目录

高级工作流技巧-1 2
Workbench 和要素流 2
变量 3
为什么要使用范围设置呢? 8
高级参数 10
使用参数来控制数据流 10
高级工作流技巧-2 14
多管道工作流 14
Startup 和 Shutdown 脚本 19
什么是 Startup 和 Shutdown 脚本? 19
创建一个脚本 19
访问 FME 变量 20
在启动脚本后停止 FME 20
Startup 和 Shutdown 的特点 20

北京世纪安图

高级工作流技巧-1



有些高级技巧对于处理在工作空间中的要素流程是非常有效的，但不是那么直观易懂，所以我们需要对背景知识有一定的了解。

Workbench 和要素流

你们可能忽略的一点就是，要素流以何种方式通过工作空间。关于它有很多的规则来规定此概念，相关的知识能让你更加有效地使用 FME 进行工作，避免一些不可预知的结果。

规则 1：要素按被读取的顺序来处理
一个数据集中的要素是被次序的读入到工作空间中的，数据集的读入次序是按照数据集出现在导航框中的先后顺序。一般，这个顺序是连续地，除非受到了某个特定转换结果的影响。

规则 1 产生的结果：
首先读取到的要素会第一个到达指定函数。一些函数利用特定的设置可以利用这个特点，让我们确认哪个要素可以第一个到达。

Clipper 函数的‘Clippers First’设置就是一个很好的例子，通过避免规则 3 带来的缺陷，以提高效率。
我们在：http://www.fmepedia.com/index.php/Clipper 中就能看到一个很好的例子。

规则 2：在工作流中尽可能地单独地处理每个要素。
这就是说，一旦读取了一个要素，FME 在对下一个要素进行读取之前，只会处理这个要素。每个函数对所有要处理的要素是按照次序处理的。

规则 2 产生的结果：
要是一个要素已经通过了第二个函数，而其他需要进入第二个函数的要素还没有被第一个函数处理完，就会产生未知的结果。对于设置了全局变量的函数就可能会受到影响。

规则 3：对要素进行分组（Group）处理的函数会保留所有的要素，直到所有要素到达可以进行处理为止。
要记住，函数是怎样处理单个要素，或是分组要素的？因为对要素进行分组处理需要所有的要素，所以这类函数会在内存中保留要素直到接受到所有要素。

规则 3 产生的结果：
函数在内存中保存要素会消耗系统资源。

如果有其它选择的话，对一个非常大型的数据集你就不会想进行这项操作。但是，你可以使用这类函数来有效地避免规则 2 产生的错误。

备注：要素可能以不同的顺序退出基于分组处理的函数（和它们进入函数的顺序不同）。

如果你觉得这些概念太复杂的话，下面的例子就会很清楚地解释这些规则，并且告诉你怎样有效地使用它们。

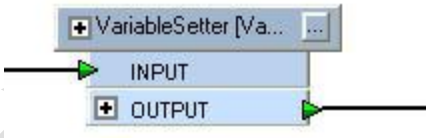
变量

它也叫做“全局变量”，是 FME 的一个功能，用来在不同要素流间传递信息；通常，从一个要素到紧接着它的下一个要素。

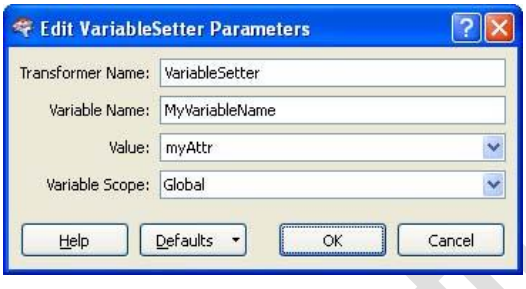
因为变量是处理数据流的一种方法，因此 FME 提供了 2 个与变量有关的函数，在函数分类 Infrastructure 下面。

VariableSetter

VariableSetter 函数是创建或设置变量值的一个工具。



以前我们也叫它 GlobalVariableSetter,，但是随着自定义函数的出现，现在去掉了“Global”这个前缀，因为在工作空间中有的变量仅仅适用于一个函数。



VariableSetter 函数有一些设置项（如左图）

Variable Name 指的是文本输入字段。变量的名字并不能通过下拉菜单进行选择。

要设置的变量值，可以是一个常量，也可以是从下菜单中选择的一个属性。

Variable Scope 可以是全局的也可以是局部的。Global 意味着在工作空间的任何一个地方都能访问它；Local 只能在自定义函数中被使用。

VariableRetriever

VariableRetriever 函数用来提取变量值，并将这个值放到一个属性内。同样的，它以前也有个前缀”global’。



VariableRetriever 和 VariableSetter 的设置类似（如左图）

Variable Name 是文本输入字段。

保存检索值的属性是一个文本输入字段，不能在下菜单中进行选择。

Variable Scope 可以是 Global，也可以是 Local。

Associative Arrays

关联数组就像一个矩阵，可以作为动态查找表。为了创建一个关联数组，可以使用@Value()功能将 variable name 设置成属性。

这种方法可能很难解释，但是 VariableRetriever 可以检索这样的一些值 (同样要使用@Value)，设置还能从值到属性进行反向查找。

全局变量和要素流

工作空间中的要素流对于变量来说是非常重要的，因为在下一个要素检索变量之前，我们必需要确定要素已经设置了一个变量。

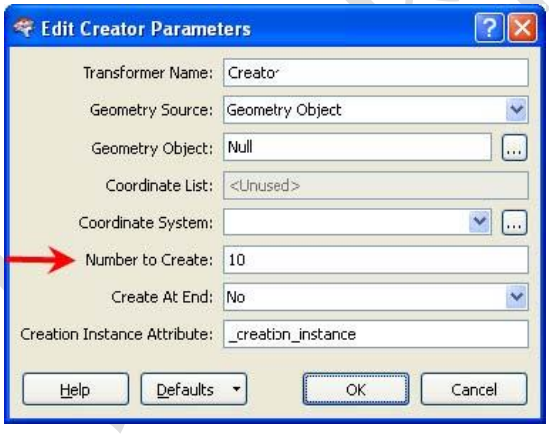
FeatureHolder 函数被用来控制要素流。



实例 1：要素流和 FeatureHolder

这里我们有许多要素，因此需要统计一共有多少个。通常，使用函数 StatisticsCalculator 来做，但是现在我们想使用变量来统计要素个数。

我们不使用源数据，而是在工作空间中创建数据。在 Workbench 中，打开一个空白的工作空间，放置一个函数 Creator。



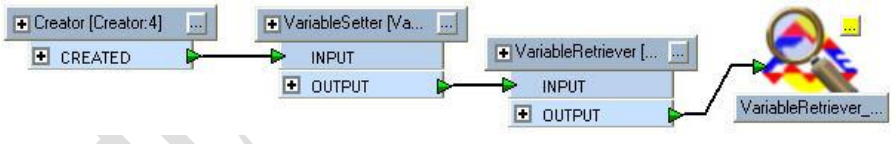
打开 Creator 设置对话框，将“number to create”设置为 10。

当要素被创建时，就会有有一个 ID 号 (_creation_instance)。

我们会使用变量，来找出最大的 ID 号。

将 Creator 与 VariableSetter 连接，选择_creation_instance 作为值域。

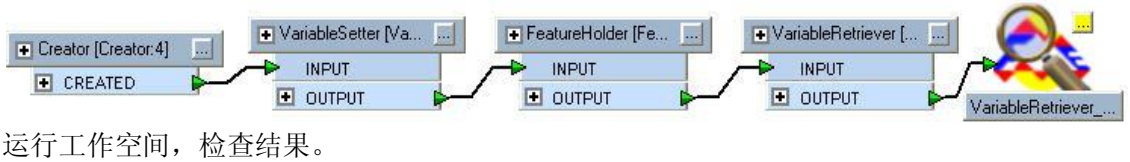
将 VariableSetter.与 VariableRetriever 连接，将‘attribute receiving value’改为 max_id，连接到 Visualizer。在 Visualizer 设置中通过属性 max_id.进行分组。然后，运行工作空间。



检查输出情况。要注意，max_id 总是与要素的 ID 相同的。这对于每个要素的单一输出类别非常重要。

同时也证明了个规则 2，如果同时用一个函数处理所有的要素，要素的 max_id 一定要是相同的。

在两个变量函数中插入一个 FeatureHolder 函数，它就会保存要素，直到所有要素都准备好了，才会进入到下一个流程——VariableRetriver。



运行工作空间，检查结果。

注意此时 max_id 在所有要素中都是一样的，并且所有的要素都出现在同样的要素类中。那是因为所有要素首先传递过 VariableSetter,然后再 FeatureHolder 处等待。此处遵循了规则 3。

练习 1：

现在我们来在一个真实环境中，操作这样使用 Variables。

在这个操作中，我们已经有了一个普通的 CSV 文件，它包括 X/Y 坐标系，需要基于此信息来建立线要素。并且，这些要素要被标记成属性。其实，这是一个很普遍的要求——因为 FME 能够处理一些格式（Z-Map, SEG-P1)。

具体步骤

1) 检查源数据

在开始数据转换之前，检查源数据通常都是第一步。因为源数据是一个 CSV 文件格式，所以你要在文本编辑器中打开它。

Source Format	Comma Separated Value (CSV)
Source Dataset	C:\FMEData\Roads\Roads.csv

你能够查看数据结构吗？其实，这并不如你所想的那么简单。道路要素的节点在文件中被顺序记录，但是并不是每个节点都有属性，每条道路都会使用一个“!”来作为开始标记。

添加头属性到每个要素的功能,就是使用变量的一个很好的例子（不需要添加任何相关的 ID 号）

2) 创建一个新的工作空间

打开 Workbench，创建一个新的工作空间，来将源 CSV 数据格式转换成“Generic (Any Format)”格式。必需要将源设置成“Has Field Names”。

3)过滤表头记录

首先我们要做的就是滤掉表头记录，从中提取属性 StringSearcher 函数(又名 Grepper) 用来进行这项操作。

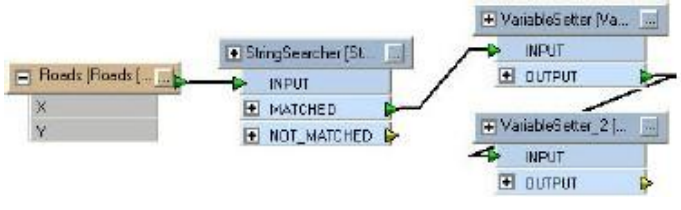
添加一个 StringSearcher 函数。需要添加一个表达式，来查找以!开头的记录。这个函数会告诉你这样将表达式定义为“starts with”。

将 StringSearcher 输出端口连接到 Visualizers，运行工作空间，检查你的设置是否正确。

4) 设置变量

我们已经滤掉了表头记录，现在要做的就是将它们保存为变量

添加 2 个 VariableSetter 函数（如右图）





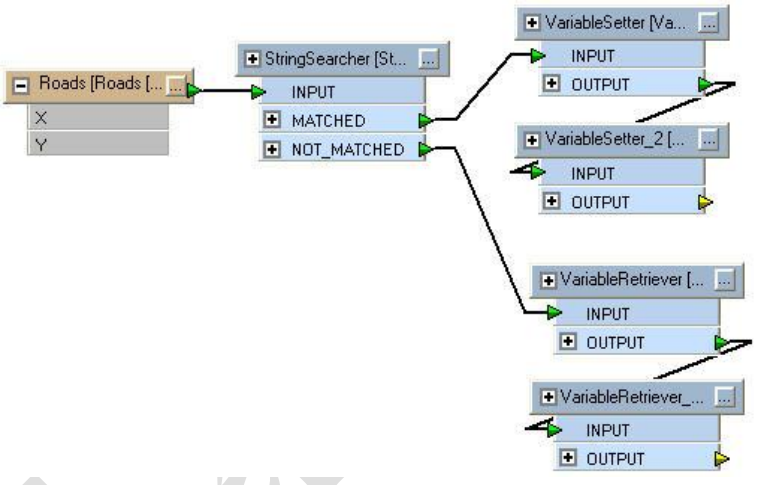
第一个变量（左图），命名 RoadCategory，获取属性 X 的值。

第二个变量，命名 RoadName，获取属性 Y 的值。

5) 获取变量

已经设置了变量，现在我们就需要检索它们。

添加两个 VariableRetriever 函数，来处理没有开头的记录。检索与 RoadCategory 相同属性名的变量，并且对 RoadName 进行同样的处理。



6) 处理点类要素

现在我们已经有了正确的属性，就能够处理 GPS 数据，并将它转变成线类要素。

使用 2DPointReplacer 和 PointConnector 函数，将没有开头的记录转化成线类要素。PointConnector 的属性应该是 RoadName

添加 Visualizer 函数，运行工作空间，检查输出是否正确。

7) 检查输出结果

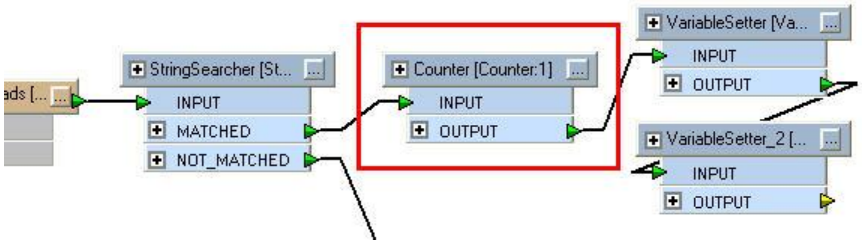
这些输出结果都是正确的吗？不是的。这是因为有许多相同路名的要素，因此我们最好为每个路名创建一个 ID 号，并将它作为区分属性的条件。

Counter 函数就可以为我们创建 ID 号。

但是，我们要将 Counter 添加到哪个位置呢？因为我们只想统计每个路径要素（而不是每个点），所以不能将它放在 PointConnector,之前。但是，如果不这么做，怎么能够将它作为区分属性的条件呢？

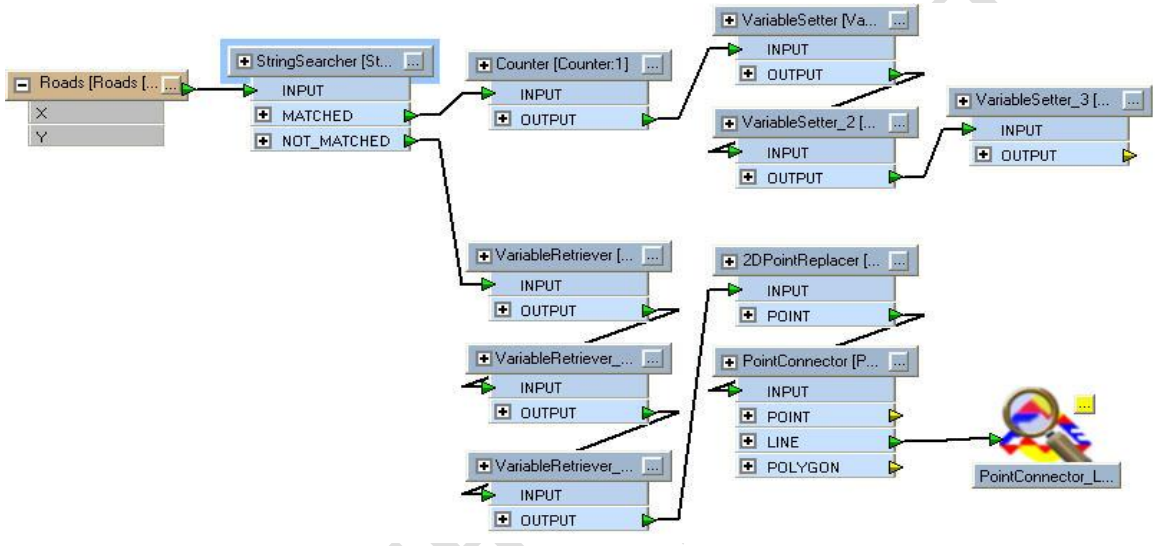
最好的解决方法就是，统计头要素，再次使用变量来转移 ID 号

添加 Counter 到头处理流程中，一定要在第一个 VariableSetter 之前。将输出结果属性设成 RoadID.



添加另外的 VariableSetter 和 VariableRetriever 函数，将 RoadID 转换成道路属性。

将 PointConnector 的区分属性更改成 RoadID (而不是 RoadName)。



这时，你的工作空间如上图，你的输出结果也如下图。

8) Fix RoadCategory

在 FME Viewer 中查询一个要素。

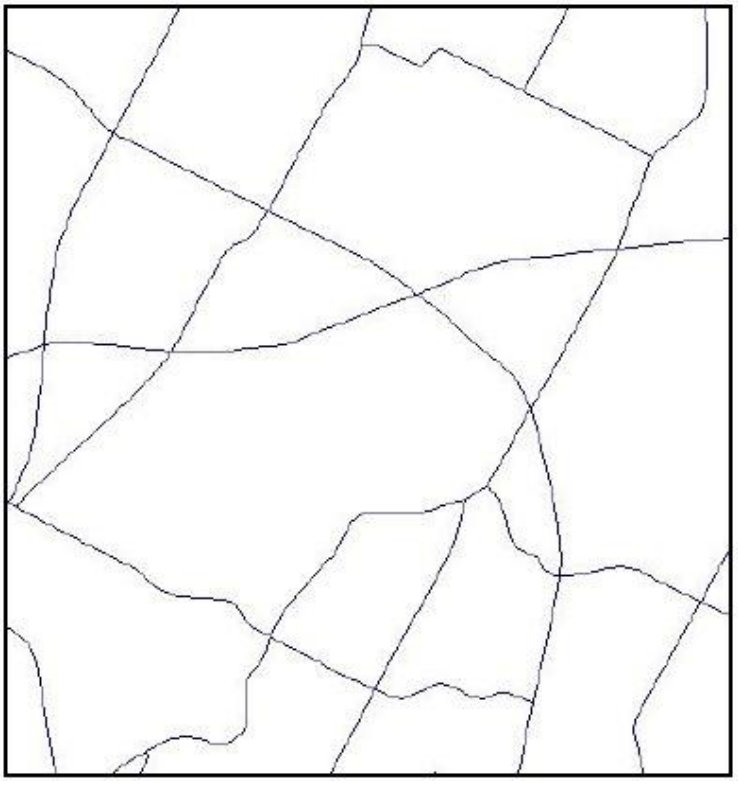
查看 RoadCategory 值。

在这个值前面仍然有“!”，所以我们要将它去掉。

使用 SubStringExtractor 函数或 StringReplacer 函数，去掉多余的属性。

因为属性值可能随着长度的变化而发生改变，所以不能仅仅使用固定值。

再次运行工作空间，检查结果。



9) 保存工作空间

我们接下来的练习还要在这个

工作空间中进行一些操作，所以要保存它。

你知道这个工作空间是怎样运行的吗？

我们依赖于数据被顺序的读取，并且按照同样的顺序来处理数据。在整个工作空间中，我们同样依赖于在一个时间内处理单一要素，而不是在一个函数处理所有要素。

讨论：下面一些情况可能会破坏到工作空间

- 如果 CSV 结构不同（例如，节点记录并不是连续地）
- 如果在 Variable transformers 前面使用基于分组处理的函数；例如，在 VariableRetrievers 之前使用 PointConnector – 你知道为什么会发生这种情况吗？

为什么要使用范围设置呢?

不仅能在 VariableSetter 和 VariableRetriever 中使用范围设置功能，也能在其他函数中，例如，Counter，进行设置，当多个函数的实例相互影响的时候，就有必要使用范围设置了。

当一个函数作为自定义函数使用，而这个自定义函数在一个工作空间中多个位置使用时，就有必要使用范围设置了。

当自定义函数在所有实例中都是独立使用的，那么就设置为本地；而自定义函数在所有实例中是相互关联的，那么就设置为全局。

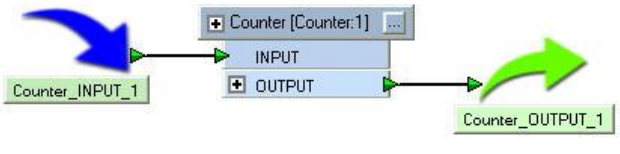
实例 2：本地/全局

我们想创建一个包含 Counter 函数的自定义函数。

首先，打开 Workbench.放置一个 Creator 函数 将 ‘number to create’设置改为 10。

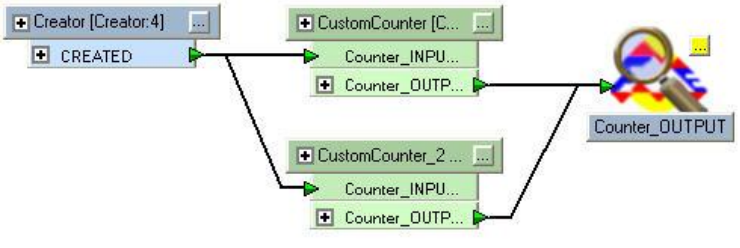
连接 Counter 到 Visualizer。

右击 Counter 函数，选择“Create a Custom Transformer”，将它命名为 CustomCounter 函数。



运行工作空间，你会得到从 0 到 9 的十个要素。

在工作空间中，放置另外一个 CustomCounter 函数—你可以对源函数进行复制，粘贴，然后查看它们。



将 Creator 和第二个 CustomCounter 进行连接，然后连到 Visualizer 函数(如左图)，就能创建数据的第二管道。

现在重新运行工作空间，你就可以得到 20 个要素输出....但是，它们是一组编号 0 到 19 的要素，

还是两组编号 0 到 9 的要素呢？

答案取决于自定义函数中 counter 的范围设置。

如果将范围设置成全局，那么自定义函数的每个实例都有相同的 counter 名称，就是一组编号 0 到 19 的要素。

反之，就是两组编号 0 到 9 的要素，具有不同的实例名称。

对于在工作空间中使用 Counters 来设置要素 IDs，这个设置是非常重要的。如果使用本地，就可能导致同一张表中有重复的 ID 号，如果使用全局，则可能导致多张表中的 ID 号丢失。

北京世纪安图

高级参数



你可能对使用发布参数来管理工作空间运行的设置很熟悉，但是，你知道吗，也能够使用这些参数来控制数据流。

使用参数来控制数据流

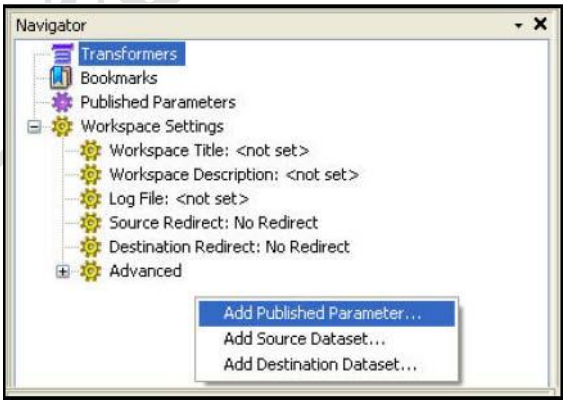
通常，控制通过工作空间的数据流取决于有关对象在不同时点的几何状态和属性。但是，FME 的不同功能能允许用户在运行工作空间之前做出某些控制数据流的决定。

第一个有助于实现我们目标的是自定义发布参数---- Custom (又名独立， Standalone or Independent) Published Parameters。

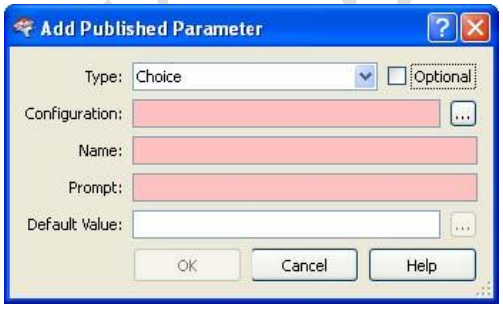
创建 Custom Published Parameters

这类参数不同于常见的已发布参数，它不依赖于任何函数，写模块，读模块，或要素类别。事实上，用户按照自己的需求在工作空间中创建它们。

右图：创建 Custom parameter 的不同方法



- 右击 Published Parameters
- 在导航框的任何空白区，点击鼠标右键（如右图）
- 使用菜单条中的插入选项



左图：用来创建发布参数的对话框要求很多字段。它们被标记成红色，只有填入了合适的内容，红色才会消退。

name 和 prompt 字段是完全相同的，就是常见参数使用的；Name 用来定义参数，prompt 提醒用户要输出正确值。



Default value 字段和用户设置参数的默认值很相似。

一个参数（发布了的，或是其它形式的），都可以被描述为一个全局常量，也就是说，开始进行

数据转换时，该常量就被设定，并且在整个转换过程中保持一致。

Custom Published Parameters – Other Settings

我们可以使用参数类型来创建不同的参数

Choice:	从下拉列表中选择的单选值
Multiple:	从下拉列表中选择的多选值
Choice or Text:	从下拉列表中选择或者允许输入值
Color Picker:	从色表选择中选择的颜色值，以 FME RGB 形式保存
Coordinate System Name:	从 FME 坐标系定义中选择的坐标系
Directory:	选择的目录或者文件夹
File Name:	选择的单个文件
Float:	浮点数
Integer:	整型数
Password:	口令输入
Text:	字符串文本

当我们需要为这些参数列别提供额外的信息时，就要用到 Configuration 字段。

Choice:	选择框
Multiple:	多选框
Choice or Text:	选择或者输入框
File Name:	文件名描述和后缀名

可选择字段也是新的。实际上，这就是一个检查框，它检查这个参数是否是可选择的，或是在运行工作空间前一定要设置一个参数值。

提取参数

因为独立的参数没有连接到任何的已有参数，读模块，或写模块，所以工作空间一定要有其它的方法来访问参数值，通过 ParameterFetcher 参数就能做到。

ParameterFetcher 提供所有已独立发布参数值。

工作空间的创建者将参数定义成可提取的，这样就能在属性中提取参数。

当数据转换进行到这一步时，已选的参数值就会发送到目标属性。



记住—参数值已经被设置了，你不能随意更改它，否则就不能对它进行读取，更新或是编写了。并没有 ParameterWriter 参数写模块。



参数列表包括所有添加到已有函数，读模块，或写模块的内容，这样就能允许用户获取参数名。通过输入参数名，用户也能够获取预定义值，例如，FME_BUILD_NUM，FME_HOME。

测试和引导到不同的处理过程
将独立的已发布参数和 ParameterFetcher 结合起来，就能允许工作空间设计者设置一个工作空间，在运行时通过不同的函数来发送要素。

例如，用户可能会被提示是否需要标记输出的数据，如果选择 YES，要素就会通过 labeller 函数进行发送，如果选择 NO，工作空间就会直接绕过这个过程。

在与 Generic Writer 一起使用时，这项操作是非常有用的；例如，Shape 格式就不用任何标记，因为它根本不支持文本几何对象。

练习 2
继续完成练习 1 未完成的部分。现在，我们要求用户以何种方式输出数据.

具体步骤

1) 打开工作空间
打开操作 1 中的工作空间，或是使用 Exercise2-Begin.fmw

2)创建 Custom Parameter
创建一个新的发布参数

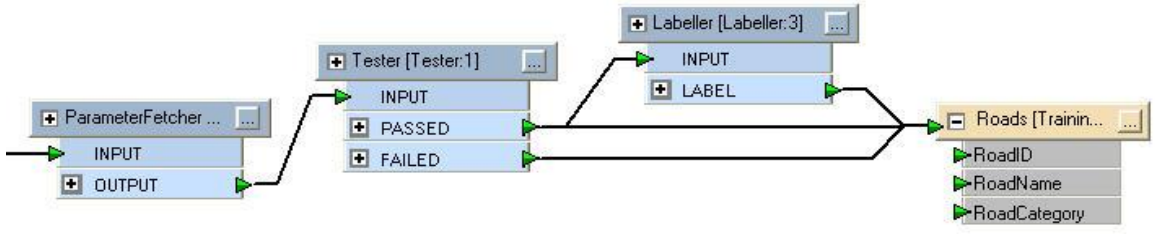
Type:	Choice
Optional:	No (i.e. don't tick the checkbox)
Configuration:	Two options – Yes and No
Name:	LabelRoads
Prompt:	Create labels from road names?

3) 放置 ParameterFetcher 函数
在最终输出之前放置 ParameterFetcher 函数，就能在 LabelRoads 属性中提取 LabelRoads 参数、

4) 放置 Tester 函数
在 ParameterFetcher 之后放置一个 Tester 函数，测试 LabelRoads 值是否是正确的。

5) 放置 Labeller 函数
将它与 Tester 的 PASSED 端口进行连接，并且将 Labeller Output 和.Tester FAILED 端口都连接到 Roads(目标要素类别)

因为 Labeller 函数将会取代道路要素，所以你需要将 PASSED 端口直接连接到输出端口（绕过 Labeller）
下图：就是工作空间的最后一部分。



6) 检查 Labeller 设置

按照以下要求更新 Labeller 设置

Lable Attribute：	RoadName
Lable Offset：	1
Lable Height：	500
Lable Spacing：	9999

7）Redirect Output

使用 File > Prompt 和 Run Translation，或者是 Ctrl+R 快捷键来运行工作空间。

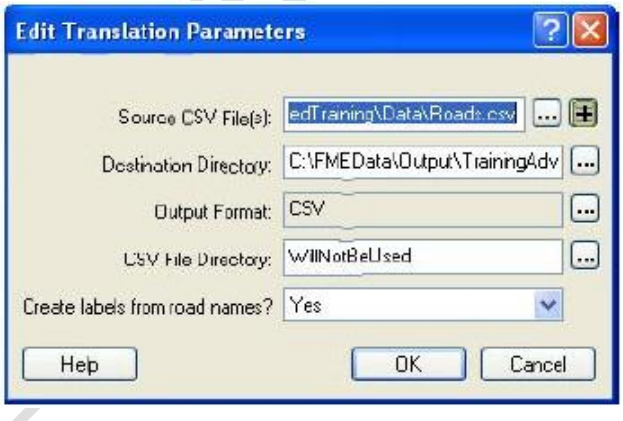
你会被提示是否要创建标记，选择 YES 或 NO，尝试运行工作空间。

8) 清空对话框

现在你可以随时打开或关闭标签。

然后，在运行的时候打开的对话框不太整洁，它可能有些额外的参数，如右图所示。删除以后不再需要的发布的一些参数，如下图所示。





高级任务

发布 Generic Writer Output Format 参数（如果还没有发布的话）

你现在可以在运行时对话框中设置该参数，并使用 ParameterFetcher 来获取选定的格式。

在本测试中这个函数是用来提取格式参数，并且检查它的格式是否为 KML。如果是 KML 格式，就要使用 KMLStyler 函数，来设置线性颜色和宽度的符号。这个格式化参数名是 OGCKML。

是否还需要设置其它的参数来写入 KML?
如果不使用这个参数就开始数据转换，会碰到问题！那么，你应该使用哪类参数来解决这个问题呢？
如果你真的想要一个 KLM 输出文件，就一定要记得取消对输出的重新定向。

高级工作流技巧-2



将数据分流可能会产生许多有创意的方法来进行数据处理。

多管道工作流

多数情况下，你不想在工作空间中复制冗余数据，因为添加额外的要素就会增加系统资源负载。

但是，有时使用多管道（又名数据流）工作空间是非常有效的，因为你能够保留要素的原始状态，仅仅在复制后的要素上附加处理结果。

例 3：SelfIntersectionFilter

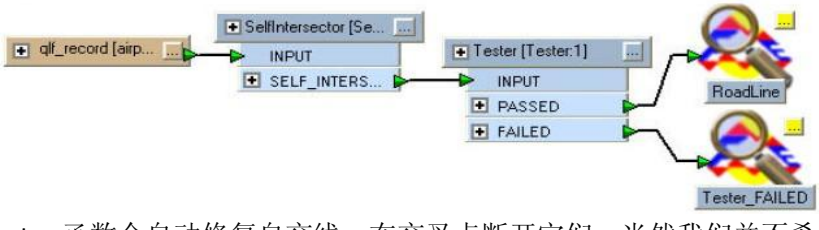
我们来创建一个 SelfIntersectionFilter 自定义函数。首先，我们需要找到自交线，但是不尝试修复它们。

在一个空白工作空间中，添加一下源数据：

Source Format	CITS Data Transfer Format (QLF)
Source Dataset	C:\FMEData\Data\Airport\airport.qlf

添加 SelfIntersector 函数，然后添加 Tester 函数来检查 where _segments > 1，通过这个设置就能帮助我们将自交线和非自交线区分开来。

添加 Tester 输出端口到 Visualizers ，然后运行工作空间。



你就可以查看哪些要素是自相交线，但是，SelfIntersector 函数会自动修复自交线，在交叉点断开它们，当然我们并不希望这种情况发生。

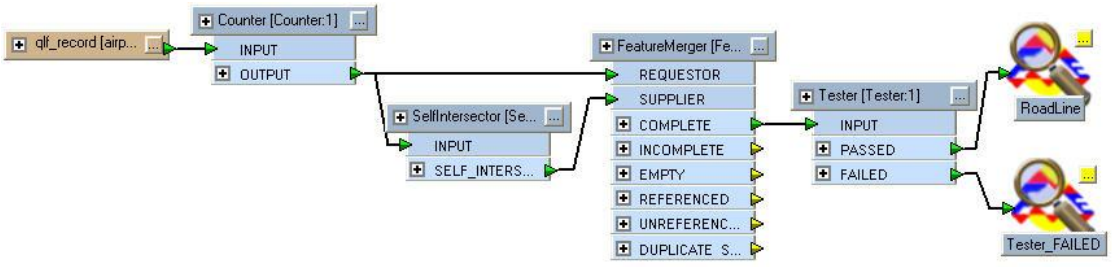
最简单的解决方法就是保持最初的源要素状态，通过融合自相交线的拷贝的信息，仅仅附加标志，表示它们是自交线。

在 SelfIntersector 之前添加一个 Counter 函数，就能给每个要素唯一的 ID。

添加一个 FeatureMerger 函数。创建 Counter 和 FeatureMerger REQUESTOR 端口之间的新连接，把被打断的要素连接到 SUPPLIER 端口。使用新的 ID 属性，连接被打断的信息到原始要素。

将 FeatureMerger COMPLETE 端口和 Tester 进行连接。

下图就是现在的工作空间



重新运行工作空间
现在你就能够区分出自相交要素，但是原始几何要素没有被改变。

讨论

作为一个高级任务，为什么不试着将它转变为自定义函数—SelfIntersectionFilter 呢？如果定义成自定义函数，在操作时，一定要考虑是否进行范围设置，全局的还是本地的？

相比数据清理这个工作空间是一个非常有趣的质量检查的例子。FME 会自动清理数据，这样就能确保将数据加载到目标数据集，但是，用户有时不想进行数据的清理，只是需要计算机提示在什么地方需要修改数据。

可以使用其它的 FME 函数来执行这项任务吗？通过设置 SpatialRelator 能测试相交，但是你能够将它变成自相交线测试吗？（注：我没有尝试过，所以不要认为这是一项高级任务。）

北京世宏宇图

练习 3：
这个练习继续完成练习 2 中未完成的操作。
我们想找出有哪些道路被河流或小溪穿过。

LineOnLineOverlayer 函数能够执行这项操作，但是也会在每个交叉点分割道路，很显然这对于正确的拓扑网络结构中是不能被接受的。

所以，我们要使用之前描述的方法来解决这个问题。

具体步骤

1) 打开工作空间

打开练习 2 中的工作空间，或使用 Exercise3-Begin.fmw

2) 添加水系数据

在工作空间中添加有关河流和小溪的数据集

Source Format	MapInfo MIF/MID
Source Dataset	C:\FMEData\Data\Hydrography\HydrographyLine.mif

3) 添加 LineOnLineOverlayer 函数

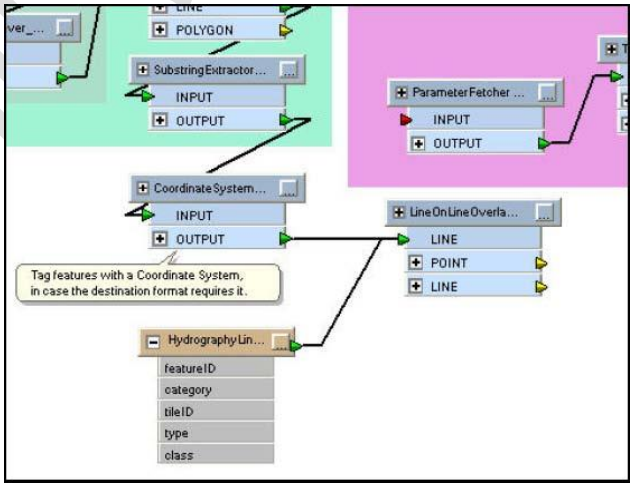
添加一个 LineOnLineOverlayer 函数。

它需要在线创建之后添加，而且可能需要放置在用户自定义的部分之前，以免在后面需要用到的时候出现问题。

在第一个 ParameterFetcher 前连接 LineOnLineOverlayer。

同时也要连接到水系要素。

右图：你的工作空间应该是这个样子。



4) 添加 Tester 函数

这个函数能帮助你检查哪个道路要素会和水系要素相交，并将道路要素交叉点分离出来。这个测试要在交叉点进行检测，交叉点来自 overlayer 函数的 POINT 端口（不是线端口，我们会通过所有数据集的一组属性来辨别交叉点；只有点类输出要素有这些属性）。

将 Tester 连接到 LineOnLineOverlayer's POINT 输出端口。

设置 Tester 来检查包括水系重叠的点类要素。因为这些要素有一组水系要素数据集属性，所以你就可以检查 featureID，category 等。

5) 添加 FeatureMerger 函数

添加 FeatureMerger 函数，就能够把得到的交叉结果传回道路要素。交叉点上的信息是 SUPPLIED，道路要素就是 REQUESTING。

联接属性是 RoadID。

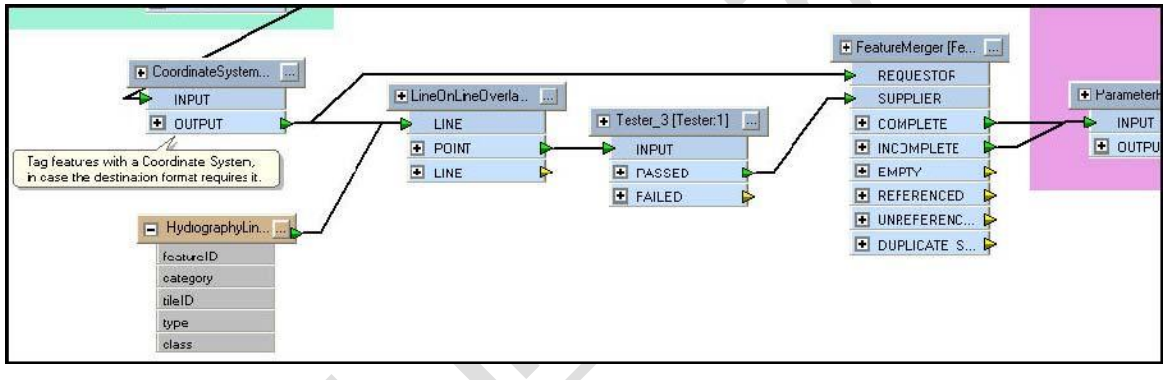
这部分会滤掉水系与水系相交的交点，因为它们不会匹配 RoadID，就会忽略掉它

6) 重新连接道路要素

FeatureMerger 中的 COMPLETE 要素指的是道路要素，这些道路要素已经被附加了一个水系交点。FeatureMerger 中的 INCOMPLETE 要素指的是道路要素，这些要素没有和水系相交。

这两类要素都要重新连接到源 ParameterFetcher。

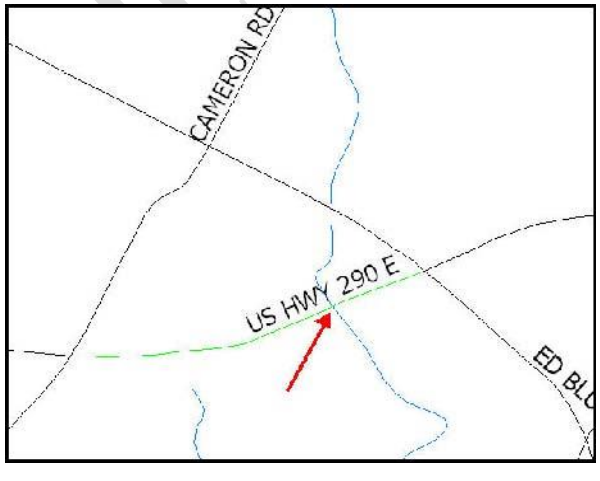
下图：对工作空间进行整理后，如下图所示：



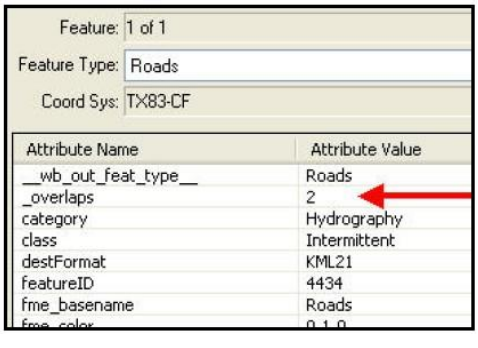
7) 运行工作空间

运行工作空间，查看输出结果。所有和水系相交的道路要素都有一个属性_overlaps ，它的值大于 1。

在 Viewer 中叠加水系要素来检查工作空间的输出结果。



左图：道路 HWY290E 和水系有个交点在高亮处，因此它有一个_overlaps 属性（如下图）



高级任务

现在工作空间就是完整的了，但如果你想进行其它的操作，就可以看看下面的这些附加任务。

1) 水系相交标注

这个操作非常简单，只需要使用 AttributeCreator 来创建一个属性 OverlapsHydrography，就能更好地标注与水系要素相交的道路要素。

2) 输出交叉点

为了显示桥的位置（交叉点），我们可以创建一个新的目标要素类别，它保存了道路和水系的交叉点。

连接正确的数据到这个要素类别（这需要思考一下）。

右图：你会发现到这个特殊的桥居然消失了。你能找出原因，并解决这个问题吗？



添加一个新的发布参数，询问工作空间的用户是否需要输出这部分数据，然后根据这个问题可能产生的结果筛选数据。

3) 清空对话框

删除 mid/mif Hydrography 的提示，就能清空运行时的对话框。

你是否能调整出现在对话框中的提示顺序呢？

4) GeometryReplacer

使用 GeometryExtractor 和 GeometryReplacer 函数，就能进行类似于复制数据—要素整合（FeatureMerger）的操作。

GeometryExtractor 保存一个原始要素，当处理数据时，GeometryReplacer 会发送回这个原始要素。

这个操作相当复杂，但是你能够调整工作空间来进行此项操作吗？关键是，怎样辨别相交的要素，怎样提取这些要素？

在运行这个工作空间时，比以前的工作空间是要快还是慢？是否需要更多的内存么？

如果你不能进行这个操作，可以检查已完成的工作空间。

Startup 和 Shutdown 脚本



有些高级技巧对于处理在工作空间中的要素流程是非常有效的，但不总是那么直观易懂，所以我们需要对背景知识有一定的了解...

什么是 Startup 和 Shutdown 脚本?

它们是 FME 的重要组件，因为通过它们，你就可以在开始工作空间之前，或是关闭工作空间之后，能够运行脚本。

一般来说，我们运行启动脚本来检查一些可能导致工作空间发生错误的问题。如果我们花了大量的时间来进行数据转换，但结果却不正确，比起这种情况，在开始转换之前就找出问题所在，则要有效的多。同时，运行脚本也可以执行一些简单任务，例如，检查源数据，或是检查输出数据集是否已经存在。

我们通常使用关闭脚本来以其它的方式来处理数据转换输出，例如，将结果文件复制到新的位置，或者创建自定义日志记录—例如，如果 FME 转换发生错误的话，就可以发邮件给管理者（或者发布到 Tweet 上）。

脚本使用的语言有两种：TCL（工具命令语言），Python。FME 对这两种语言提供了内嵌的编译器，所以我们可以直接使用这些脚本语言，而不需要额外的安装。



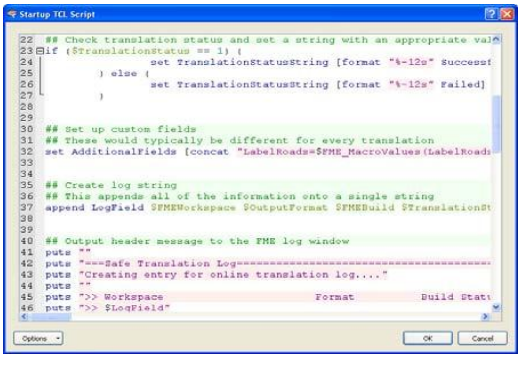
创建一个脚本

我们可以在外部创建脚本，然后通过 FME 访问它（使用 TCL 或者 Python 输出命令），或者嵌入到一个映射文件，但是这些方法都比较复杂，最简单的方法就是选择导航框中的高级工作空间设置，就能创建和发布脚本。

左图：使用 TCL 和 Python 语言来定义启动和关机脚本。

下图：双击脚本设置，打开对话框，点击省略号键，就可以打开编辑对话框（如右图）





访问 FME 变量

关闭和启动脚本能够访问不同的 FME 全局变量，发布参数，以及预先定义的宏。这部分我们使用的是 TCL 语言，而不是 Python。

FME 全局变量

TCL 关机脚本能够访问许多表示转换统计的全局变量，例如，FME_Status 会告诉用户转换过程失败了（0），或是成功了（1），FME_ElapsedTime 会告诉你完成转换花费了多长时间。

使用以下的语法，就能非常简单的访问到全局变量（以 FME_Status 为例）：

```
set    myVariable $FME_Status
```

在 FME 基本手册中已经列出了所有的全局变量，另外，启动和关闭 TCL 脚本共享一个通用编译器，也就是说，你可以在启动脚本时创建自己的全局变量，在关闭脚本时访问这些变量。

发布参数

在 TCL 中，使用 FME_MacroValues 就能访问发布参数：

```
set myVariable $FME_MacroValues(myPublishedParameter)
```

预先定义宏

你可能对一些预定义宏，例如 FME_MF_DIR 很熟悉，但是，事实上还有许多其它的宏，例如：

```
set myVariable $FME_MacroValues(FME_BUILD_NUM)
```

对于 Python，你以前需要设置 import __main__ 和 use __main__.FME_MacroValues；但是对 FME2009 ，你只需要简单地使用 pyfme，例如：

```
import pyfme
myVariable=pyfme.FME_MacroValues['FME_BUILD_NUM']
```

在启动脚本后停止 FME

当启动脚本运行失败后，FME 会停止转换过程。如果你想在脚本发生错误后，仍然进行转换，就需要找出错误。通过使用 error 指令来结束 TCL 脚本，能让用户有目的地阻止数据转换。

Startup 和 Shutdown 的特点

- ESRI 用户可以使用 Python 脚本来访问 ArcObjects，也就是说能够运行脚本，在转换之前删除拓扑，并且在转换后重建拓扑
- 不能在自定义格式（fds）文件中使用启动/关机脚本
- TCL 不能访问 FME Objects 功能，也就是说它不能使用任何函数或者管道工厂。但是，Python 能够访问一些函数，但是这些函数是不需要要素的。
- 可以使用多组脚本，但并不推荐。
- 通过执行 exec 命令，TCL 脚本就能访问另一个工作空间或者 FME 过程，但是， 你应该直接输出 exec 命令，通过附加 2>NUL：

更多有关启动和关闭脚本的信息可以在 FME Workbench 帮助，FME 基本培训手册或是

fmepedia.com 网站中获取。

练习 4

这个练习会完成练习 3 未完成的任务

当转换过程完成，我们通过运行 TCL 脚本来创建自定义转换日志，提醒管理员出现了什么问题。

不要担心—你不需要任何有关 TCL 和编程方面的知识。

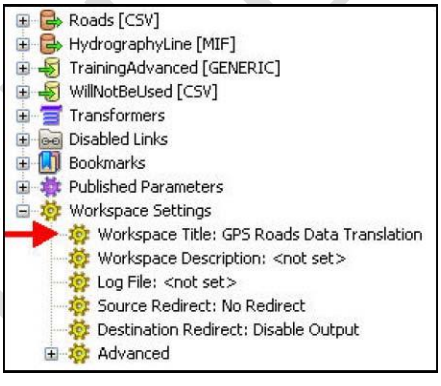
具体步骤

1) 打开工作空间

打开练习 3 中完成的工作空间，或者使用工作空间 Exercise4-Begin.fmw

2) 添加工作空间名

为了创建一个日志输出，我们需要知道工作空间的名称。导航框中的工作空间标题已经连接到了 MAPPING_FILE_ID。



如果我们要设置一个标题，通过设置 MAPPING_FILE_ID，就能通过使用全局变量 MappingFileId 来获取这个值。

在导航框中，找到 Workspace Title 设置，将它设置成 GPS Roads Data Translation（如右图）：

3) 创建关闭脚本

现在我们来创建一个关闭脚本。事实上，之前我们已经创建了，所以我们只需要将它复制粘贴到正确的位置。

在导航框中找到 Shutdown TCL Script，双击这个设置，打开脚本编辑窗口。

打开文本文件 C:\FMEData\Resources\Miscellaneous\TclScript.txt ，将文件的内容复制粘贴到 Shutdown TCL 脚本编辑对话框；或者，引用它作为 TCL 源命令。

注：对 Python 用户而言，脚本放置在：

C:\FMEData\Resources\ Miscellaneous \PythonScript.txt

4) 检查脚本

在开始转换之前，我们看一下脚本。

总的来说，脚本会读取很多转换参数，创建一个单一的字符串作为转换日志条目。转换日志会用一个固定的格式，保存所有转换的总结。

我们准备使用的转换日志是一个在线日志，它接受 HTTP POST 请求，这个请求使用 TCL 脚本来创建和发送。

浏览: www.safe.com/results 网站，你会看到这个在线日志。

我们可以在文件夹 C:\FMEData\Resources\ Miscellaneous 中找到这个网站的 PHP 脚本和 CSS 样式。PHP 脚本包括基于 MySQL 数据库的结构，但是服务器名，用户名，以及密码是没有的。

把转换过程写入到这个日志，脚本可以检查转换过程是否执行成功。如果转换过程发生了错误，就会给系统管理员发邮件，通知他们发生了错误。

想了解更多的信息，就查看脚本—脚本本身写的很清楚，并且有详细的注释说明。我们唯一不熟悉的可能是 http 和 mime/smtp 指令，虽然这些指令是 TCL 存储库的一部分，我们不需要单独安装它们。

5) 运行转换

现在我们来开始转换。一定要注意 FME 日志窗口的附加信息。



注意：检查发布参数名是否与在 TCL/Python 脚本中一致，如果不一致，会出现错误“no such element in array”的提示。

检查 www.safe.com/results （如果有需要就刷新页面）你就会在页面顶部看到转换结果（如下图）。



6) 终止转换过程

这次转换是成功的。现在我们要想出一个方法来终止转换，并且再次运行它（提示：一个函数带有错误的设置—例如 2DPointReplacer，或者是一个数据格式带有不支持的几何类型，就一定会导致转换错误）。

注意一下转换过程是怎样报告错误的（与成功转换是不同的）。

当然，也有可能不能成功的给管理者发送邮件，因为脚本的账号信息是假的（但是要注意脚本是以何种方式捕捉到这个错误的）。

如果你已经知道了邮箱的账号信息，或你已登录的服务器的信息，你可以在脚本中输入正确账号，并且运行它，来证明这个脚本是可以工作的。

7) 恭喜

至此，你已经成功地完成了这个练习。现在你就有了一个工作空间，它能将一些自定义源数据转换成用户选择的格式（包括用户定义的内容和样式），同时进行了一组空间要素处理，这些操作保留了转换的管理和元数据的过程。



高级任务

如果你想完成最后的任务，为什么不创建一个参数呢？这样用户就可以选择是否要将它们编写到在线日志中。通过这个设置，就可以检测工作空间，但却不会被

反复提示发生错误。这样，你就需要编辑 TCL 脚本了。

同时，如果你既是一个 Python 专家，也是一个 Twitter 爱好者，就可以尝试让 FME 来给 tweet 发送信息。你需要安装一个 Python Twitter API，例如 http://pypi.python.org/pypi/twitter/1.2.1，使用函数 PythonCaller 来发布正确的指令。

北京世纪安图