Style Name Maps and Style Definition Ids in GTViewer 14

Overview

GTViewer has supported a feature call **Style Name Maps** since version 9 (circa August 2010). This feature is a fairly significant addition to GTViewer, but it has not been promoted much since it was introduced. With the **Style Name Maps**, the GTViewer element format was upgraded to include a 32 bit integer value to store a **Style Definition Id**. If a **Style Definition Id** is not provided for an element, the element format will appear unchanged and is still compatible with versions of GTViewer prior to version 9. However, with this new feature you can define a **Style Definition Id** on an element which is mapped to a **Style Definition** for a given Zoom Level range via the **Style Name Map**. Thus, the element's **Style Definition** no longer must be bound to its **Filter Id.** This subtle change turns out to be a major shift in the way GTViewer data can be created.

The **Style Name Map** feature has been largely ignored over the last few years for several reasons. First, changing the element format is a fairly major change to **GTViewer**. Since the element format support has been available for a while now, adding **Style Definition Ids** to your data will not cause problems with older versions of GTViewer (but the new functionality won't be supported either). Second, support for the **Style Name Map** was not available in a .GTX extract file. Third, it has taken a while for the **GTData** tools and the FME Writer Plugin to catch up, and there has not been a good way to create data using the **Style Definition Id** without manually generating your data with the **GTData Objects**.

With Version 14 of GTViewer, all of these problems have been addressed. The **Style Name Maps** are now fully supported in both the GTM and GTX files. **Style Name Maps** have also been enhanced to map **Style Names** directly to **Style Definitions** or to an **Abstract Name** which is then mapped to a set of zoom level threshold ranges mapped to a style definition. **GTData** tools such as **GTIndex**, **GTPack**, and **GTExtract** have been updated to support **Style Name Maps**, and the **FME Writer Plugin** has also been enhanced to not only support the **Style Definition Id**, but to also provide a new mode for generating **Filter Files** that are decoupled from the Style Definition File (something not possible with previous versions of the plugin).

The Style Name Map

The **Style Name Map** is a text file that defines how a **Style Definition Id** (defined on the GTViewer element) is mapped to a **Style Definition** defined in the **Style Definition File** (usually called style.def). The **Style Definition** contains a set of properties (color, weight, linestyle, size, justification, etc.) that define how an element appears when it is rendered in GTViewer and has traditionally been used with the **Style Map File** (usually called Style.map). The **Style Map File** maps a **Category** and **Filter Id t**o a **Style Definition Name**; whereas, the **Style Name Map File** maps the **Style Definition Id** on the individual element to a **Style Definition Name**.

The **Style Name Map File** contains two sections. The first section, called **Id To Name**, maps a **Style Definition Id** number to an **Abstract Style Name**. This section's entries are in the following format:

```
<Style Definition Id> = <Abstract Style Name>
```

Both the **Style Definition Id** and **Abstract Style Name** must be unique for all entries.

The second section, called **Name to Style**, is optional. If it is not defined in the **Name to Style** section, an **Abstract Style Name** will be mapped to a **Style Definition** of the same name. However, you can map an **Abstract Style Name** to one or more entries of the following format:

```
<Abstract Style Name> = <Style Definition Name>
```

or

```
<Abstract Style Name> = <Style Definition Name>|<min threshold>|<max threshold>
```

Here, the **Abstract Style Name** is mapped to a **Style Definition Name.** If the min and max threshold values are not provided, they will default to 0 (not used).

The following example shows a Style Name Map:

StyleNameMap.map

```
[Id To Name]
1=STREETS - A21
2=STREETS - A51
3=STREETS - A63
4=STREETS - A31
5=STREETS - A15
6=STREETS - A35
7=STREETS - A25

[Name To Style]
STREETS - A15=Main Road - CLose|0|100
STREETS - A15=Main Road - Far|100|0

STREETS - A21=Main Road - CLose|0|100
STREETS - A21=Main Road - Far|100|0
```

Here the **Id to Name** section maps Style Definition Ids 1 to 7 to Abstract Style Names. In this example, the ids are mapped to different classifications of road types (A21, A51, etc.). Here only, Streets – A15 and Streets – A21 are defined in the **Name to Style** section because they are the only ones that have different style definitions used for different Zoom Level Ranges. The rest of the Abstract Style Names just default to a Style Definition of the same name. The **Style Definition File** (style.def) for this **Style Name Map** would look like this:

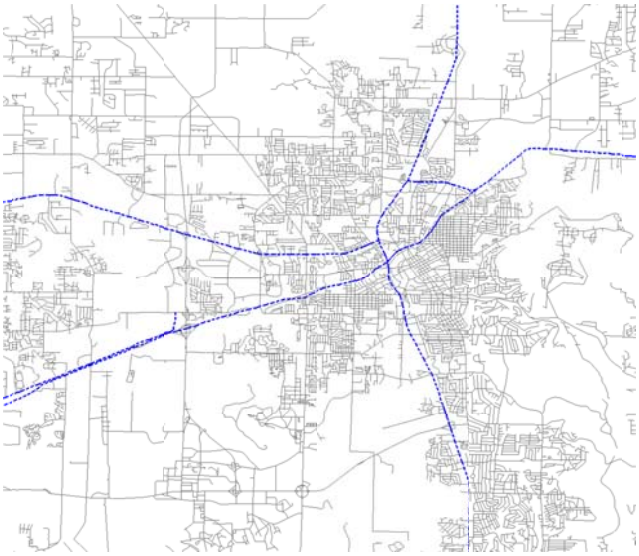
Style.Def

```
[Style Definitions]

[STREETS - A21]
ColorValue=100|100|100

[STREETS - A51]
ColorValue=100|100|100

[STREETS - A63]
```

```
ColorValue=100|100|100

[STREETS - A31]
ColorValue=100|100|100

[STREETS - A15]
ColorValue=100|100|100

[STREETS - A35]
ColorValue=100|100|100

[STREETS - A25]
ColorValue=100|100|100

[Main Road - Close]
ColorValue=255|0|0
Weight=5

[Main Road - Far]
ColorValue=0|0|255
Weight=3
Style=2
```

All of the lesser roads are just styled as gray lines while the Major Roads (A21 and A15) are styled as dashed blue lines when zoomed out far and solid red lines when zoomed in close. The following screenshots show the difference.

Far Out:



Close In:

The Category definition for the Roads in the GTM file looks like the following:

```
[Category 2]
Name=Category 2
IndexFile=test_2.gtn
graphicsFile=test_2.gtg
Filter=test_2.flt
StyleNameMap=StyleNameMap.map
DisplayStatus=0
MinDisplayThreshold=0
MaxDisplayThreshold=0
SelectFlag=0
DataId=1
```

The General Info section would still contain the entries to define the Style Definition File and the Style Map File (no different than before):

```
StyleDefinitionFile=Style.def
StyleMapFile=Style.map
```

The **Style Map File** just contains a header entry to make it a valid file (no mappings are provided):

Style.Map

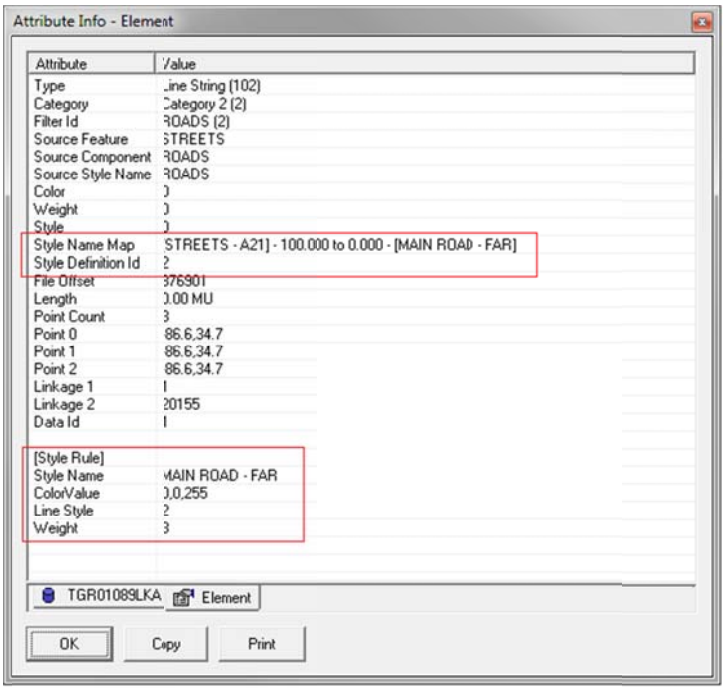
```
[Style Map]

# FilterMap|min|max|category|filterIdLow|filterIdHigh|Style|
```

One question that you may have is: Why are the Style.def and Style.map files defined in the General Info section while the Style Name Map is defined in the Category section? This is because each Category can have its own **Style Name Map**, or they can share a **Style Name Map** with other categories. If more than one Category section specifies the same **Style Name Map File**, they will share it. All of the **Style Name Maps** will use the Style Definitions in the **Style Definition File**.

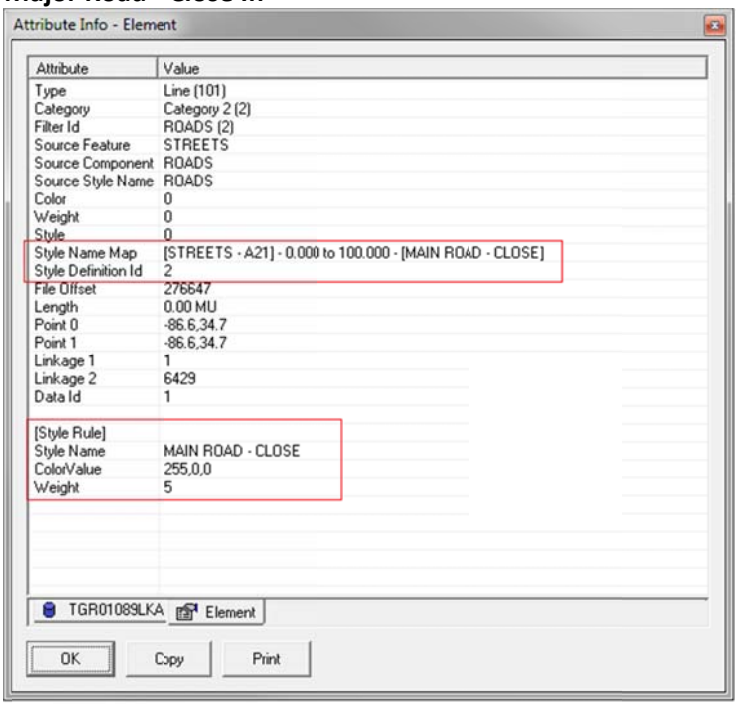
Attribute Info Dialog

The **Style Name Map** and **Style Definition Id** are both displayed in the **Attribute Info Dialog** on the Element tab:

Major Road - Far Out



Major Road - Close In



Style Map versus Style Name Map

The **Style Map** and **Style Definitions** have been supported by **GTViewer** since version 1.1. They are used by all current conversion tools including those for G/Technology, Smallworld, Oracle Spatial, and Millsoft. The FME Writer Plugin also uses them extensively.

The **Style Map** is a very powerful tool for assigning symbology to elements. By mapping the **Filter Id** to a **Style Definition**, it allows you to defer specifying the actual **Style Definition** properties until after the data is converted, and you can change styles without reconverting the data. Before the **Style Map** was supported, each element had to have its style explicitly defined at conversion, and the style could not be changed without changing the element (or reconverting the data). While this method sounds primitive, it does work quiet well for Microstation and AutoCad data or any data whose styles are **Instance Based** and cannot be easily generalized. The **Style Map** also has to ability to map **Filter Ids** to a **Style Definition** based on a **Zoom Level Range**, so you can specify different symbology for the same element at different zoom levels (also something that Instance Based symbology can't do).

The **Filter Id,** which is required on all **GTViewer** elements, is used to control the display of elements in the view. The **Filter Files** map the **Filter Id** to a user-friendly name and also groups them together into logical sets of features. When the user uses the **Display Filter Dialog** in **GTViewer**, it basically tells which **Filter Ids** can be display and which ones will be hidden, then when the view is rendered only elements with the displayed **Filter Ids** will be drawn. One drawback to using the **Filter Id** for both display control and style mapping is that the display filter resolution must match the same resolution required to allow all of the required styles. For example, let say you have a Primary Conductor, and you want to have two display groups (Overhead and Underground). However, your symbology needs to show both orientation and number of phases:

- Above Ground – 1 Phase
- Above Ground – 2 Phase
- Above Ground – 3 Phase
- Underground – 1 Phase
- Underground – 2 Phase
- Underground – 3 Phase

We will need 6 Filter Ids to support all 6 styles. You can do some creative grouping to make simplify the Display Filters to 2 groups (Above Ground and Underground), and you may actually want to be able to control the display at this resolution. But this problem can grow out of hand pretty quickly. What if we wanted to add state to the symbology? If we just used Proposed, Inservice, and Retired, you would need 18 Filter Ids to support all of the different styles.

As mentioned early, the **Style Name Map** unbinds the **Filter Id** from the **Style Definition**. So, the **Style Definition Id** has no dependence on the **Filter Id**, and your **Display Filters** can be whatever you want, while your **Style Definition Id** can specify the styles you need. The **Style Definiton Id** is an Instance Based property, but it still has the advantage of deferring the specification of the Style Properties until after the data conversion and allowing changes to the Styles without reconverting the data. The cost here is that the GTViewer elements are slightly larger (4 bytes).

You can also use **Style Maps** and **Style Name Maps** at the same time. The **Style Map** will always take precedence, and it may become unnecessarily complex if you use both methods for defining styles at the same time. Not a recommended approach, but fully supported.

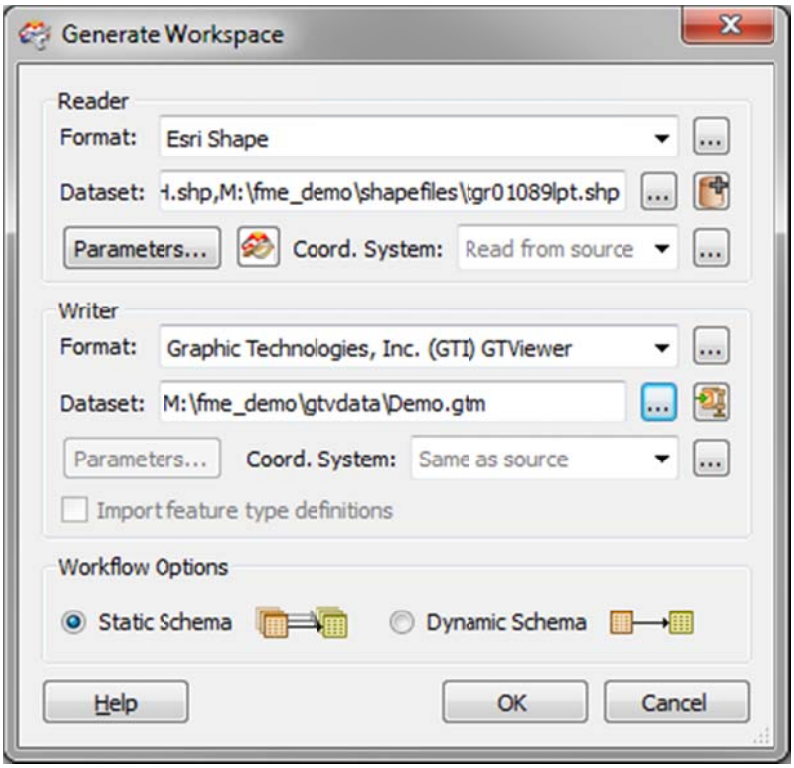
Creating Data with Style Name Maps

Currently, the easiest way to create **GTViewer** data that uses a **Style Name Map** and **Style Definition Ids** is to use the **GTViewer FME Writer Plugin**. For those of you who have used the GTViewer FME Writer Plugin before, there are a few things that are different. There are some new **Filter File** creation modes and several new Format attributes. If you have existing FME Workbenches, it will be difficult (but not impossible) to migrate them to the new modes; however, it is recommended to just create a new workbench and use the new modes.

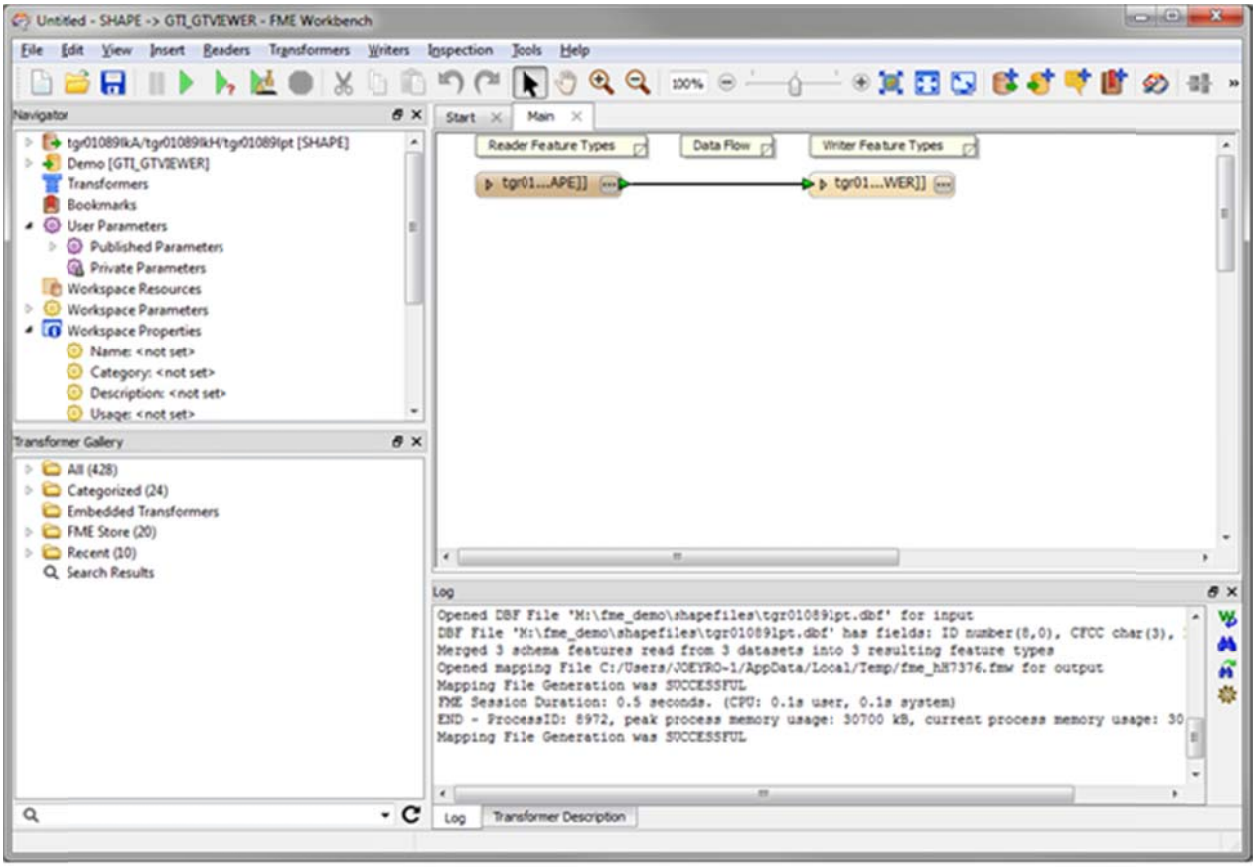
I will illustrate using the new modes with a simple example. I will convert the TIGER data seen in the examples above and go through each step of the process.

1) Create a new Workspace. The Reader will be Esri Shape and I will specify a TIGER shapefile (tgr01089lkA.shp which contains roads).

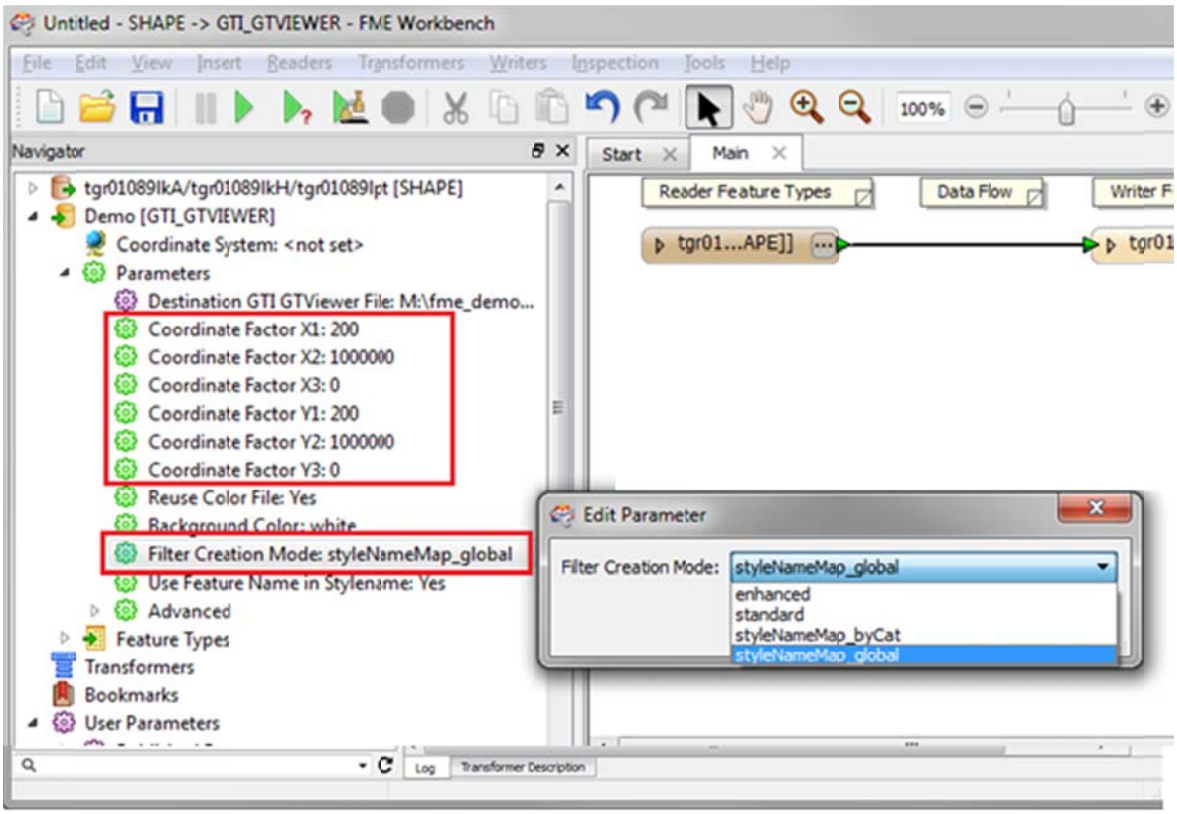
   The Writer will be GTI_GTViewer and I will give it Demo.gtm as the dataset to create.



2) The layout will look like this to start with:



3) In the Navigator, set the Coordinate Factors and Filter File Creation mode like the following:



The TIGER data is in an LL-83 projection (Latitude/Longitude), so the **Coordinate Factors** are for this are the standard values for latitude/longitude data (200/1000000/0). The **Filter Creation Mode** is set to **styleNameMap_global** which will create one **Style Name Map** for the entire conversion. You could select **styleNameMap_byCat** to create a separate **Style Name Map** for each category. The default mode for FilterCreateMode is **Enhanced** which creates the normal **Style Map** and **Style Definition** files.

4) There are 4 new Format Attributes available:
   a. **gti_groupFilterName** – specifies the Filter File Group for the feature created. If this entry is not specified, it will default to "Features".
   b. **gti_FilterName** – specifies the name of the Filter Name for the feature created. All features using this Filter Name will have the same Filter Id. If this entry is not specified, it will default to the gti_featureName value (which will default to the Reader's Feature Type name if not defined).
   c. **gti_style_def_id** – specifies the Style Definition Id to put on the element. This entry should only be used if you are manually setting the Style Definition Id, which is generally not the case. It is similar to using the gti_filterId attribute (which is also rarely used).
   d. **gti_featureName** – specifies the name of the feature. If this attribute is not defined, the Feature Type for the reader will be used as the default.

5) When you are using the **styleNameMap_byCat** or **styleNameMap_global** modes, the **gti_styleName** entry is used to specify the name of the **Abstract Style Name** for the **Style Name Map**. The **Style Definition Id** is automatically generated (or looked up) and assigned to element's **Style Definition Id** property. This format property greatly simplifies the creation of the **Style Name Map** since you can specify the Style Name as a constant or as a combination of the feature's attribute values (using the

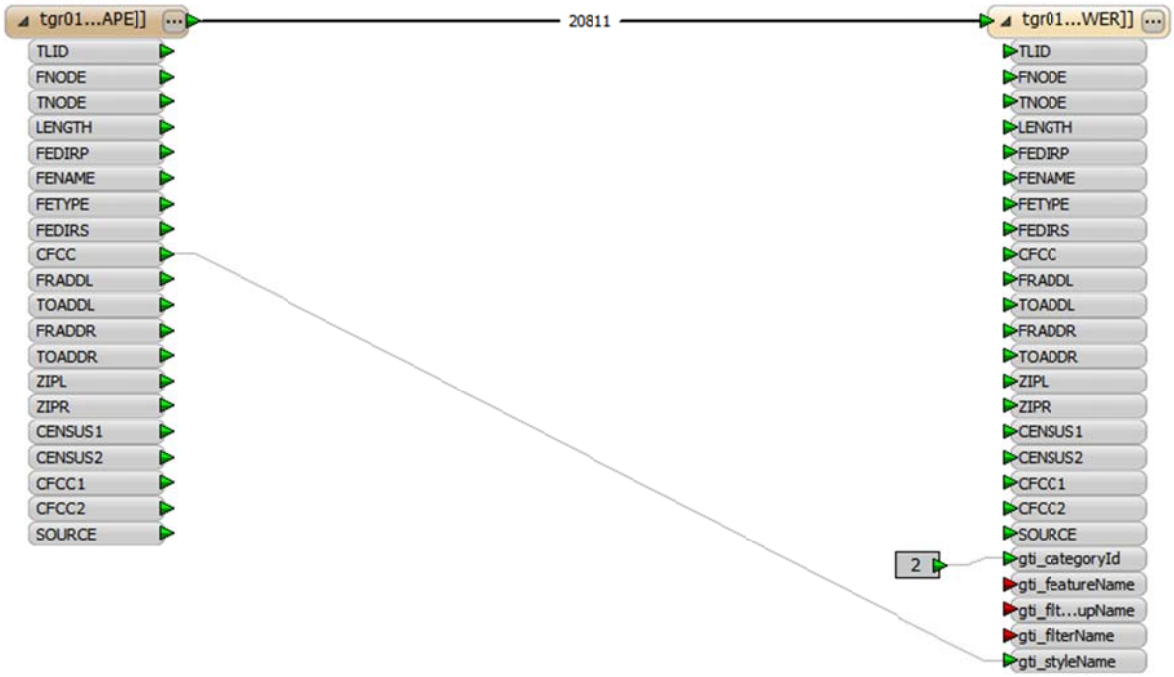
string concatenator transformer). The Navigator also has the **Use Feature Name in StyleName** option. If this option is set to Yes, then the Filter Group Name will be prepended to the style name automatically (just as it is in the Enhanced mode). If no Filter Group Name is specified, the Feature Name (defined by gti_featureName or the default Feature Type name) will be prepended. If the option is set to No, then it will just be the value you set it to. This option can greatly simply the setting of the style names if your style names between features are keyed on similar attribute values. Otherwise you would always have to use a String Concatenator transformer to add the Feature Name to the Style Name.

6) Grouping of the Filter Ids with the previously available modes was limited. With the **Enhanced** mode, you got a group for each Feature Type and all individual styles used by that Feature Type were the Filter Items in the group. This was a significant enhancement over the original method (**Standard** mode) which just put all of the Filter Items in one big group. With the **Style NameMap** modes, the grouping by feature type doesn't make as much sense since you don't have to organize a large number of styles for that feature into a group. You can just have one Filter Item for the FME Feature Type and they can be grouped however you like using the **gti_groupFilterName** format attribute. If you don't specify a **gti_groupFilterName**, it will default to "Features". You can still specify the Feature Type as a group and have any number of Filter Items in it by using the **gti_filterName** format attribute.

7) When creating your workspace, it is probably a good idea to expose the following format attributes for all of your writer feature type:

gti_categoryId
gti_filterGroupName
gti_filterName
gti_styleName
gti_featureName

8) For this example, the first feature type to configure is the street Shapefile (tgr01089lkA). I am going to set the following properties:
   a. Set **gti_categoryId** to 2 (even though this is the default).
   b. Set **gti_styleName** to the CFCC attribute. This is the Census Feature Class Code and is a good attribute for specifying different styles for the roads. A15 is an Interstate, A21 is a Primary road. These road types will be styled differently than the others.



9) Running FME will produce a dataset with a **Style Name Map** that looks like this:

```
[Id To Name]
1=TGR01089LKA - A41
2=TGR01089LKA - A21
3=TGR01089LKA - A51
4=TGR01089LKA - A63
5=TGR01089LKA - A31
6=TGR01089LKA - A15
7=TGR01089LKA - A35
8=TGR01089LKA - A25
9=TGR01089LKA - A71
10=TGR01089LKA - A74
11=TGR01089LKA - A11
12=TGR01089LKA - P41
```

```
[Name To Style]
```

An **Abstract Style Name** is created for each CFCC value found in the data. Since the **Use Feature Name in Style Name** option is set to Yes, the **Feature Type Name** is prepended to the value (in this example it is TGR001089LKA). If we had set the **gti_filterGroupName** or **gti_featureName** to a value like "Roads" then the Style Name Map would look like:

```
[Id To Name]
1=Roads - A41
2=Roads - A21
3=Roads - A51
4=Roads - A63
5=Roads - A31
6=Roads - A15
7=Roads - A35
8=Roads - A25
9=Roads - A71
10=Roads - A74
11=Roads - A11
12=Roads - P41

[Name To Style]
```

The Filter File created by this conversion would look like this:

```
[Filter Info]
1|1|0|0|100|FEATURES|GROUP:FEATURES,1|0|0|0|0|
2|1|1|1|100|ROADS|GIS1:1,1,FEATURES,ROADS|0|0|0|0|
```

If **gti_filterGroupName** is set to "Roads" this will change the group name to "Roads" like this:

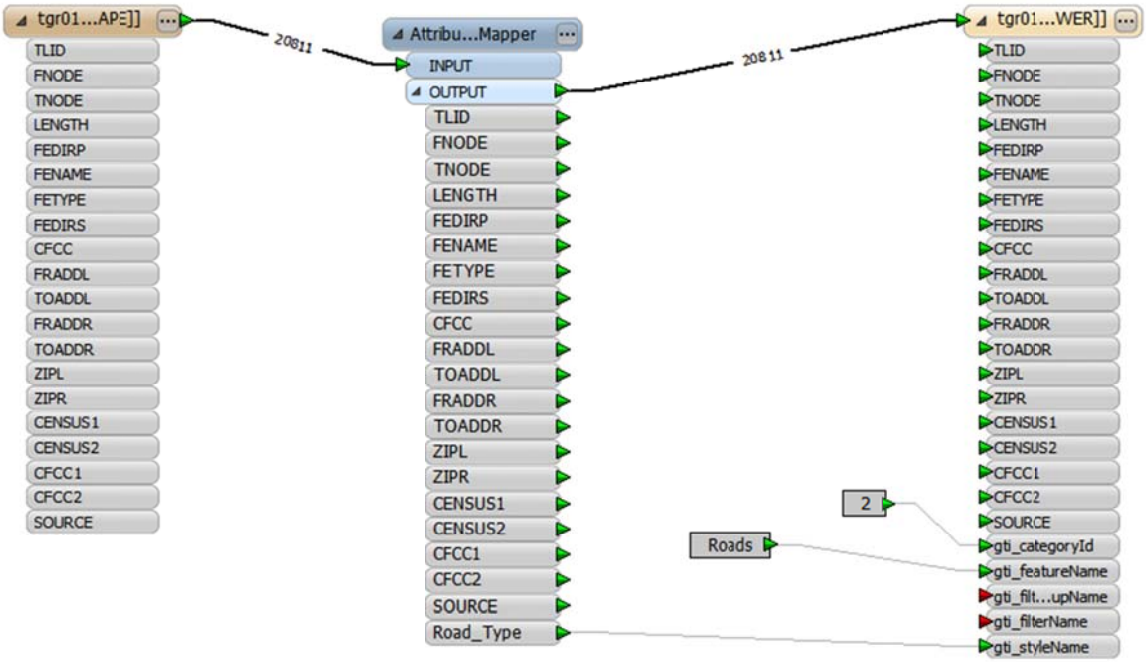
```
[Filter Info]
1|1|0|0|100|ROADS|GROUP:ROADS,1|0|0|0|0|
2|1|1|1|100|ROADS|GIS1:1,1,ROADS,ROADS|0|0|0|0|
```

In this case, setting the group name may not be very useful since the Group and the Filter Item are the same name.

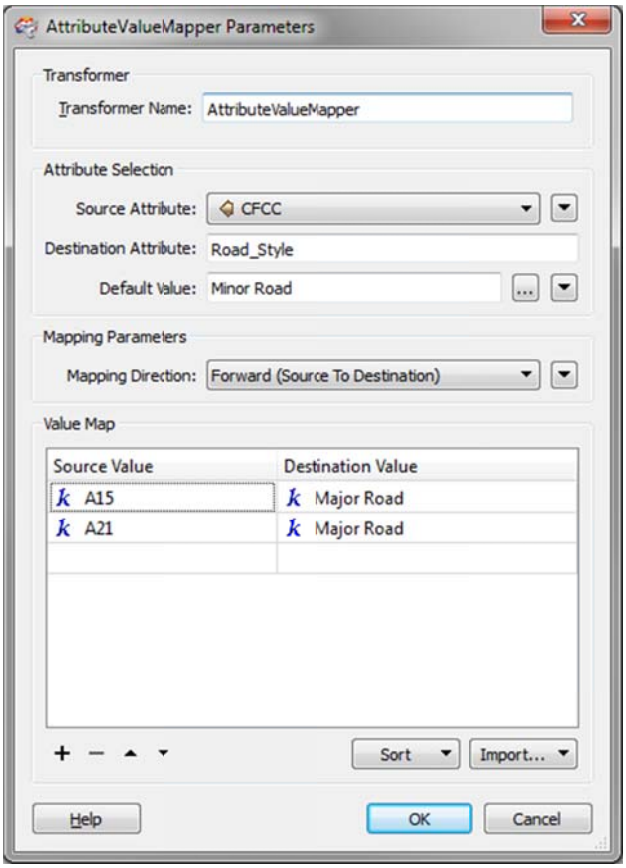
If **gti_featureName** (instead of gti_filterGroupName) is set to "Roads" this will change the feature name to "Roads", but not the group name like this:

```
[Filter Info]
1|1|0|0|100|FEATURES|GROUP:FEATURES,1|0|0|0|0|
2|1|1|1|100|ROADS|GIS1:1,1,FEATURES,ROADS|0|0|0|0|
```

10) We can expand this example to have the Major Roads and Minor roads with different Filter Ids by adding a transformer to base the **gti_filterName** value on the CFFC attribute. I will use a Value Mapper Transformer to map the value of A15 and A21 to "Major Road" and all others to "Minor Road". Then the Road_Type attribute will be set to the **gti_styleName** Format Attribute:



The Attribute Value Mapper Transformer just set the A15 and A21 value to "Major Road" and the default is "Minor Road"



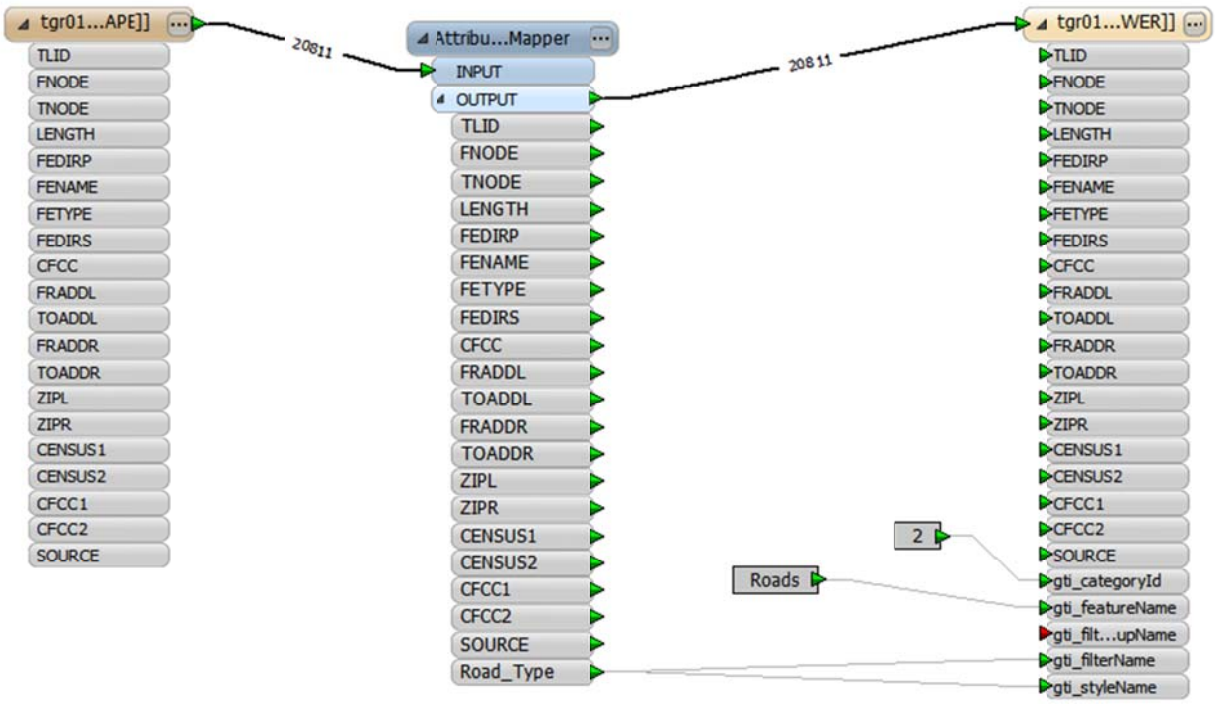
The Style Name Map is simplified to the following with this change:

```
[Id To Name]
1=ROADS - MINOR ROAD
2=ROADS - MAJOR ROAD

[Name To Style]
```

Since the **gti_filterName** is not set, the filter name will default to the feature name:

```
[Filter Info]
1|1|0|0|100|FEATURES|GROUP:FEATURES,1|0|0|0|0|
2|1|1|1|100|ROADS|GIS1:1,1,FEATURES,ROADS|0|0|0|0|
```

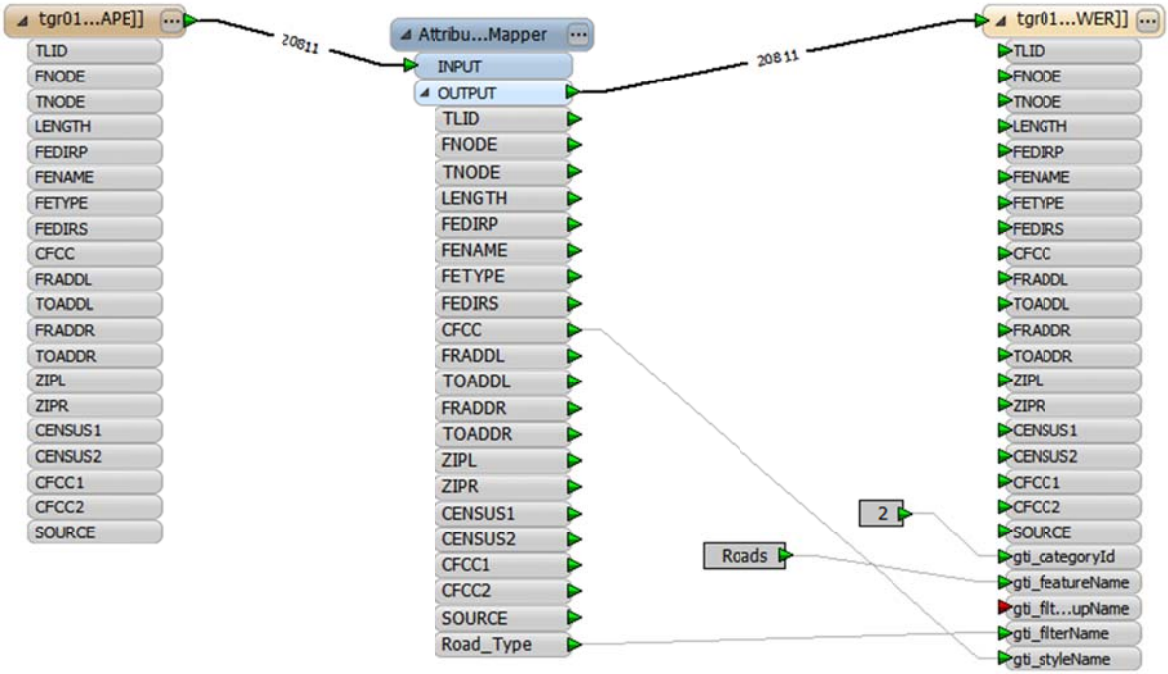
However, if we also set the **gti_filterName** format attribute to the Road_Type value produced by the Value Mapper transformer, we can have Minor Roads and Major roads as the two Filter File entries:



The Filter File will look like the following:

```
[Filter Info]
1|1|0|0|100|FEATURES|GROUP:FEATURES,1|0|0|0|0|
2|1|1|1|100|MINOR ROAD|GIS1:1,1,FEATURES,MINOR ROAD|0|0|0|0|
3|1|2|1|100|MAJOR ROAD|GIS1:1,2,FEATURES,MAJOR ROAD|0|0|0|0|
```

The above example is virtually identical to the old style Enhanced Filter Creation mode. To really show off what the Style Name Map can do, the example need to be a changed so that the **gti_filterName** and **gti_styleName** are not the same.



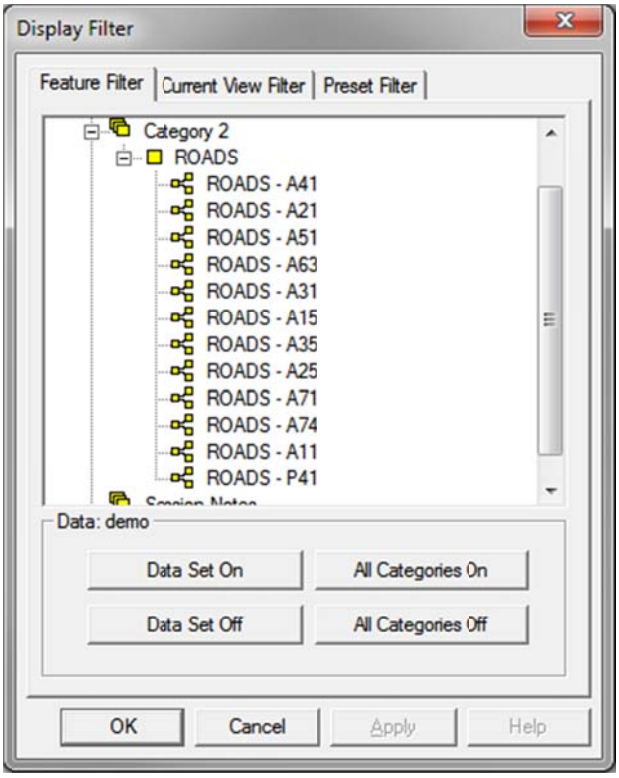
The only change here from the previous example is that the **gti_styleName** is set to CFCC instead of the Road_Type attribute. The **gti_filterName** is still set to Road_Type. The **Style Name Map** looks like this:

```
[Id To Name]
1=ROADS - A41
2=ROADS - A21
3=ROADS - A51
4=ROADS - A63
5=ROADS - A31
6=ROADS - A15
7=ROADS - A35
8=ROADS - A25
9=ROADS - A71
10=ROADS - A74
11=ROADS - A11
12=ROADS - P41

[Name To Style]
```

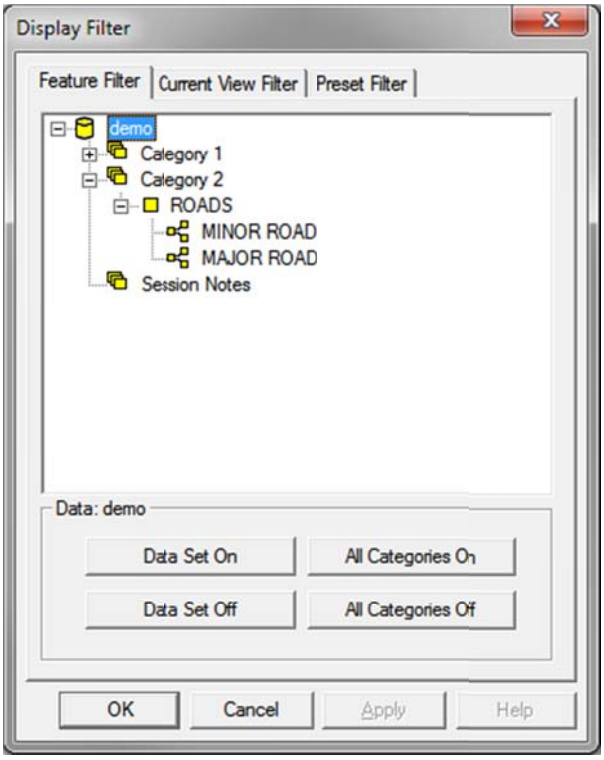
And the **Filter File looks** like this:

```
[Filter Info]
1|1|0|0|100|FEATURES|GROUP:FEATURES,1|0|0|0|0|
2|1|1|1|100|MINOR ROAD|GIS1:1,1,FEATURES,MINOR ROAD|0|0|0|0|
3|1|2|1|100|MAJOR ROAD|GIS1:1,2,FEATURES,MAJOR ROAD|0|0|0|0|
```

Now, we have 12 different **Style Definitions** used with only two **Filter Ids**. This could not be done with the old Style Map. And while this change is subtle, it does provide a powerful new way to organize your data. If you had wanted to keep all 12 Style Definitoin for the individual road type, the Filter Display would have looked like this with the Enhanced mode:



Whereas it looks like this with the **Style Name Map** while still having all 12 styles:



Conclusion

The **Style Name Map** and **Style Definition Id** are new tools for you to use. There is no reason to switch to this method of creating data unless you have a particular reason to. However, if you are just starting to convert your data to the GTViewer format or planning on converting new data, it is an option that you should consider.