. Often FME is able to read a schema-less GML dataset by interpreting the namespaces used, but if not the translation will fail with an error.*

*To read a GML dataset without a schema requires the user to use the XML reader with Feature Paths defined as in the previous example”*



### Advanced Task

When you know the geographicNames document is meant to represent Italy, the workspace output appears a little strange. The Lat/Long axes are being confused.

Experiment with the parameter *GML SRS Axis Order* in the Navigator window to fix this problem.

## Advanced XML Reading



*XQuery and XFMaps*

### XML Reading Methods

There are multiple ways to read XML (or GML) data, depending on whether a formal reader already exists for that profile, what data needs to be extracted, and exactly how the data is to be converted to a relational structure.

- Pre-defined Formats and Schemas
- XML Reader: Feature Paths
- XML Reader: GML Extraction
- XML Reader: XQuery
- XML Reader: xfMaps

We've already covered pre-defined formats and feature paths so let's look at the remaining three – more advanced – items on this list.

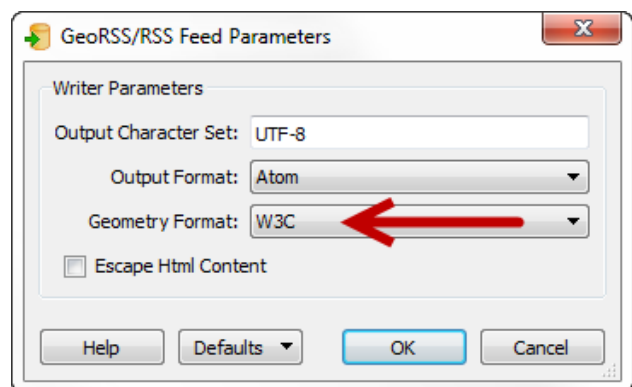
### GML Extraction

As noted, some formats can be a hybrid of XML and GML, just storing geometry as GML but using other XML grammar to store attribute and other content.

For example, KML uses GML to define its feature geometry, but the remainder of a KML document is not pure GML; it concerns styling and symbology markup that is totally unrelated.

Conversely, GeoRSS can store its geometry in a number of ways; not just as GML, but also as GeoRSS-Simple, an OGC application schema called GeoRSS GML, or as W3C GeoRSS.

A parameter exists on the GeoRSS writer in FME, to allow the user to select which of these types of data to write:



Other formats (or application schemas) work on a restricted set of GML grammar without formally defining it as a GML profile. The German AAA GML Exchange Format (NAS) is one example.

FME allows GML geometry to be extracted from an XML dataset, using the XML reader and a *GeometryReplacer* transformer.



Example 4: Extracting GML from an XML Dataset	
<b>Scenario</b>	FME user; City of Interopolis, Planning Department
<b>Data</b>	Schools XML/GML
<b>Overall Goal</b>	Extract geometry and attributes from the XML document
<b>Demonstrates</b>	Extracting GML data embedded in an XML document
<b>Starting Workspace</b>	None
<b>Finished Workspace</b>	C:\FMEData\Workspaces\PathwayManuals\XML4-Complete.fmw

In this example, we have been presented with an unknown dataset. It's believed to be XML and with contents that include geometry in GML form. Other than that its format is a complete mystery.

**1) Start FME Workbench**

Start FME Workbench and begin with a blank canvas.

**2) Add Reader**

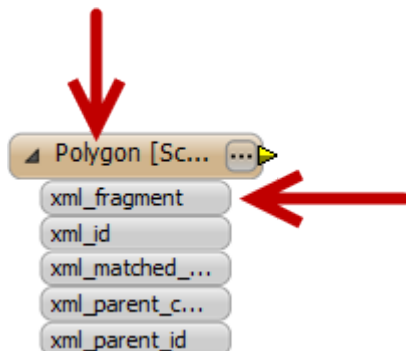
Add a reader to read the following dataset:

**Reader Format** XML (Extensible Markup Language)  
**Reader Dataset** C:\FMEData\Resources\XML\Schools.xml

**Parameters**  
 Elements to Match Polygon  
 Flatten Options Enable Flattening = NO (unchecked)

The parameters are best set in the Add Reader dialog by clicking the parameters button. Ensure – as above – that the flatten options are turned off. Enter “Polygon” as the Element to Match. This means we are extracting data that is tagged as <Polygon>.

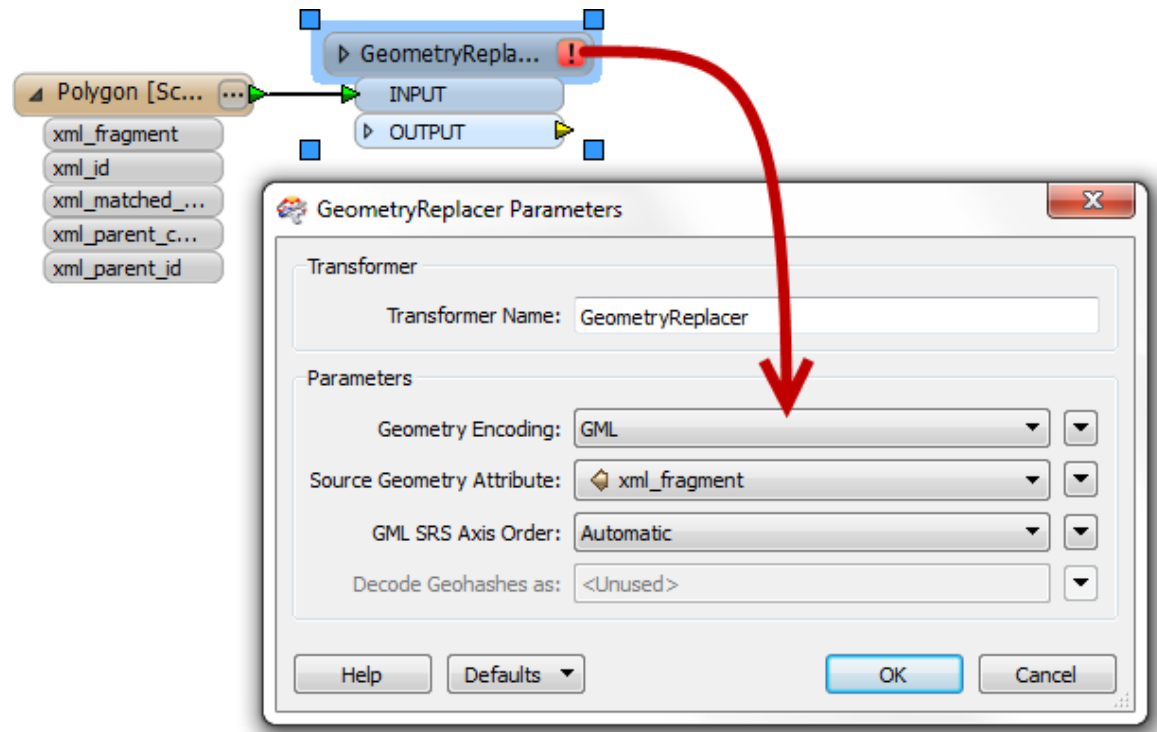
When the reader has been added, notice how the Feature Type carries the name of the XML element we matched, and that the XML content for the element has been extracted into an attribute called xml\_fragment



### 3) Add GeometryReplacer

Add a GeometryReplacer transformer.

Open the parameters dialog and set the Geometry Encoding parameter to *GML*. Also set the Source Geometry Attribute parameter to *xml\_fragment*

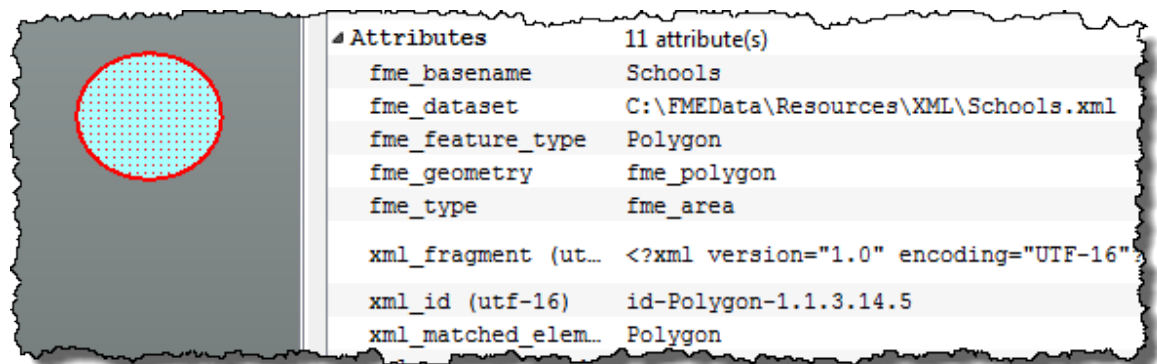


Click **OK** to close the dialog and accept these changes.

### 4) Connect Inspector

Connect an Inspector transformer to the GeometryReplacer. Save and then Run the workspace.

Notice that the geometry has been extracted from the XML but, at the moment, there is no other data except for a handful of Format Attributes.

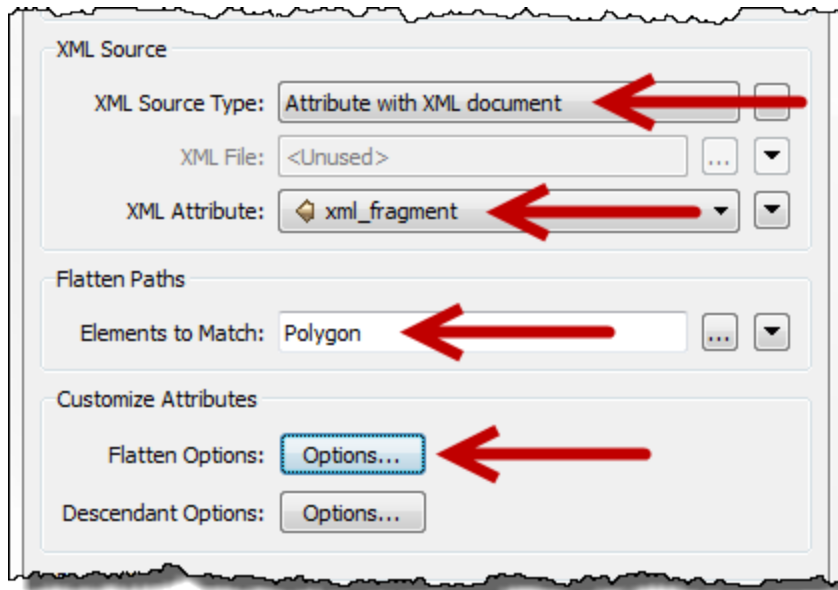


### 5) Add XMLFlattener

Back In Workbench, add an *XMLFlattener* transformer.

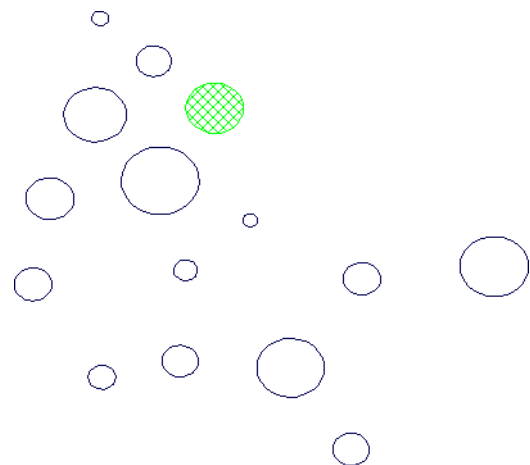
Set the parameters as follows:

XML Source Type	Attribute with XML document
XML Attribute	xml_fragment
Elements to Match	Polygon
Flatten Options	Enable Flattening = YES (checked)



### 6) Rerun Workspace

Rerun the workspace. Query one of the ellipsoid features. Notice how it now has a set of attributes including *name* and *description*.



#### Question:

Can you now tell what format of data you are really dealing with here? The namespace declaration – now exposed as an attribute called *Polygon.xmlns* – should give you a very big clue!

Although the example is a little contrived, this technique will come in handy for handling datasets that are, genuinely, XML documents with GML geometry.

## XQuery

XQuery is a commonly used XML language designed to query collections of XML data. XQuery is described as being to XML what SQL is for databases.

XQuery can be used to extract attributes of XML/GML elements such as `xml_id` or `gml_id`, CRS, and geometry traits. It is particularly useful if you want to extract or update specific elements from an XML stream or document, but don't necessarily want to transform or flatten the entire structure.

Where XQuery may end up being a lot more work than the other techniques discussed in this chapter is converting from XML to or from a relational or standard GIS data representation.

FME's XQuery transformers are as follows:

- XQueryUpdater Updates XML document using XQuery Update expressions
- XQueryExploder Generates new features with XQuery expressions
- XQueryExtractor Extracts XML into feature attributes using XQuery expressions

FME extends XQuery with various functions to retrieve and set values from within an XQuery script run under FME. Some of these functions are as follows:

- `fme:has-attribute(<string>)`  
Returns a boolean (true/false) value if the current feature has the specified attribute
- `fme:get-attribute( <string> [<default value>] )`  
Returns the value of an attribute
- `fme:set-attribute( <string>, <value> )`  
Sets the value of a feature attribute

## XQuery and Namespaces

A namespace is a way to uniquely identify different elements in the same XML document.

For example, a dataset might contain roads and rivers, each with an attribute called ID. Namespaces would be used to prevent confusion between the roads ID and the rivers ID.

XQuery uses namespaces for this reason, and they must be declared in the query content.

As a simple example:

### XML:

```
<roads:feature>                                <rivers:feature>
  <ID>A123</ID>                                  <ID>B345</ID>
</roads:feature>                                </rivers:feature>
```

### XQuery:

```
declare namespace x = "roads";
string(//x:ID)
```

Without the namespace declaration there would be no result because XQuery could not be sure to which namespace `string//ID` should match to.



Example 5: Reading XML with XQuery	
<b>Scenario</b>	FME user; City of Interopolis, Planning Department
<b>Data</b>	FME Course locations (XML)
<b>Overall Goal</b>	Extract course information using XQuery
<b>Demonstrates</b>	Reading XML data with XQuery statements
<b>Starting Workspace</b>	None
<b>Finished Workspace</b>	C:\FMEData\Workspaces\PathwayManuals\XML5a-Complete.fmw C:\FMEData\Workspaces\PathwayManuals\XML5b-Complete.fmw

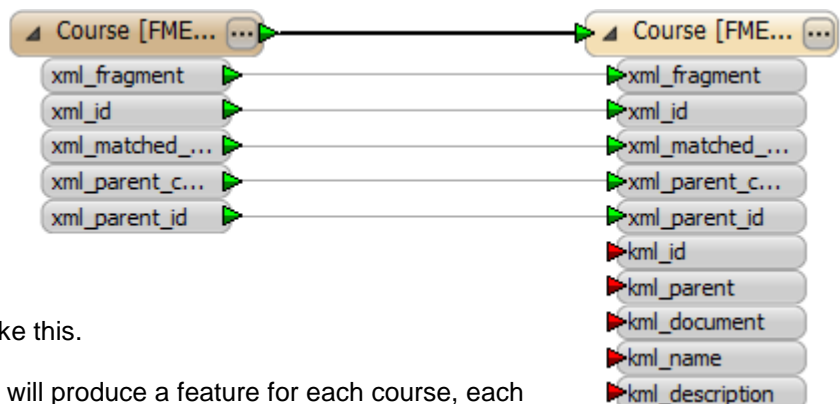
The task here is to extract data from an XML document using XQuery. The document holds information about upcoming FME training courses.

**1) Start FME Workbench**

Start FME Workbench. Open the Generate Workspace dialog.

Set up a translation as follows:

- Reader Format** XML (Extensible Markup Language)
- Reader Dataset** C:\FMEData\Resources\XML\FMECourses.xml
- Reader Parameters**
- Elements to Match Course
- Flatten Options Enable Flattening = NO (unchecked)
- Writer Format** Google Earth KML
- Writer Dataset** C:\FMEData\Output\DemoOutput\FMECourses.kml



The workspace will look like this.

When run, the workspace will produce a feature for each course, each of which contains an *xml\_fragment* attribute that we'll be working with.



### 2) Add XQueryExtractor

Add an XQueryExtractor transformer between the Reader and Writer Feature Types, connecting only the QUERY\_RESULTS output port.

Open the parameters dialog.

The XQuery Input will be an XQuery expression, equivalent to a 'select' or 'where' clause.

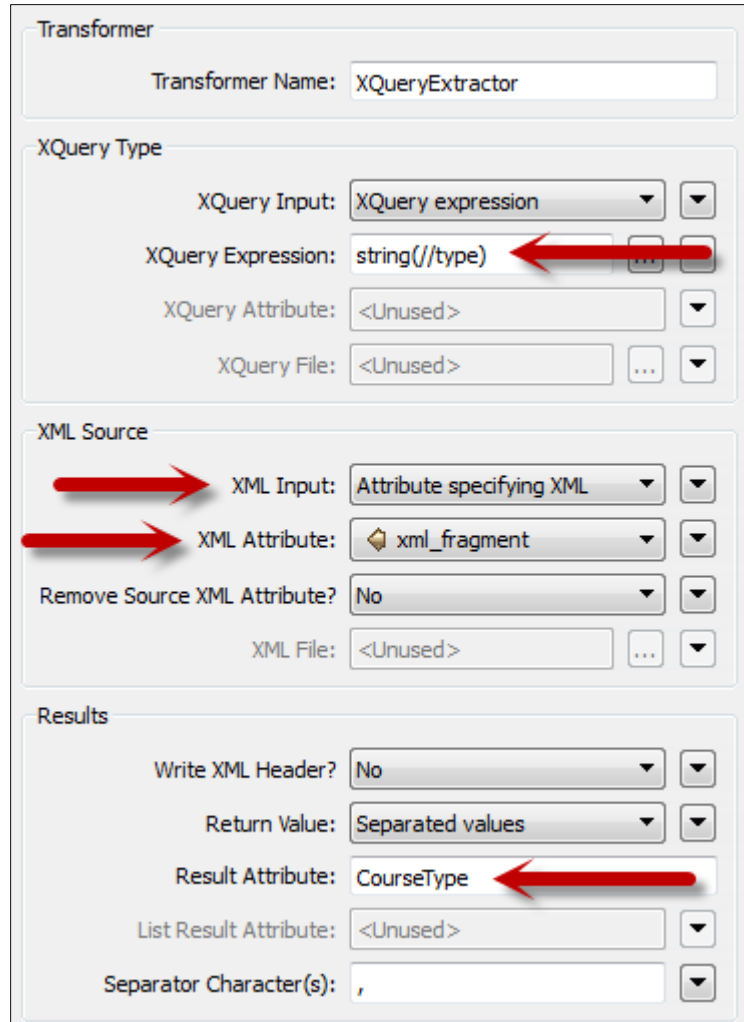
In this case set the expression to: `string(//type)`

The XML Input will be from an attribute, for which you should select `xml_fragment`

Set the Result Attribute to `CourseType`

In other words, select the `Course Type` from `xml_fragment` and put the result into an attribute called `CourseType`

If you wish, you can connect the workspace to an Inspector and run it, to see what the output is.



### 3) Add Another XQueryExtractor

Each XQuery transformer lets us extract another set of information from the source data. So, add one more XQueryExtractor transformer, connected to the previous.

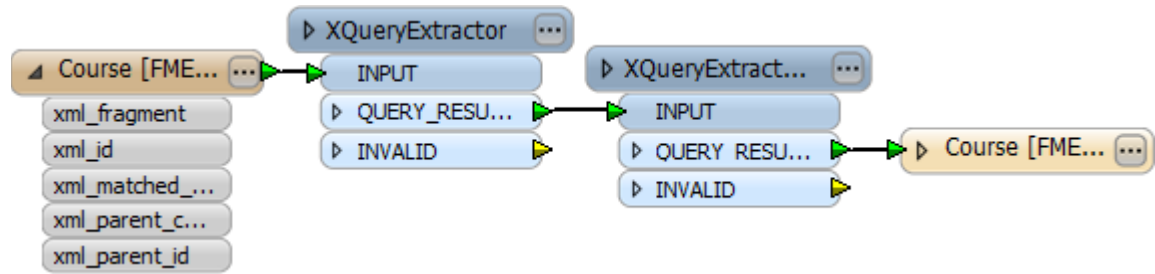
Their parameters should be set to:

XQuery Expression	<code>string(//location)</code>
XML Attribute	<code>xml_fragment</code>
Result Attribute	<code>CourseLocation</code>

These two transformers return the values of the course type and location elements, as shown in the FME Data Inspector.

Attributes		16 attribute(s)
__wb_out_feat_type__		Course
CourseLocation (utf-8)		Austin, TX
CourseType (utf-8)		Server
fme_basename		FMECourses

The workspace will now look like this:



#### 4) Add One More XQueryExtractor

The previous XQueryExtractors used basic XQuery to extract information. However, there are FME-specific XQuery functions available to carry out similar actions.

Add one more XQueryExtractor transformer, connected to the previous. This will be used to extract the coordinates (Lat/Long) of each training course location. But this time we will use FME-specific XQuery expressions.

Open the parameters dialog. For the XQuery expression, open the editor dialog (by clicking the [...] button).

Enter the expression as:

```
fme:set-attribute( "CourseLat",string(//coord/@lat) ),
fme:set-attribute( "CourseLong",string(//coord/@long) )
```

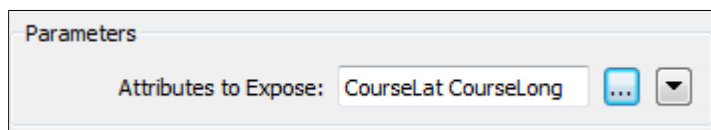
“fme:set-attribute” is the FME XQuery function. It says to create an attribute (called CourseLat) using the value from the regular XQuery expression (//coord/@lat)

Again, the XML source parameters are:

XML Input	XML Attribute
XML Attribute	xml_fragment

The Result Attribute is not necessary (because the FME XQuery function already defines it), but we do need to expose the new attributes in Workbench. This is because FME can't tell what attributes are going to be created until run-time.

Choose the Attributes to Expose parameter and enter CourseLat and CourseLong as the attributes to expose. This will return and expose the values for latitude and longitude.



Re-connect the Inspector transformer and re-run the workspace to prove it is correct.



Professor Lynn Guistic says...

“One difference between the expressions in step 3 and step 4 is a subtle one.

The @ symbol denotes that the information is stored as an “attribute” of the coord element. Without it the expression denotes a “child” element.

To be absolutely clear, use //coord/@lat for this XML:

```
<coord lat="12" >
```

...whereas you would use //coord/lat for this XML:

```
<coord>
<lat>12</lat>
</coord>
```

So now you know!”

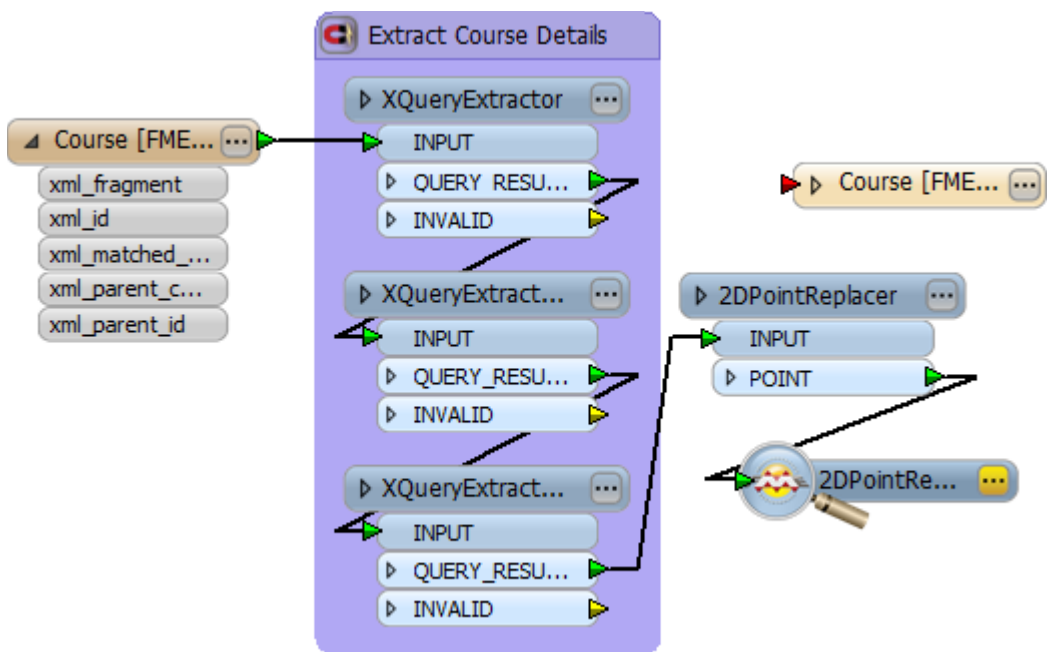
### 5) Add 2DPointReplacer

Add a 2DPointReplacer transformer and use it on the attributes CourseLat and CourseLong to turn the feature from a null-geometry to a point geometry.

Remember, longitude is the X coordinate, latitude the Y!



The workspace will now look like this:

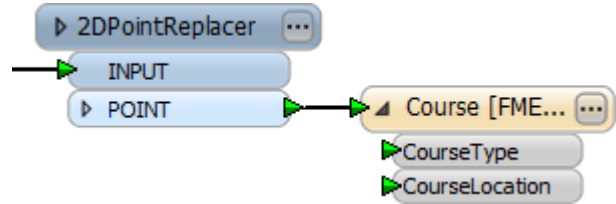


### 6) Edit/Map Schema

Remove the xml attributes and any exposed format attributes from the Writer (KML) schema, as they are not really needed.

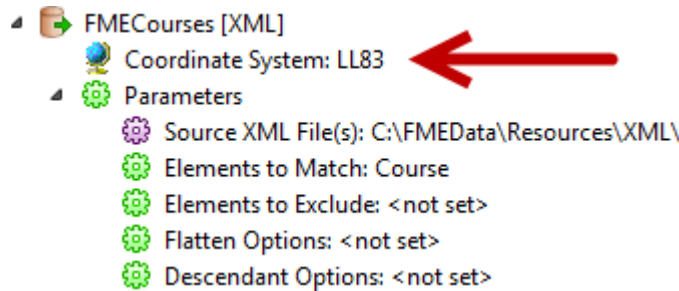
Add two new attributes to the schema to store course type and course location.

Map the XQuery-created attributes to their correct feature type attributes.



### 7) Set Coordinate System

As a final step, set the coordinate system for the XML Reader to be LL83. If a coordinate system is not set then the KML writer will reject the output.



An alternative method would be to use the *CoordinateSystemSetter* transformer.

### 8) Run Workspace

Run the workspace and inspect the output dataset in Google Earth, to prove it is correct.



#### Advanced Task

If you are interested in KML, use the KMLStyler and KMLPropertySetter transformers to give a better symbology to the output features. This might include a new icon and use of an attribute for the KML name.

If you want to experiment more with XQuery, use more XQueryExtractor transformers to retrieve the values for the course dates. You will need to read Prof Lynn Guistic's comments to determine whether these are attributes or child elements, then act accordingly.

If you are interested in both KML *and* XQuery, then take the course dates and use them to set a KML timespan element! A timescale control bar in Google Earth will show whether you have been successful.



## xfMaps

xfMaps are an FME-specific document for defining how paths in an XML document are mapped to features within FME.

Remember that XML data is often nested, whereas FME uses a “flat” data structure. When converting XML data FME needs to know how each level of the hierarchy is to be handled.

A schema records the vocabulary of the XML document (how it is structured and with what names) but – unsurprisingly – there is no XML standard for mapping data structures to FME!

Therefore, you might need to use an xfMap when dealing with:

- Features within features
- Aggregates
- Multi-Records/Geometries
- Inheritance
- Dynamic Schemas

For example, in this extract from the FME course calendar, there are three courses scheduled to go ahead in Vancouver:

```
<Course>
  <type>Desktop</type>
  <location>Vancouver, BC</location>
  <coursedate> 2014-05-22</coursedate>
  <coursedate> 2014-11-13</coursedate>
  <coursedate> 2014-12-21</coursedate>
</Course>
```

...using the flatten option will give one record with a list of course dates – but perhaps you want one feature for each course. An xfMap would allow you to map that as required.

As another example, this entry from the FME course catalog:

```
<CourseCatalog>
  <course>
    <type>Desktop</type>
  </course>
  <course>
    <type>Desktop</type>
    <subtype>Advanced Raster</subtype>
  </course>
</CourseCatalog>
```

...there perhaps needs to be a different action when there is a course subtype – but what? An xfMap will help define how to handle this data.

Much effort in recent FME development for XML support has been expended to make it unnecessary for you to work with xfMaps. However, since they are core to how FME processes XML, it is certainly worth being aware of them, and having some idea as to how they work.

## Reading XML Fragments



*This section covers FME's ability to read raw XML data and preserve it in an attribute. There are several reasons why you would want to.*

Sometimes it's necessary to read XML intact – whether it be just a portion of the document or the entire document – and not try to interpret it, flatten it, or translate it to another format.

A good example of this is metadata. Often metadata is stored as XML. When you translate and transform a dataset to which the metadata refers, you will want to update the metadata contents.

Rather than try to interpret and extract features from this XML, it's far easier to read the relevant fragment, update that fragment, and write it back.

Reading data this way can be achieved with a number of different pieces of functionality.

- Text File Reader
- Data File Reader
- AttributeFileReader Transformer

### **Text File Reader**

The Text File reader allows the user to read the contents of a text file.

Normally the object is to read one line at a time, from a file where each line is a separate record.

However, there is also a parameter called “Read Whole File at Once” that allows the entire contents of a file to be read into a single attribute. Obviously this can be of use to read an entire XML document.

### **Data File Reader**

The Data File reader, like the Text File reader, can be used to read the contents of a file.

The difference is that the Data File reader is *intended* for non-text files, but that still doesn't affect its ability to read an XML document.

Again there is a parameter called “Read Whole File at Once” that allows the entire contents of a file to be read into a single attribute.

### **AttributeFileReader Transformer**

This transformer reads the contents of a file into a single attribute. It's sometimes used to read photographic images (or other raster files) to insert into a database but, again, it is perfectly acceptable for use in reading an XML document.

The useful thing about this transformer is that it can accept the filename from an attribute; therefore a user can provide a dynamic list of files to read, rather than hard-coding them all.



Example 6: Updating XML Metadata	
<b>Scenario</b>	FME user; City of Interopolis, Planning Department
<b>Data</b>	Cycle Routes (Shape) and metadata (XML)
<b>Overall Goal</b>	Update cycle metadata with XMLUpdater
<b>Demonstrates</b>	Reading entire XML documents
<b>Starting Workspace</b>	None
<b>Finished Workspace</b>	C:\FMEData\Workspaces\PathwayManuals\XML6-Complete.fmw

The city has a Shape dataset of cycle routes. Metadata for SHAPE files is stored as a separate XML document. The task is to update one of these metadata documents.

### 1) Inspect Data

Take a look at the reader dataset's existing metadata using an XML browser or text editor. The file can be found at: <C:\FMEData\Data\Transit\BikeRoutes\BikeRoutesH.shp.xml>

Notice the tag <themekey> :

```
<themekey>Austin City Data</themekey>
```

It is this that we wish to update to say Austin City Data – Bike Routes.

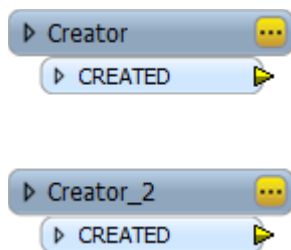
### 2) Start Workbench

Start FME Workbench and begin with a blank canvas.

### 3) Add Creators

Place two Creator transformers, each to create one null feature.

One feature will be used to trigger the metadata reading/writing process; the other will be used to trigger the XML update operation.



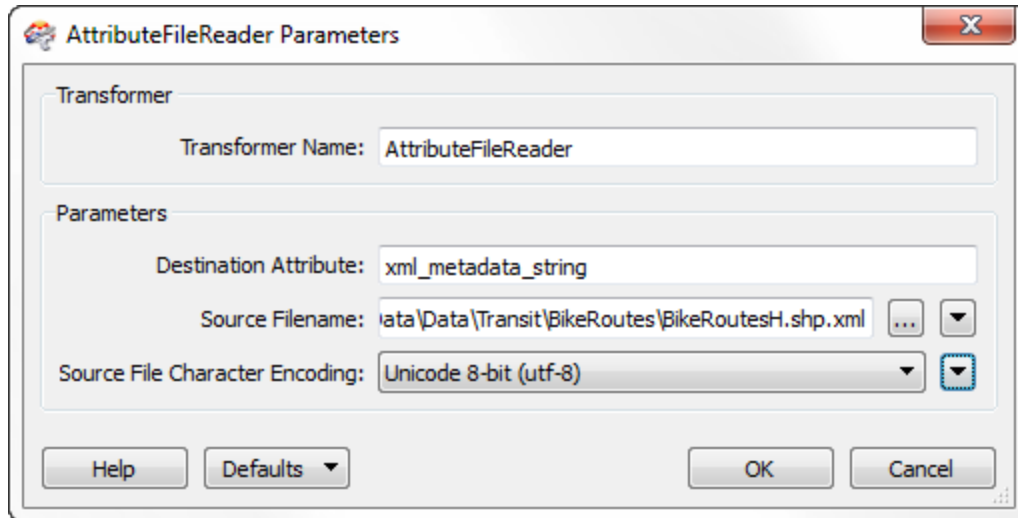
By default, a Creator transformer creates a single null feature, so here two transformers can be placed, and the parameters left untouched.

#### 4) Add AttributeFileReader

Add an *AttributeFileReader* transformer after one of the *Creator* transformers. An *AttributeFileReader* reads the contents of a file and stores it in an attribute.

Open the parameters dialog and set the source filename to be the XML Metadata document associated with the source shape dataset (*C:\FMEData\Data\Transit\BikeRoutes\BikeRoutesH.shp.xml*)

Set the destination attribute to be called *xml\_metadata\_string*  
 Set the source encoding to be Unicode 8-bit (utf-8)

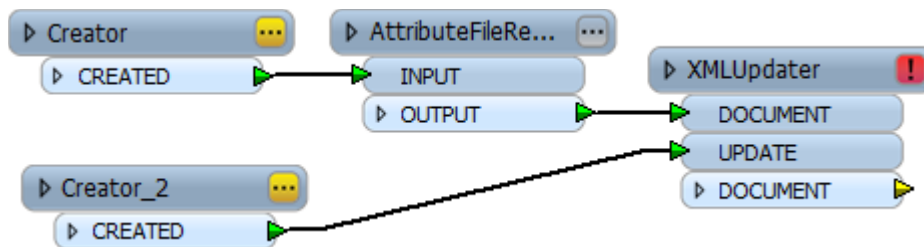


#### 5) Add XMLUpdater Transformer

Add an *XMLUpdater* transformer after the *AttributeFileReader*.

The input port to connect is DOCUMENT as the incoming feature contains the xml document.

Connect the second – as yet unused – Creator transformer to the UPDATE port.





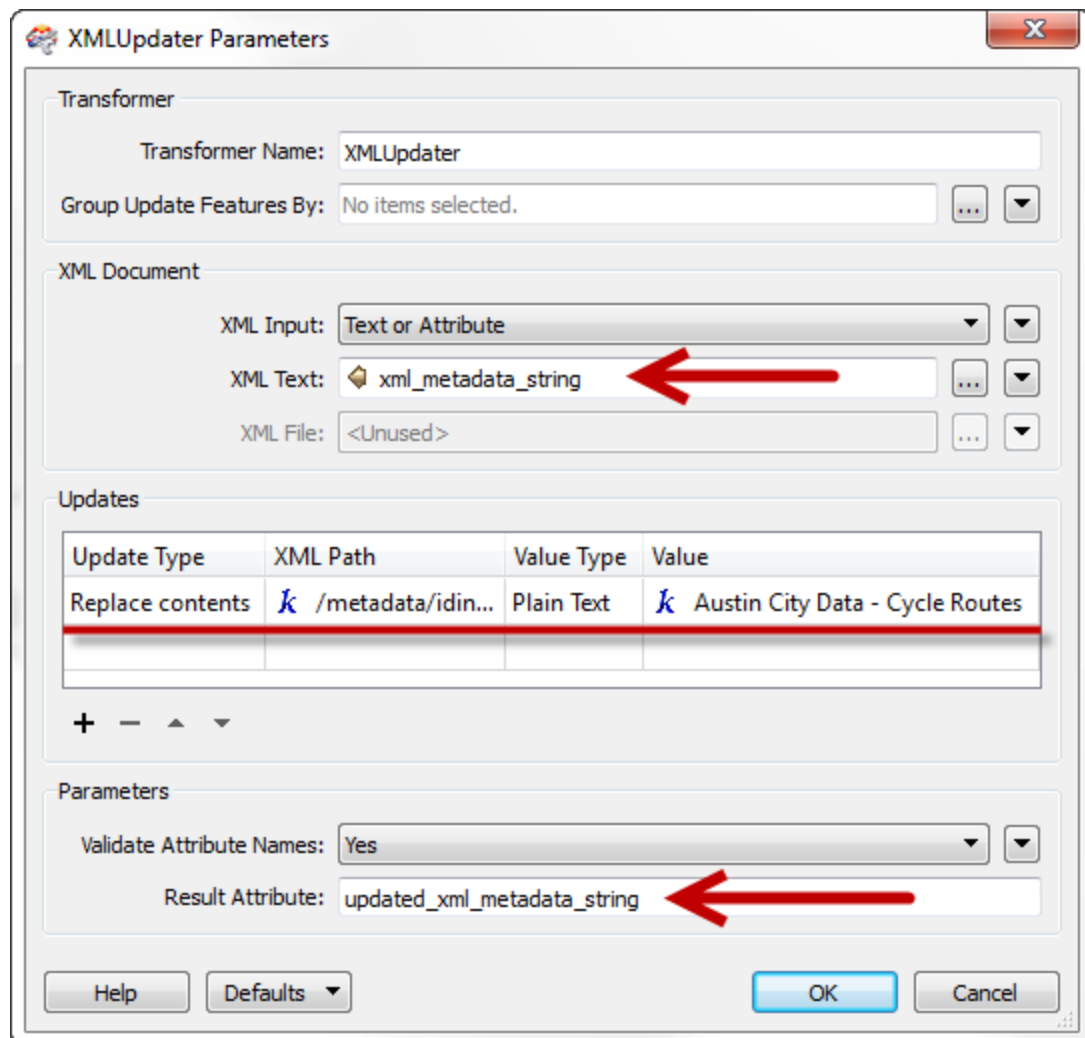
### 6) Set XMLUpdater Parameters

Open the XMLUpdater parameters dialog. Set:

XML Input                      Text or Attribute  
 XML Text                      Set to Attribute Value > xml\_metadata\_string  
 Result Attribute              updated\_xml\_metadata\_string

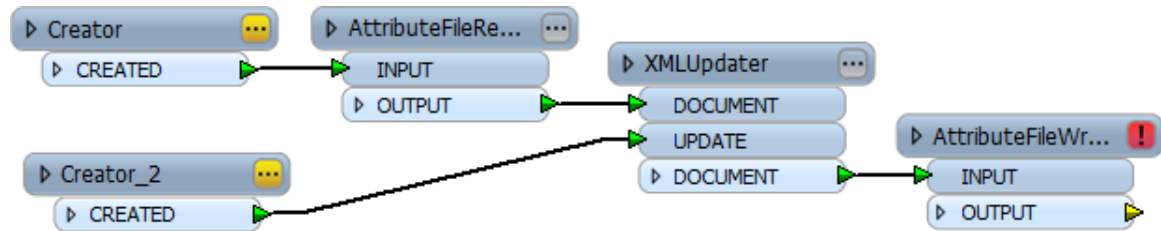
Now for the update part. Set:

Update Type                  Replace Contents  
 XML Path                    /metadata/idinfo/keywords/theme/themekey  
 Value Type                  Plain Text  
 Value                         Austin City Data – Cycle Routes



### 7) Add AttributeFileWriter

Add an AttributeFileWriter transformer.



Open the parameters dialog and set it to write the contents of updated\_xml\_metadata\_string to a new XML document, again with utf-8 encoding.

### 8) Save and Run Workspace

Save the workspace and then run it.

Inspect the output XML dataset to ensure it is correct.

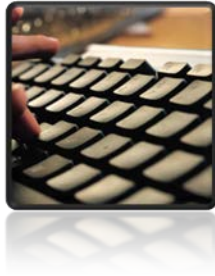
```
- <keywords>
  - <theme>
    <themekt>REQUIRED: Reference to a formally registered thesaurus or a
      similar authoritative source of theme keywords.</themekt>
    <themekey>Austin City Data - Cycle Routes</themekey>
  </theme>
</keywords>
```



### Advanced Task

If there is time, practice using the XMLUpdater by updating other metadata fields such as ModDate, and the contact fields (those beginning with “cnt”). Be sure to update fields with varying XML paths, to get familiar with the required syntax.

## Session Review



*This session was all about XML data and FME.*

### What You Should Have Learned from this Session

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

#### **Theory**

- XML is a **markup language** whose contents and structure is defined by a **schema**
- **GML** is an XML grammar that defines geographical entities. Predefined formats are usually defined by an **Application Schema**
- FME provides many pre-defined **XML readers and writers** so that the user does not need to manually work with the raw XML/GML data
- FME can extract information from an XML document using a **Feature Path, GeometryReplacer transformer, XQuery**, or an **xfMap**
- XML documents can be read in their entirety or as a **fragment**, often for uses such as updating XML-format **metadata**

#### **FME Skills**

- The ability to read an XML format that has a pre-defined reader or writer
- The ability to extract data from an XML document using a Feature Path
- The ability to extract GML geometry from inside an XML document
- The ability to extract data from an XML document using XQuery
- The ability to read XML files and fragments into an attribute, and then update the contents