

# Article

« [FME Desktop \(/S/Topic/0TO4Q000000QL9uWAG/Fme-...\)](#)

## XML Writing with XMLTemplater

Aug 3, 2022 • Knowledge

### Product Type

FME Desktop

---

### FME Version

2022.0

---

**Tutorial:** [Tutorial: Getting Started with XML \(/s/article/tutorial-getting-started-with-xml\)](#) | **Previous:** [How to Read XSD-Driven XML \(/s/article/How-to-read-XSD-Driven-XML\)](#) | **Next:** [How to Consume and Produce XML using Application Schemas with the XSD-Driven XML Writer \(/s/article/XSD-Driven-XML\)](#)

## Introduction

XML is a form of structured text that is commonly used in open standards and exchange formats. It is also frequently encountered in web and messaging applications. Being able to convert data to XML is an important capability of FME. This article covers FME's approach to writing XML and includes a basic example to help you get started. It assumes some understanding of basic FME and XML concepts. For example, the term 'element' in XML can relate to either a feature or an attribute in FME terms, depending on how the XML document is interpreted. See [FME Approach to XML \(/s/article/tutorial-fme-approach-to-xml\)](#) for more background.

## Writing XML

FME uses a template approach to writing XML. The XML document structure is placed in or referenced by an XMLTemplater transformer. Then fme:get-attribute functions are placed within the template at the locations where you want feature attribute values to be merged into the document. This functions much like a mail merge operation, where a letter template receives name and address values from each record to dynamically generate customized messages.

XML documents can be generated one per feature such as for dataset metadata. Or root and sub-templates can be used to model a document that has one root with multiple child elements based on a multi-record dataset. The XML document structure can be generated from an XSD using the XMLSampleGenerator or obtained from a sample XML output document. Then element values are replaced with FME field values using fme:get-attribute functions where we want the content to be driven by the source data. The XML that is produced by XMLTemplater is stored in a single \_result attribute that can then be validated and formatted with the XMLValidator and XMLFormatter transformers. In this exercise, we use sample XML to seed the template and will leave validation and formatting to later exercises.

Typically the final output is written to disk using the Text File writer since the XML result attribute is just a large text field. The XML writer is not often used since it does not yet support reading schema from XSD. There are special cases where the XML writer is beneficial, such as using fragment mode to write very large XML datasets. This advanced approach allows XML fragments to be written as they are received rather than building a buffer of the entire document before

writing as is the case with the Text File writer.

## Content Overview

- [Part 1: Writing One XML Record](#)
- [Part 2: Writing Multiple XML Records](#)

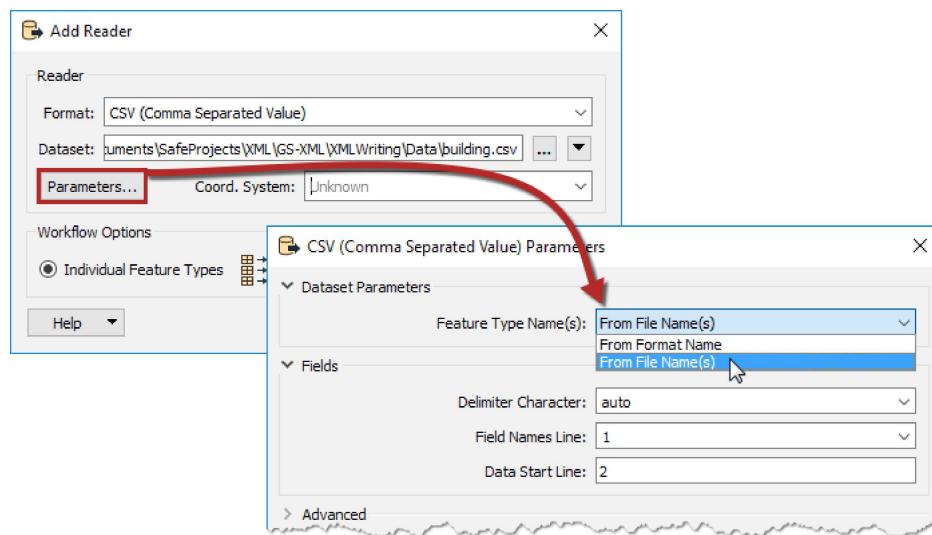
## Step-by-step Instructions

### Part 1: Writing One XML Record

The following exercise shows how to take a single record fed into an XMLTemplater template, and generate a basic XML document that matches the structure found in `safe_building_demo.xml`.

#### 1. Open FME Workbench

Open FME Workbench and start with a blank workspace. Drag and drop the attached source `building.csv` file, which is available from the Files section of this article. Click on the Parameters button and then change the Feature Type Name(s) to From File Name(s), this will make it easier to distinguish between the two CSV files. There are no other parameters to set since we are planning on writing to a text-based file.

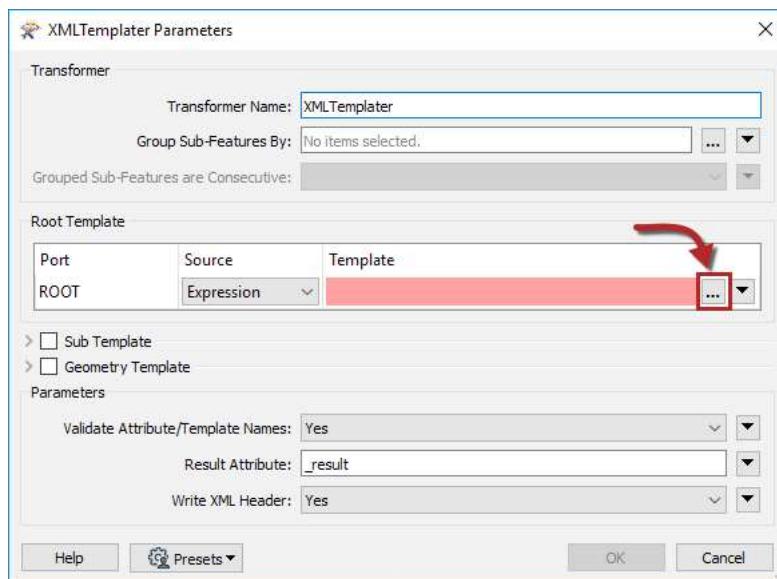


#### 2. Add an XMLTemplater

The first step for converting this to XML is to generate the root element. To create the XML template, we need to copy the contents from the root object of an example of the type of XML file that we would like to generate - in this case, the root from `safe_building_demo.xml`.

Add an XMLTemplater transformer to the canvas and connect the CSV source building feature type to it. Open the XMLTemplater parameters and click the ellipsis for Template to open the ROOT Template Expression dialog. Paste in the following:

```
<?xml version="1.0"?>
<Dataset xmlns="http://www.safe.com"> (http://www.safe.com);
    <Building id="Building">
        <Address>street address</Address>
        <City>city</City>
        <Province>BC</Province>
        <Country>Canada</Country>
        <Location>
            <Longitude>0</Longitude>
            <Latitude>0</Latitude>
        </Location>
    </Building>
</Dataset>
```



### 3. Add a Logger

Add a Logger transformer after the XMLTemplater transformer and then run the workspace. Scroll up to view the XML in the Translation Log. Notice that the template has written out the output that is within the `_result` field, which is the same as the template we just added.

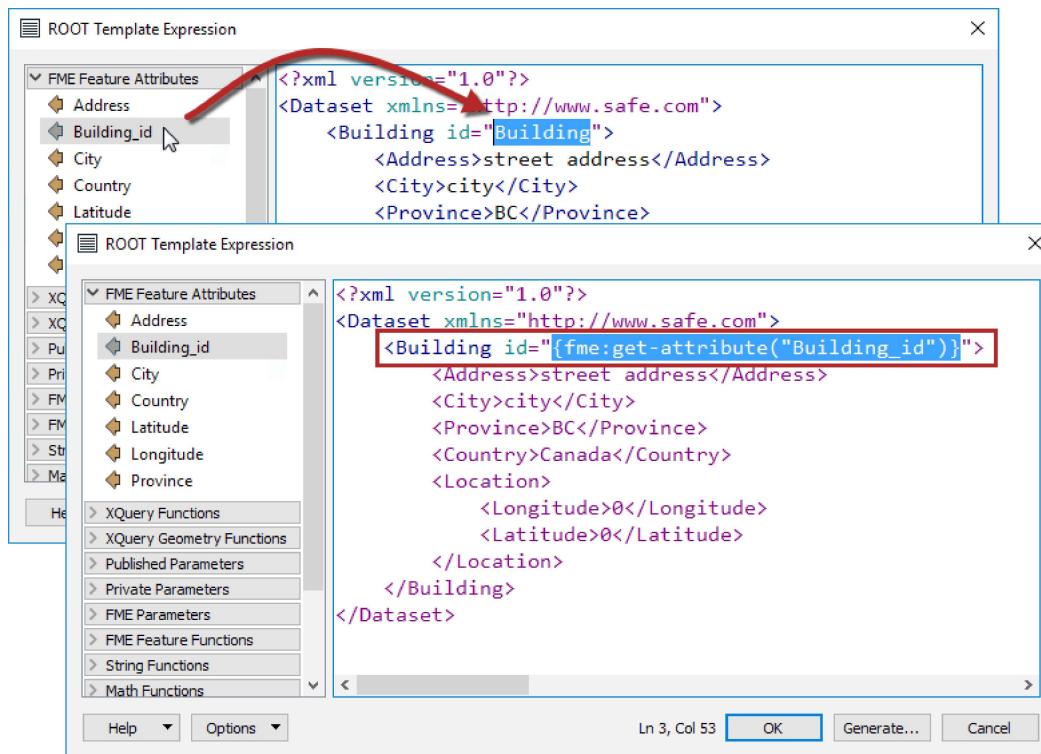
What we really want is to merge the values from the input CSV record into the template before we write the XML.

```
97 2020-01-20 11:13:53| 0.7| 0.0|INFORM|Attribute(encoded: UTF-8)
: '_result' has value '<?xml version="1.0" encoding="UTF-8"?>
98 <Dataset xmlns="http://www.safe.com">
99     <Building id="Building">
100         <Address>street address</Address>
101         <City>city</City>
102         <Province>BC</Province>
103         <Country>Canada</Country>
104         <Location>
105             <Longitude>0</Longitude>
106             <Latitude>0</Latitude>
107         </Location>
108     </Building>
109 </Dataset>'
```

### 4. Modify the Template

To merge the values from the CSV into the template, we need to modify the template. Open up the XMLTemplater, click on the ellipsis to edit ROOT, then click on the contents of each element we want to merge values into, and then click on the attribute we want to merge values from.

This should insert the attribute function associated with the attribute we selected (`{fm: get-attribute("Building_id")}` etc) into the location of the template where we positioned the cursor. Remember to keep the quotes around the building attribute.



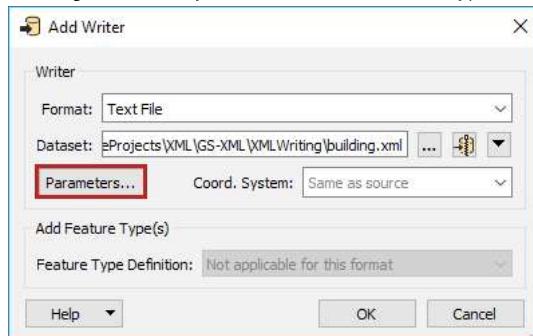
Once you have done this for all the available attributes you should have a ROOT template that looks like this:

```
<?xml version="1.0"?>
<Dataset xmlns="http://www.safe.com" (http://www.safe.com");>
    <Building id="{fme:get-attribute("Building_id")}">
        <Address>{fme:get-attribute("Address")}</Address>
        <City>{fme:get-attribute("City")}</City>
        <Province>{fme:get-attribute("Province")}</Province>
        <Country>{fme:get-attribute("Country")}</Country>
        <Location>
            <Longitude>{fme:get-attribute("Longitude")}</Longitude>
            <Latitude>{fme:get-attribute("Latitude")}</Latitude>
        </Location>
    </Building>
</Dataset>
```

You can also copy and paste the above text to save time.

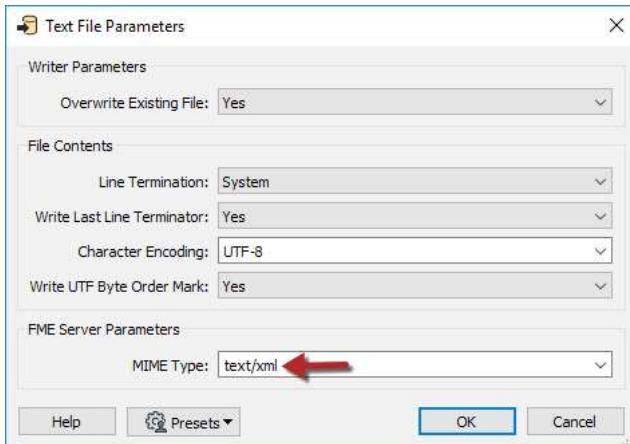
## 5. Add a Writer

We can now write our data out to a text file. Add a new writer to the canvas and set the Format to Text File. Browse to a location to save the file and name it building.xml. You may have to switch the Save as Type to All Files (\*).



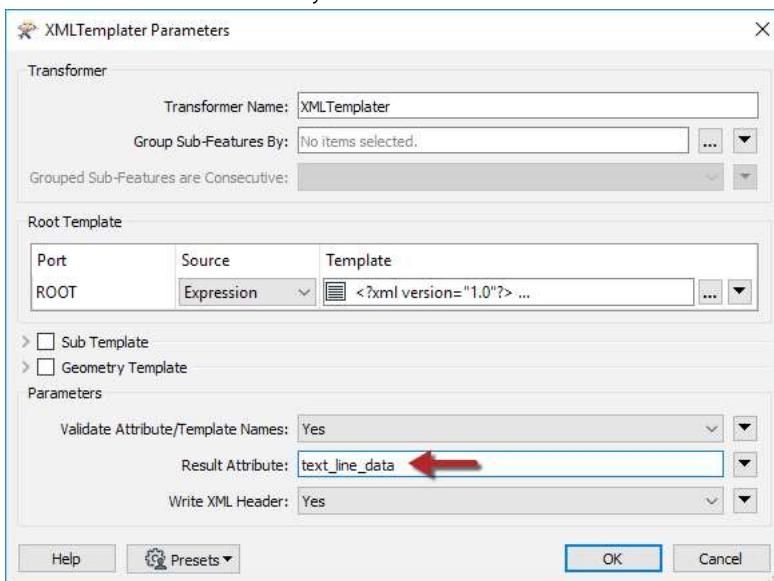
Open the parameters, and then set the output MIME type to text/xml. This can be important for displaying the XML correctly in some viewers such as web

browsers.



Click OK to complete creating the writer and then connect it to the XMLTemplater transformer. You can delete the Logger if you wish.

One final step before we can run the workspace is to change the Result Attribute in the XMLTemplater from \_result to text\_line\_data. Changing this attribute allows the XML to be understood by the Text File writer.



## 6. Run the Workspace

Run the workspace and examine the resulting building\_output.xml output. See how these values are merged into the XML document structure. You should see the following output. Note how the fme:get-attribute("") functions have been replaced by the feature attribute values:

```
<?xml version="1.0" encoding="UTF-8"?>
<Dataset xmlns="http://www.safe.com"> (http://www.safe.com");
    <Building id="Surrey Head Office">
        <Address>7445 132 St.</Address>
        <City>Surrey</City>
        <Province>BC</Province>
        <Country>Canada</Country>
        <Location>
            <Longitude>-122.860</Longitude>
            <Latitude>49.138</Latitude>
        </Location>
    </Building>
</Dataset>
```

You can view the output data in Visual Preview, the written text file, or the Log File if you still have the Logger attached.

The screenshot shows the FME Workbench Visual Preview tool. The title bar says "Visual Preview". Below it is a toolbar with icons for various preview options. The main area is titled "Table" and has a dropdown menu set to "text\_line". There is a "Columns..." button. The data grid displays the following XML structure:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <Dataset xmlns="http://www.safe.com">
3     <Building id="Surrey Head Office">
4         <Address>7445 132 St.</Address>
5         <City>Surrey</City>
6         <Province>BC</Province>
7         <Country>Canada</Country>
8         <Location>
9             <Longitude>-122.860</Longitude>
10            <Latitude>49.138</Latitude>
11        </Location>
12    </Building>
13 </Dataset>

```

Below the grid, there is a search bar, a dropdown for filtering by column, and a status message: "1 selected / 13 row(s)".

## 7. Save the Workspace

Save the workspace as we will be using the same workspace in Part 2.

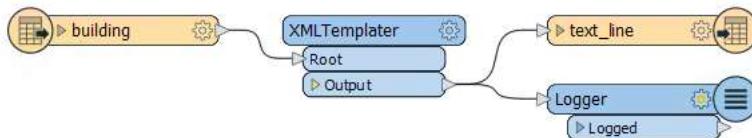
## Part 2: Writing Multiple XML Records

The following exercise shows how to take the workspace from Part 1 and add multiple child elements based on input records using an XMLTemplater subtemplate.

Going back to our original task, we want to use FME to generate XML content that matches the structure found in `safe_building_demo.xml`. We generated the root content in example [Part 1](#), so the next step is to generate the Room elements.

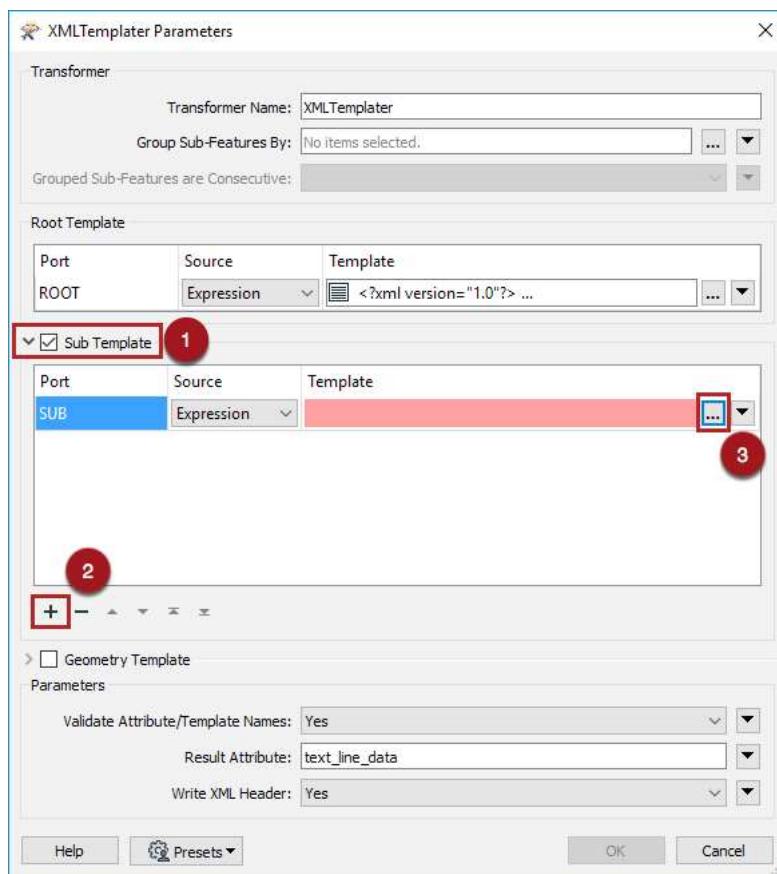
### 1. Open the Previous Workspace

Open up the workspace that you created from Part 1, or download the `WritingXML-Part1-Complete.fmw` completed workspace.



### 2. Edit the XMLTemplater

Open up the XMLTemplater and enable Sub Template. Click on the plus sign (+) to add a new sub template.



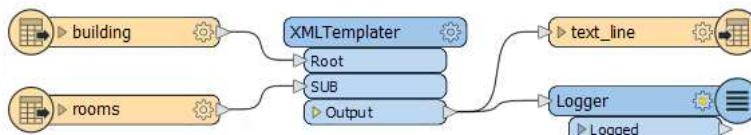
As before, we start the template with a single example instance or element of the object we want to generate. In this case, we need an example of the Room element, which we can get from the first room in safe\_building\_demo.xml:

```
<Room id="Admin_100">
    <Name>Reception</Name>
    <Category>Admin</Category>
    <Area units="m2">12</Area>
</Room>
```

Click on the ellipsis for the sub template and then copy and paste the above XML. Then click OK twice to save the change to the XMLTemplater. You will notice a new input port on the XMLTemplater called 'SUB'. We need to connect the Room input features to this in order to have the fields we need available to the XMLTemplater on the feature schema.

### 3. Add Another Reader

We will need to read in the Room features before we can connect it to our XMLTemplater. Add another CSV reader and browse to the downloaded Rooms.csv. Click on the Parameters button and then change the Feature Type Name(s) to From File Name(s). Click OK twice to add the reader. Once the reader has been added, connect it to the SUB input port on the XMLTemplater.



### 4. Modify the XMLTemplater Sub Template

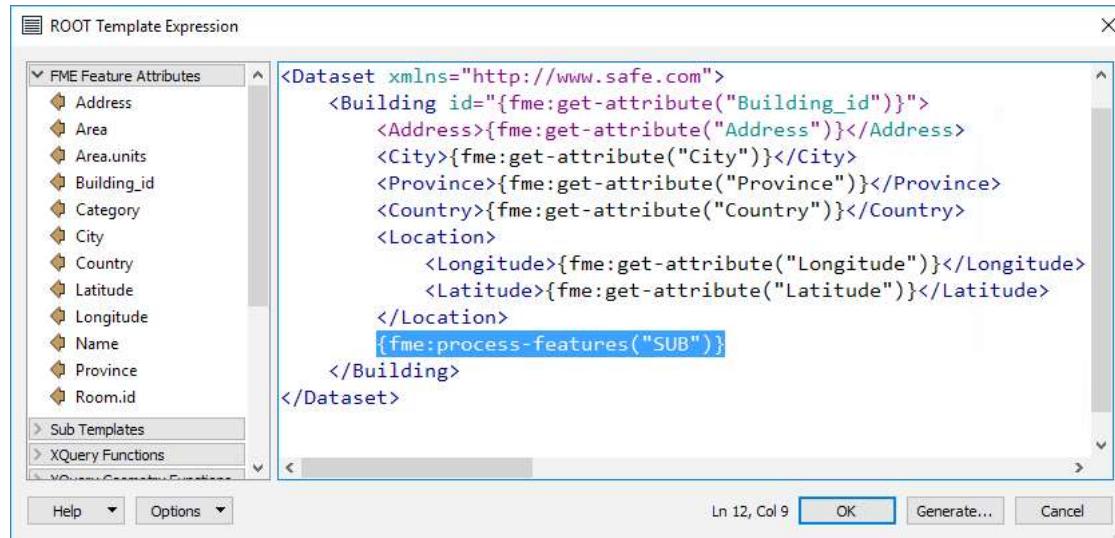
Now that the room feature type is connected, we can replace the values with the values we read from the input CSV file. Open the XMLTemplater and modify the sub template, by clicking on each of the attributes. This should yield a sub template that looks like this:

```
<Room id="{fme:get-attribute("Room.id (http://Room.id)")}">
  <Name>{fme:get-attribute("Name")}</Name>
  <Category>{fme:get-attribute("Category")}</Category>
  <Area units="{fme:get-attribute("Area.units")}">
    {fme:get-attribute("Area")}</Area>
</Room>
```

## 5. Modify the ROOT Template

To complete the XMLTemplater configuration, we need to tell the root template to call the sub template. This is done using the fme:process-features("SUB") function. Add this to your root template so that the Room child elements are inserted at the end of the Building parent object.

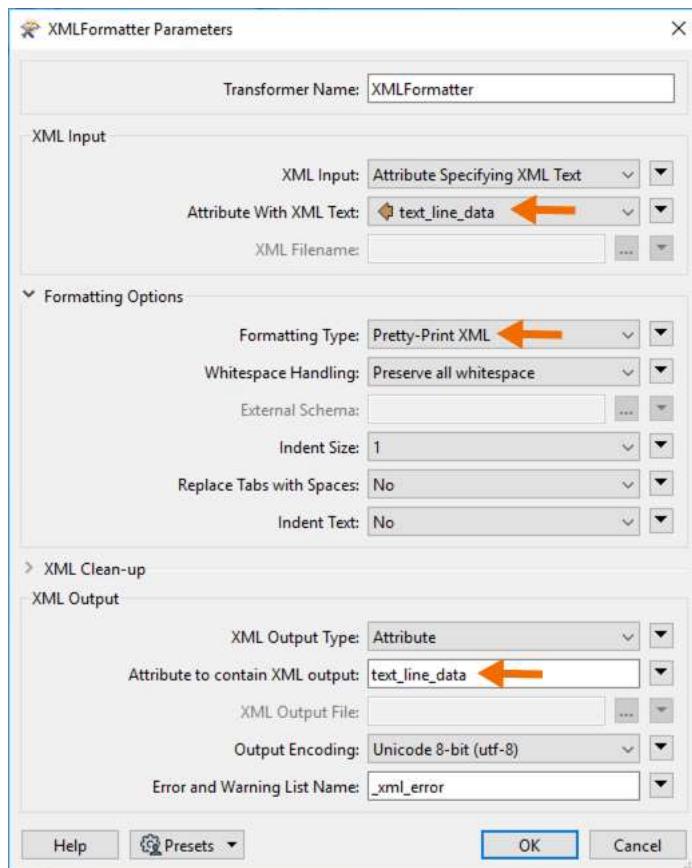
```
<?xml version="1.0"?>
<Dataset xmlns="http://www.safe.com" (http://www.safe.com">;
  <Building id="{fme:get-attribute("Building_id")}">
    <Address>{fme:get-attribute("Address")}</Address>
    <City>{fme:get-attribute("City")}</City>
    <Province>{fme:get-attribute("Province")}</Province>
    <Country>{fme:get-attribute("Country")}</Country>
    <Location>
      <Longitude>{fme:get-attribute("Longitude")}</Longitude>
      <Latitude>{fme:get-attribute("Latitude")}</Latitude>
    </Location>
    {fme:process-features("SUB")}
  </Building>
</Dataset>
```



## 6. Add an XMLFormatter

Finally, it's a good idea to format our XML before writing to disk. This makes it a lot easier to review our output and troubleshoot any problems, and has the added bonus of catching syntax errors since only valid XML can be formatted.

Add an XMLFormatter transformer after the XMLTemplater. Set the Attribute with XML text to text\_line\_data. Set the Attribute to contain XML Output to text\_line\_data as well. Leave the rest of the settings with their default values (Formatting type listed under Formatting Options should be Pretty-Print XML).



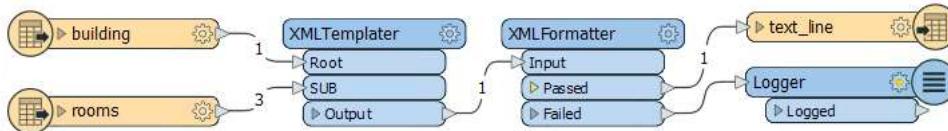
## 7. Connect the XMLFormatter

Connect the XMLFormatter transformer to the Text File writer. If you still have your Logger transformer, move it to the Failed output port on the XMLFormatter.

Once everything is connected, run the workspace. Your output should appear as below. Note the multiple Room elements that are inserted inside the Building object after the Location element:

```
<?xml version="1.0" encoding="UTF-8"?>
<Dataset xmlns="http://www.safe.com"> (http://www.safe.com);
    <Building id="Surrey Head Office">
        <Address>7445 132 St.</Address>
        <City>Surrey</City>
        <Province>BC</Province>
        <Country>Canada</Country>
        <Location>
            <Longitude>-122.860</Longitude>
            <Latitude>49.138</Latitude>
        </Location>
        <Room id="Admin_100">
            <Name>Reception</Name>
            <Category>Admin</Category>
            <Area units="m2">12</Area>
        </Room>
        <Room id="Sales_101">
            <Name>Sales Office</Name>
            <Category>Sales</Category>
            <Area units="m2">20</Area>
        </Room>
        <Room id="Meet_102">
            <Name>Meeting Room</Name>
            <Category>Meetings</Category>
            <Area units="m2">25</Area>
        </Room>
    </Building>
</Dataset>
```

The completed workspace is shown below after a run. Note the XMLTemplater has one input feature for the root Building element, three input features for the Room elements, and one output feature for the combined XML document's output.



Continue to the next article: [How to Consume and Produce XML using Application Schemas with the XSD-Driven XML Writer](#) ([/s/article/XSD-Driven-XML](#)).

## Data Attribution

Data created in-house by [Safe Software Inc.](#) (<http://www.safe.com>).

---

### First Published Date

7/29/2020, 12:16 AM

---

### Last Published Date

8/3/2022, 9:45 PM

[Transformation  
\(/s/topic/0TO4Q000000...\)](#)[FME Desktop  
\(/s/topic/0TO4Q000000...\)](#)

Sort by:

Latest Posts ▾



LizAtSafe ([/s/profile/0050c00000CeGV0AAN](#)) (Employee) published a new version of this Knowledge.  
August 3, 2022 at 9:45 PM ([/s/feed/0D54Q00009ggQspSAE](#)).

 [Like](#)
 [Comment](#)
[Log In to Comment](#)


trentatsafe ([/s/profile/005a00000CdnWnAAJ](#)) (Employee) published a new version of this Knowledge.  
July 29, 2022 at 2:00 PM ([/s/feed/0D54Q00009fwXflSAE](#)).

1 view

 [Like](#)
 [Comment](#)
[Log In to Comment](#)
[Follow](#)

#### [Files \(1\) \(/s/relatedlist/ka14Q000001DWy3QAG/AttachedContentDocuments\)](#)

2a. XML Writing with the XMLTemplater

[ZIP](#) Aug 3, 2022 • 71KB • zip

[View All](#)

[\(/s/relatedlist/ka14Q000001DWy3QAG/AttachedContentDocuments\)](#)

## Related Articles

[XML Writing to Custom Application Schemas - XMLTemplater - Basic Example \(/s/article/xml-writing-to-custom-application-schemas-xmltempl\)](#)

[Working with Geodatabase Metadata: Writing/Updating Metadata \(/s/article/working-with-geodatabase-metadata-writing-to-xml\)](#)

[XML FAQ: Reading and Writing XML \(/s/article/xml-faq-reading-and-writing-xml\)](#)

[Harvesting and writing ISO19115 XML Metadata \(/s/article/harvesting-and-writing-iso19115-xml-metadata\)](#)

[Writing JSON with the JSONTemplater \(/s/article/json-writing-with-jsontemplater\)](#)

[Getting Started](#)

[Ideas](#)

[Feedback](#)

[\(../s/topic/0TO4Q000000QKioWAG/welcome/bridea/acideasULT\\_bridea\\_c/00B4000000AVHfVgM/LkoFWpziDyaWQkL78\)](#)

[Forums \(./s/forums/\)](#)[Groups](#)[Knowledge Base \(./s/knowledge-base/\). \(./s/group/CollaborationGroup/00Ba000000A0BxJEAV\)](#)[Support \(./s/support/\)](#)[\(https://safe.com\)](https://safe.com)[Register / Log In \(/s/login/\)](#)© Safe Software Inc | [Legal \(https://www.safe.com/legal/\)](#)[\(https://www.safe.com/legal/\)](https://www.safe.com/legal/)

#### Land Acknowledgement —

Safe Software respectfully acknowledges that we live, learn and work on the traditional and unceded territories of the Kwantlen, Katzie, and Semiahmoo First Nations.