

目录

通用工作空间.....	3
为什么要使用通用工作空间.....	3
实例场景.....	3
什么情况下使用通用工作空间.....	4
通用格式.....	5
通用编写器.....	5
Example 1-通用编写器.....	6
通用源层.....	7
未定义的要素类.....	7
要素类选择.....	8
要读取的要素类.....	8
Example 2-Feature Types to Read.....	10
通用目标层.....	11
Data Fanouts.....	11
要素类型扇出.....	12
Feature Type Fanout Basics.....	12
Example 3-要素类扇出.....	13
数据集扇出.....	15
Dataset Fanout Basics.....	15
Example 4-数据集扇出.....	16
批量处理扇出.....	18
Methodology.....	18
Drawbacks.....	19
通用属性.....	20
动态模式.....	20
实例场景.....	21
Example 5-通用编写器和动态模式.....	22



通用工作空间概述.....	24
表格综览.....	24
使用需知.....	25
单元复习.....	26
你应该从这单元中学到什么.....	26

北京世纪安图

通用工作空间



通用工作空间指的是以最简单的方法，进行多个转换

为什么要使用通用工作空间

总的来说，FME 转换比较容易，但是在不尽量不改变工作空间的情况下，对它进行复制就不那么容易了。

我们并不推荐将多个编写器添加到一个工作空间，除非是真的有这个需求。

这种行为，就像是给工作空间的创建增添不必要的工作，只会加重终端用户的工作量，因为他们不得不在两个很形似的转换中挑选一个。

一个通用工作空间就是说，你可以反复使用它作为多种用途，并且不必要每次使用时都编辑它。

实例场景

在以下情况下，通用转换的优点：

- 源数据转换保存不变，但是输出格式却发生了改变
- 输出格式保存不变，但是目标坐标系发生了改变
- 格式保持不变，但是数据转换却发生了改变
- 格式和转换都保持不变，但是输出数据集的数量和结构都发生了改变
- 或者是，这四种情况的结合



Police Chief Webb-Mapp 说过...

“当你在安装 FME Server 时，通常会碰到上面的情况，所以这单元讨论的问题对于用户创建工作空间是非常重要的。”

什么情况下使用通用工作空间

在一个 FME 工作空间中存在大量的组件，如下：

- **Format**
在不需要添加多个阅读器或编写器的情况下，这个工作空间就可以读取任何格式的数据，并且编写到任何格式的数据集中。FME2008 中的 Generic Write 支持编写
- **Schema: Layers**
这个工作空间可以从源数据集中读取任何一个层，并且编写任何一个层。
Merge Filter 和 Data Fanouts 支持它。
- **Schema: Attributes**
无论被定义的模式是什么，这个工作空间都可以读取任何一组属性，并编写它们。
FME2009 中的 Dynamic Schemas 支持这个工作空间。

最后，FME 用户得到的版本就是只含有一个阅读器和一个编写器的工作空间，可以灵活的使用它以任何模式来读取任何格式的数据，并且编写成其它格式。

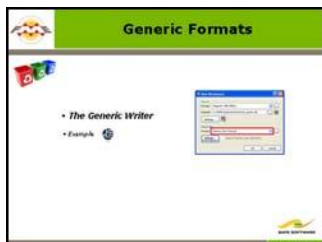
- ✧ 现在就可以获取这个版本，但是它并不只是一个发展，因为你可以发现在许多情况下都用到了 FME 的通用功能。

通用阅读器

现在它还只是一个概念，但是 FME2010 版本就会使用到它。

它与通用编写器很相似，就是说：可以支持任何格式数据集的阅读器。

通用格式



最开始通用编写器是作为一个工具，使用它工作空间就不需要依赖于目标格式了。

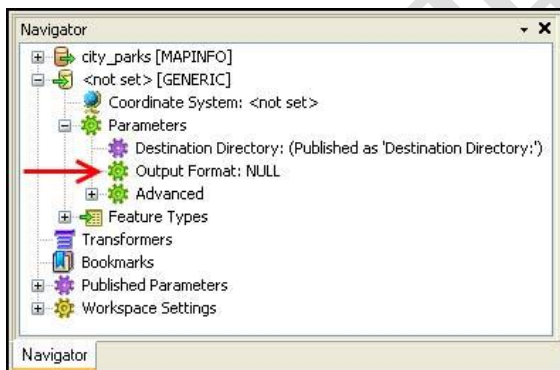
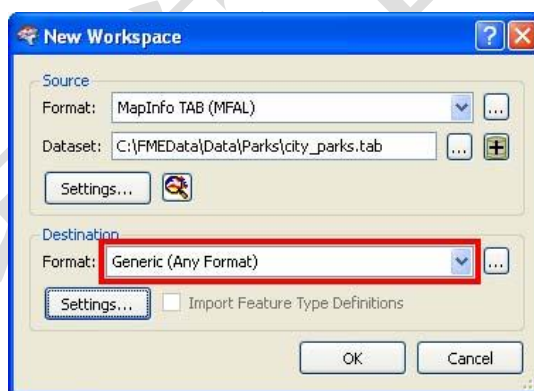
通用编写器

它是一个实物，或格式，它表示一个不含有特殊格式的编写器。

当我们运行含有通用编写器的工作空间时，就可以设置 **Navigator** 方框中的一个函数，来决定要编写的数据格式。

右图：创建一个工作空间，使用通用编写器来转换有关公园的数据。

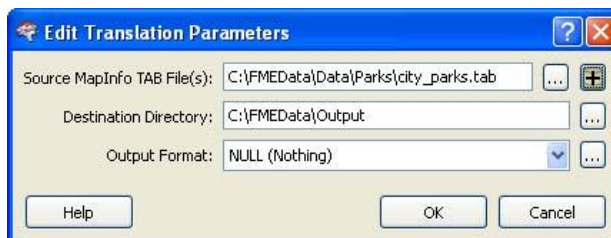
一旦创建了工作空间，用户就可以选择要编写的格式了。



左图：**Navigator** 方框不仅仅显示了一个目标位置参数，也显示了目标格式参数。

备注：目标位置通常是一个目录，即使选择的格式是文件形式的。

右图：因为可以发布目标格式参数，所以在操作时，按照提示用后要选择输出格式。



Example 1-通用编写器

这个例子展示了通用编写器的使用方法。

目的： 我们想创建一个工作空间，它会转换 INTEROPOLIS 城市的中转数据，并且将它上传到 FME Server 中，这样这个城市的居民就可以在互联网上下载数据，并且我们希望用户能够自己选择数据的格式。

具体步骤

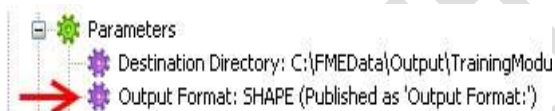
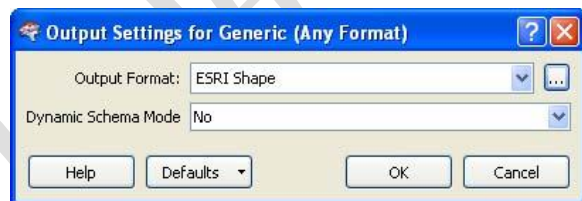
1) 开始转换

启动 Workbench，将 Autodesk SDF 转换成 Generic (任何格式)。

Source Format	Autodesk MapGuide Enterprise SDF
Source Dataset	C:\FMEDData\Data\Transit\Transit.sdf
Destination Format	Generic (Any Format)

在目标设置中，进行以下设置：

Dynamic Schema	No Output
Format	ESRI Shape



2) 发布参数

这个操作的目的之一是让用户能够自己选择要输出的数据格式。在 Navigator 方框中找到 Output Format 参数，并发布（如左图）。

3) 重新工作空间

使用 Prompt and Run 选项，重新运行工作空间，这样你就能够自己选择要编写的格式了。

这样，我们就创建了一个工作空间，能够将中转数据转换成任何格式的数据！

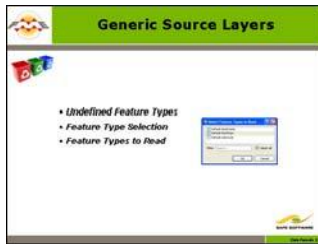


Professor Spatial F.M.E., E.T.L. 说过...

“有趣的是，即使最开始的通用编写器格式是 Shape，Workbench 也不会包含一组 GeometryFilter 函数。

这是因为，我们打算让通用编写器来对几何形分型分类，所以工作空间就没必要是一个通用工作空间了。”

通用源层



“Feature Types to Read” 指的是一个 Reader 参数，使用它就能对工作空间要素类进行高级控制。它能够简化复杂的工作空间。

为了动态地读取一系列用户自定义的层（但是提前并不知道），我们就需要首先处理未定义的要素类，并且控制已有的要素类。

未定义的要素类

工作空间的一个功能就是，添加所有的源数据和要素类到一个单一的工作空间。

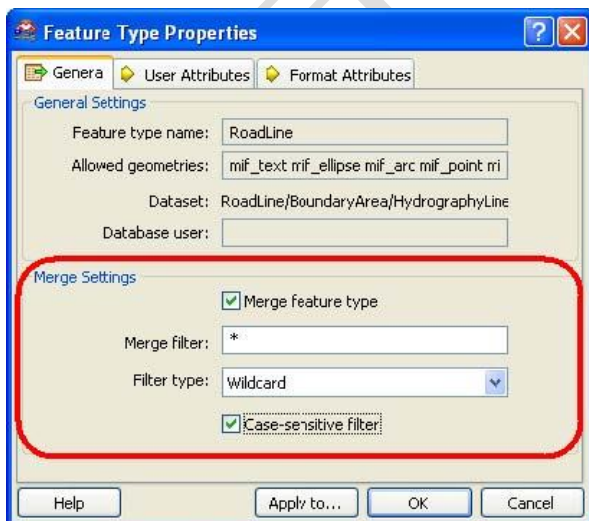
但是它的确就是，并不能动态地处理：它不能够处理那些事先没有定义的要素类。

第五单元（数据集和要素类）中，我们学到了一种方法来处理这个问题：使用 Merge Feature Type。



Merge Feature Type

记住，Merge Feature Type Filter 定义了哪些要素类能够通过，所以，它也能够创建一个通用工作空间。使用 Merge filter 设置，我们就能够读取任何一个源类，而不必理会是否在工作空间中提前定义了它们。



左图：来自单元 5 中的例子。用户在‘Merge feature type’一栏中打钩，将它作为一个通配符。

要素类选择

即使我们创建了一个工作空间，并且已经定义了所有的源数据和要素类，但是，通常用户并不想读取所有的数据和要素类，他们能够自己选择要读取的。

例如（如右图），这个工作空间含有大量的要素类(BusRoutes, BusStops and metrorail)，但是工作空间的创建者希望限制要读取的内容，这样终端用户就能够自己选择了。



断开要素类

最常见的方法就是，在运行工作空间之前，断开连续的要素类(例如，BusRoutes)，或者删除它们(like metrorail)

但是，很显然地，用户就需要编辑工作空间，并且也不能够在运行时选择关闭或打开要素类。

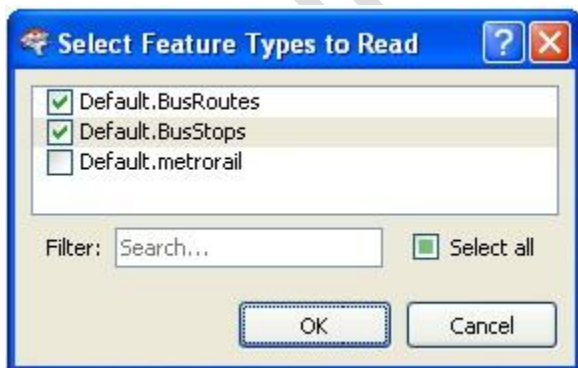
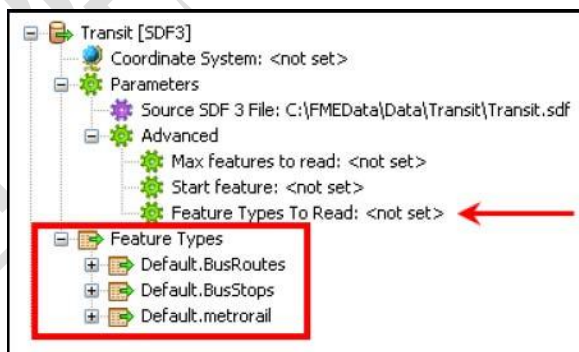
要读取的要素类

“Feature Types to Read” 参数解决了动态地选择要素类的问题（例如，不必要编辑工作空间）使用 Reader 参数就能够选择要通过的已定义要素类。

你也可以将它理解成为过滤掉不需要的数据：Merger Filter 一个更加精确的版本。

右图：在这个编写器下列出了三个不同的要素类“Feature Types to Read”参数在最上面。

现在它的值是<not set>，这表示，用户并没有使用它，会从源数据集中读取所有的要素。



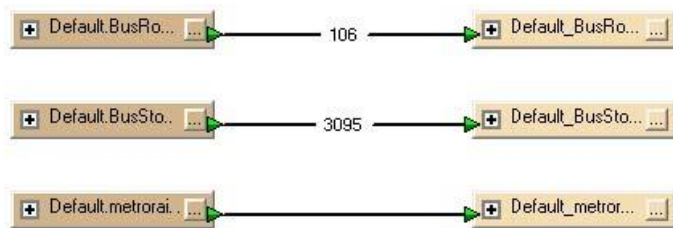
跟所有的参数一样，这个参数也可以双击进行编辑。

左图：用户正在编辑参数，并将它设置成只能读取 Rail, Rivers 和 Roads。

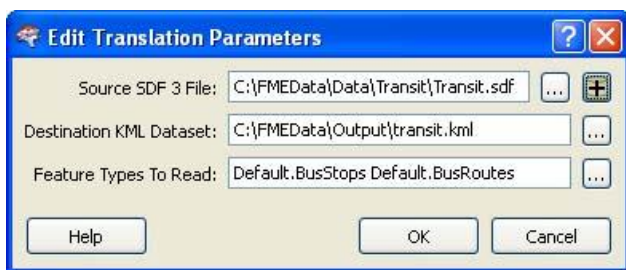
下图：Read 参数的要素类就如 Navigator 面板上的样式。

 Feature Types To Read: Default.BusStops Default.BusRoutes

右图：在运行工作空间后，要素计算显示只有已选要素类的要素才能够通过这个工作空间。



同样地，因为参数是可发布的，这样就能够创建一个通用工作空间，从这个空间中用户能够选择要读取的要素类。



左图：通过发布 Feature Types To Read 参数，按照提示用户就可以选择它了。

右图：...这就是说，也可以在指令条中设置它。

```
Windows command-line to run this workspace:
fme.exe wb-xlate-1231281562712_8468
--SourceDataset_SDF3 C:\FMEData\Data\Transit\Transit.sdf
--DestDataset_OGCKML C:\FMEData\Output\transit.kml
--FEATURE_TYPES "Default.BusStops Default.BusRoutes"
```

限定

这是一项高级操作，但是 Feature Types to Read 仅仅列出了工作空间中被定义了的要素类，就不能够重新读取源数据集，读取那些未被定义的要素类，并且即使设置 merger filter，也不能读取那些未定义的。

换句话说，如果我有一个数据库阅读器，我能够选择的表格就只是那些在工作空间中已经定义了的要素类。FME 不会检查数据库，给我一份包含所有表格的列表。



在 FME 将来的版本中会解决这个问题，并将它作为通用阅读器的一部分。

另一个会改善的功能就是，创建要素类组，让用户自己选择要素类组，而不是单个的要素类。

例如，在上面的工作空间中，就可以将 Rails 和 Roads 合并为一个叫做 Transportation 的组，Buildings 和 Schools 合并为一个叫做 Structures 的组。

Example 2-Feature Types to Read

这个例子会展示如何使用 Feature Types to Read 参数。

目的: 我们希望扩展例 1 中的工作空间，这样用户就能够选择要转换的源要素类

具体步骤

1) 开始转换

如果有需要，就启动 Workbench，打开例 1 中的工作空间。

2) 发布 Feature Types to Read

在 Navigator 方框中，找到 Feature Types to Read 参数，然后发布它。

3) 开始转换

保存工作空间（之后我们会再次用到它），使用 File > Prompt and Run，运行它。限定工作空间读取 BusRoutes 和 BusStops。

检查输出（以及 Workbench 日志窗口），确保输出结果是正确的。



```
=====
Features Read Summary
=====
Default.BusRoutes                      106
Default.BusStops                      3095
=====
Total Features Read                    3201
=====
```

通用目标层



扇出是 FME 最强大的功能，使用它就能够很简单地进行一些复杂的操作。

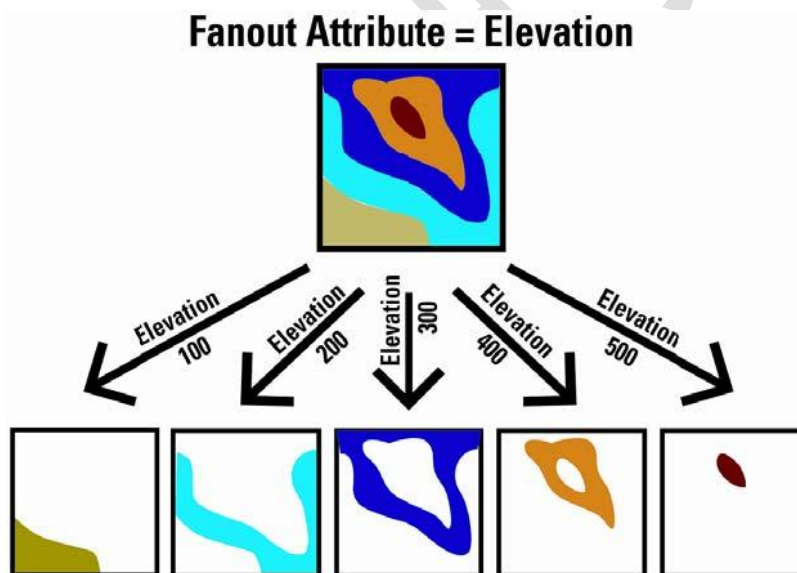
在工作空间目标方面，你可以将要素类处理理解成更加“动态”的，而不是“通用”的。在一些用户自定义输入的基础上，就可以编写临时的要素类。

Data Fanouts

不需要首先在目标模式中添加输出数据，使用扇出就可以将数据分裂成多个输出。我们将它理解成“通用”，这时因为分类数据是很简单的，并不需要编辑或更新工作空间。

通常，FME 会自动将多个源数据集合并成一个输出数据集，总的来说，通过添加多个编写器，以及处理源要素，就能够创建多个数据集了。

使用一个自定义属性值，将数据分配到另外的要素类或数据集，一个扇出就可以自动创建多个输出了。



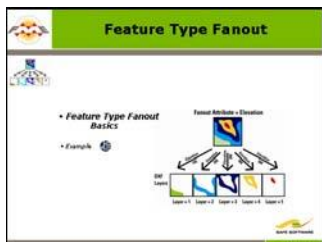
左图：这里，根据每个要素的值，就可以将一个 Elevation 数据的源数据集被分类成多个输出。

听起来很熟悉，是吗？这与运用到源数据的“merge filter”很相似，事实上，一些用户都会参考 merge filter 来“Fan-in”。

扇出的一个主要优点就是它有高度的灵活性—不依赖固定的模式。

有两类扇出：要素类扇出和数据集扇出。

要素类型扇出

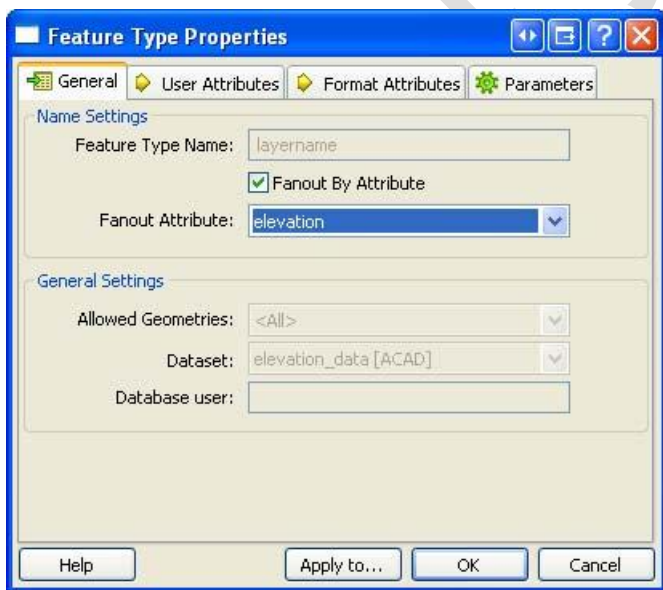
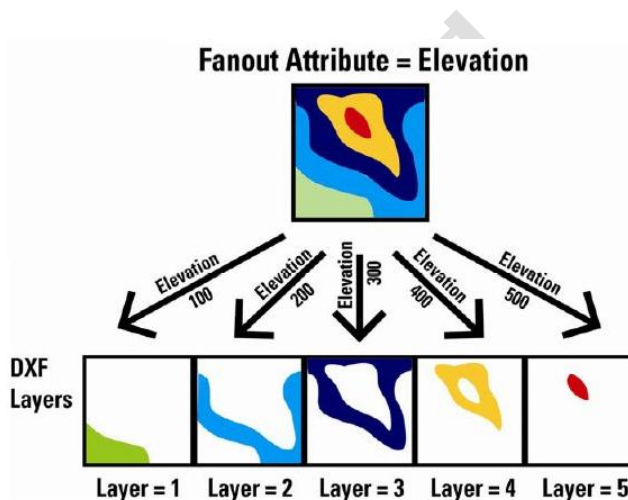


使用它就能在一个单一数据集中，将数据分裂成多层。

Feature Type Fanout Basics

不需要创建大量的输出数据集，使用它就能在单一目标数据集中将数据分裂成多个要素类。

右图：在立体图属性的基础上，将数据划分成多个部分，然后在 DXF 文件中，将每个部分编写到不同的要素类中。



使用目标要素类的 properties 对话框，定义要素类扇出。

左图：设置一个要素类扇出，就会出现上面的图标。将数据编写到 AutoCAD dxf 文件，使用“Elevation”属性来进行扇出。

为每个“elevation”属性值创建一个新的要素类（在 DXF 文件中），然后将含有属性值的要素编写到这个要素类中。



Example 3-要素类扇出

在这个例子中，将一个要素类扇出应用到 Interopolis 城市的数据集。

目的

工程部有一组 AutoCAD 文件来储存有关水管分布的信息。你要做的就是，创建一个 GML 数据集，在一个单独的类别中储存每个管道的直径。

具体步骤

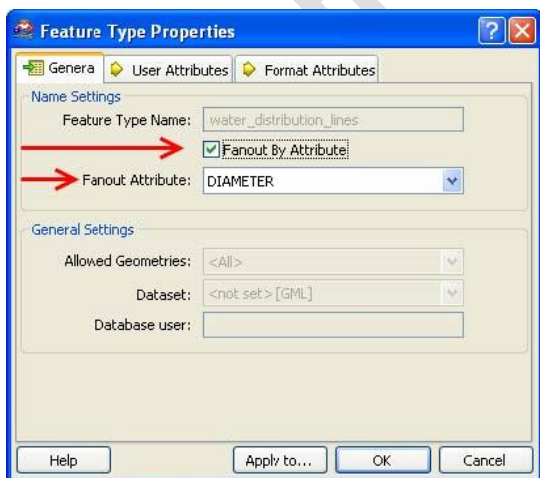
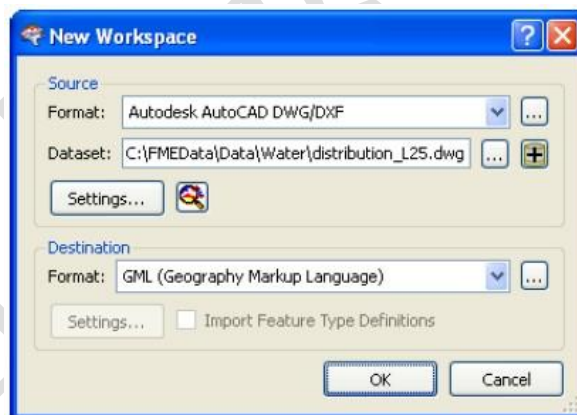
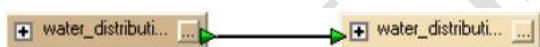
1) 开始转换

启动 Workbench 将 AutoCAD 转换为 GML

Source Format	Autodesk AutoCAD DWG/DXF
Source Dataset	C:\FMEData\Data\Water\distribution_L25.dwg
Destination Format	GML (Geography Markup Language)

在接受新工作空间之前，单击 **setting** 按钮并确信 AutoCAD 源要素类型是由 AutoCAD 属性结构分类的。

在工作空间中，我们需要的唯一的数据源要素类型是 **water_distribution_lines**



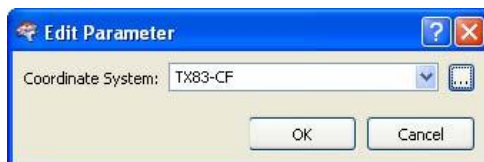
2) 创建扇出

打开 **water_distribution_lines** 要素类的 **feature type properties** 对话框，在 **Fanout by attribute** 一栏打钩，然后选择 **DIAMETER** 作为扇出属性。

3) 设置一个坐标系

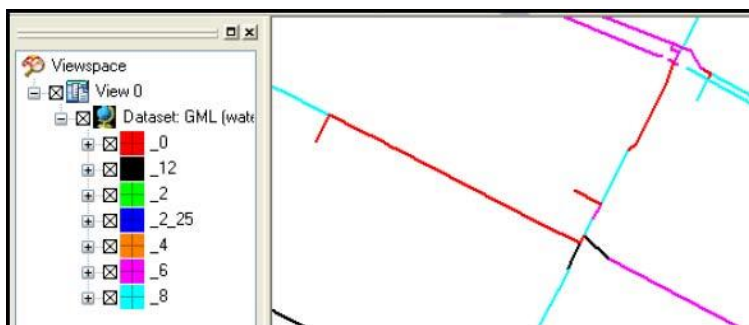
即使没有一个已知的坐标系，FME 也能够创建 GLM 数据。因为 FME 不能从源 AutoCAD 中读取一个坐标系，所有就必须自己定义坐标系。

将目标坐标系设置成 TX83-CF，使用导航方框中的坐标设置，或者 `CoordinateSystemSetter` 函数。



4) 开始转换

创建一个目标 GML 文件，来编写和开始转换，这样就能创建一个单一的 GML 文件。



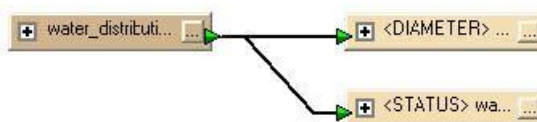
在 FME Universal Viewer 中打开输出 GML 文件。观察怎样为含有不同直径的管道创建不同的要素类。（如左图）

5) 创建第二个扇出

现在我们来完成第二个操作—创建第二个要素类。

使用右键> 复制就能复制已有的目标要素类 将这个新要素类命名为 `water_distribution_status`（但是，首先你需要关闭扇出）。

选择 `STATUS` 作为扇出属性，将源要素类连接到这个新的要素类，实际上，你就是在复制数据。

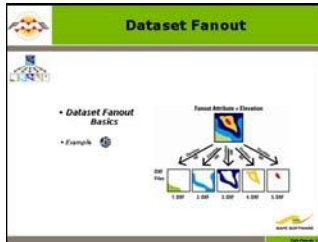


6) 保存，重新运行工作空间

保存这个工作空间！因为待会你还要用到它。

重新开始转换。使用 FME Universal Viewer 刷新输出数据集。观察是怎样创建第二组要素类的（与管道形态有关）。使用不同的属性就能将要素类设置成扇出。

数据集扇出

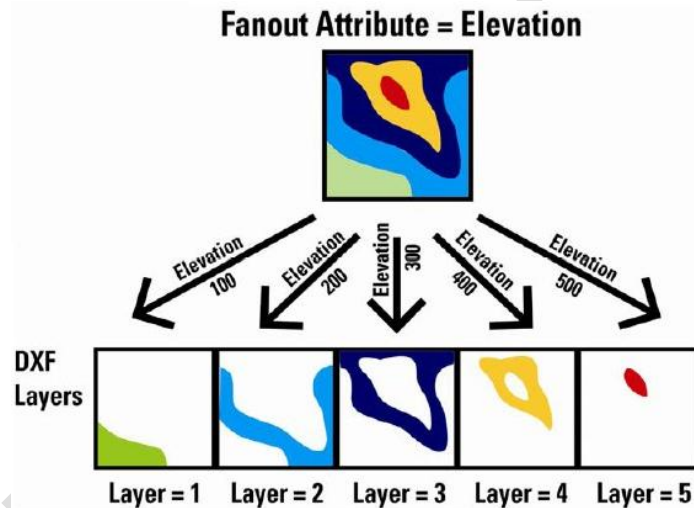


它能够在不同的数据集中将数据分裂成多个要素类

Dataset Fanout Basics

在不同的数据集中，他将相同的要素类划分在一起。

右图：在 Elevation 属性的基础上，将数据划分为多个部分，每部分就会编写到 DXF 文件中的同一个要素类。



找到导航窗口中的高级设置，就可以定义一个数据集扇出了。双击‘Fanout Dataset’设置，设置成 YES，也可以设置其它的参数。

左图：一个数据集的扇出设置显示了上面提到的图标。通过扇出“elevation”属性，就能够将数据编写到一组 AutoCAD 文件中。

也可以为每个“elevation”值，以及含有这个值的要素（已经被编写到了数据集中），创建一个新的数据集了。



Example 4-数据集扇出

这个简单的例子会将一个数据集扇出应用到 Interopolis 城市的数据集中。

目的

策划部使用一个 GML 数据集来保存 Interopolis 城市的区域划分信息，你要做的就是为每个区域创建一组 AutoCAD DWG 文件。

具体步骤

1) 创建转换

启动 Workbench，然后将 GML 转换为 AutoCAD。

Source Format	GML (Geography Markup Language)
Source Dataset	<u>C:\FMEData\Data\Zones\zoning.gml</u>
Destination Format	Autodesk AutoCAD DWG/DXF



左图：源数据集里唯一的要素类就叫做 zoning。

2) 开始转换

设置一个目标 DWG 文件，来编写，进行转换。注意，只能创建一个单一的 DWG 文件。

3) 设置一个数据集扇出

在 Workbench 的导航方框中，找到输出数据集。右击这个数据集，选择 Fanout Dataset，就会打开一个扇出参数对话框。

将 Fanout Dataset 设置成 Yes，就会激活其它的扇出参数字域。

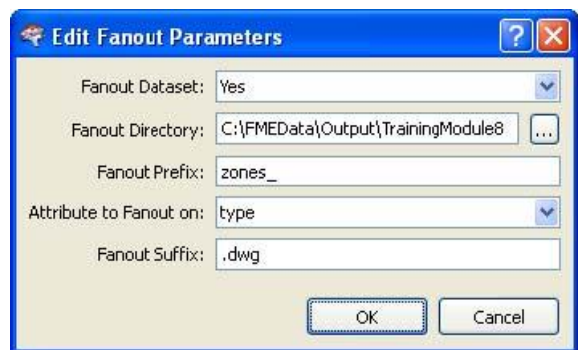
4) 设置扇出参数

如果还没有设置扇出目录，就可以点击浏览键，然后进行设置。目录必须是 "...C:\FMEData\Output\TrainingModule8".

将 'type' 设置成 'Attribute to Fanout on'

Fanout Suffix 的默认值一定要是：.dwg

右图：需要的扇出参数



Mr CAD 说过...

“使用前缀就能够在每个输出数据集名的前面添加一串描述文字。是否定义了目标数据集会影响到前缀的第一个值。”

5) 重新运行工作空间

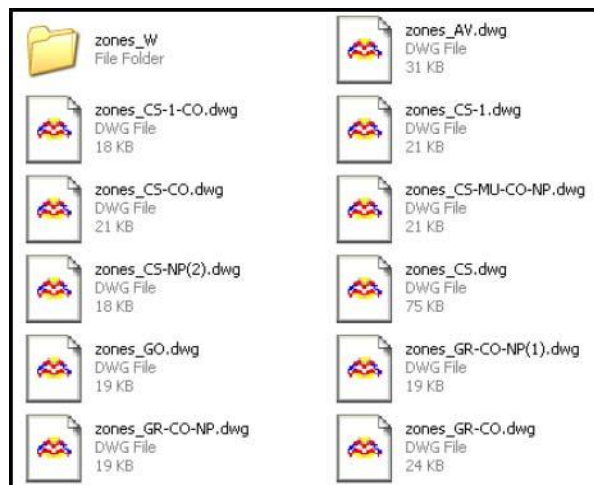
重新开始转换。现在已经创建了大量的数据集（DWG）文件，有趣的是，你会发现在日志窗口中每个数据集都有一组单独的数字，它们表示有多少要素存在于这个特殊的区域。

右图：Windows Explorer 中的一个输出数据集样本。

每个文件代表一个区域名，例如，在文件 CS-MU-CO.dwg 中的 CS-MU-CO 结果。

我们要注意一个小问题：标有“W”的文件夹是由区域名种的“/”符号产生的。

事实上，区域名是 w/LO,它导致出现越来越多的 W 文件夹,包含一个叫做 LO 的数据集。



6) 检查输出

在 FME Viewer 中检查一个输出文件。一共有多少个要素类？这些要素都是我们需要的吗？

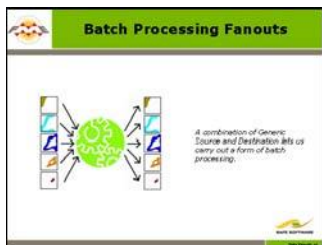
7) 额外的任务

使用一个文件夹形式的目标格式，例如，ESRI Shape 或 MapInfo MID/MIF，再次进行这个操作。输出是怎样与 DWG 输出不同的？



也能够对已发布参数进行数据集扇出前缀和后缀设置，这样用户就能够在运行时，或者通过一个指令来定义它们，但是，你不能够发布扇出属性。

批量处理扇出



将 Generic Source 和 Destination 结合使用就能够模拟一个批量处理

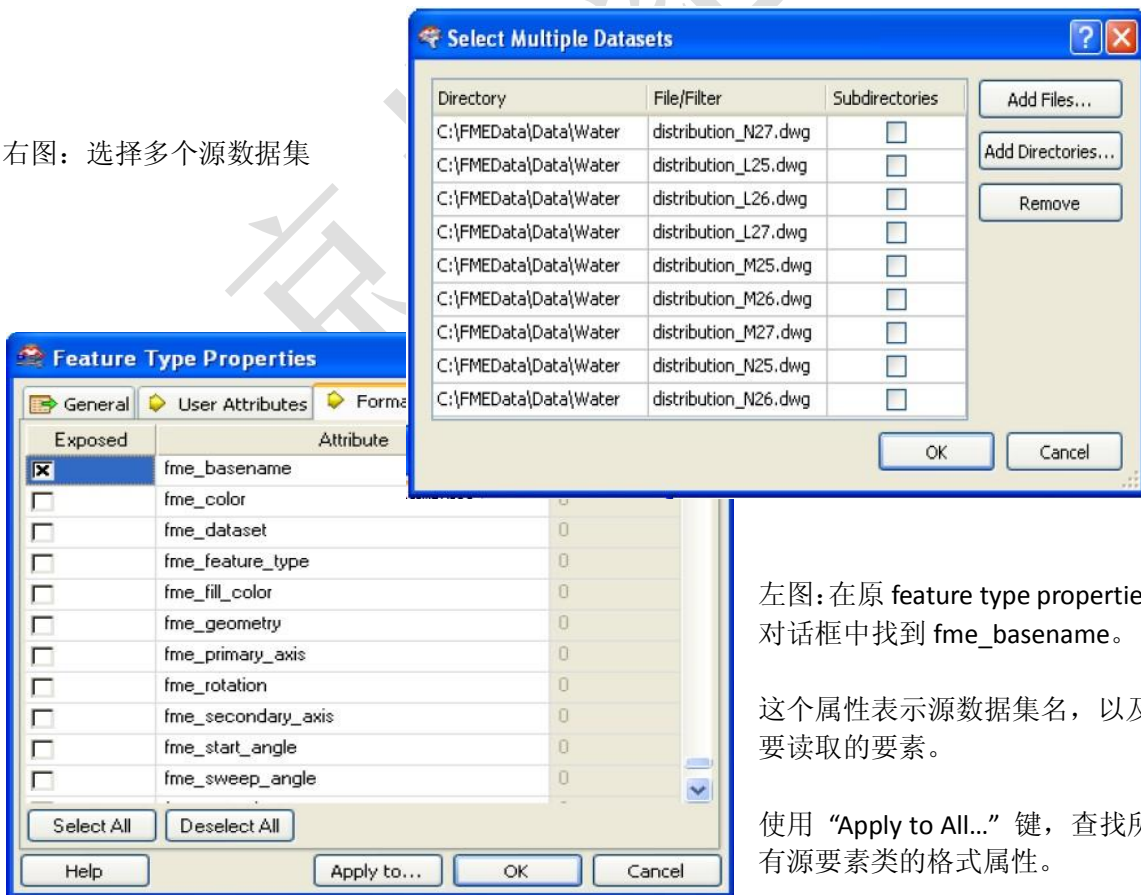
Methodology

批量处理的基本定义就是：读取多个数据集，并且编写同等数量的目标数据集。将 merge filter 和 dataset fanout 结合使用就能够模拟批量处理：简单地读取工作空间中的一批源数据集，处理它们，然后将他们恢复成原来的结构。

我们可以将它描述成模拟批量，事实上，这与 FME 定义的批量处理是不同的，但是能够产生相同的结果，所有有时可以取代真正的批量处理。

这个功能的关键就是要能够判断哪些要素来自于哪些源要素集，以及哪些扇出进行了判断。很幸运，FME 就有一个叫做 fme_basename 的格式属性，它包含了这个信息。

右图：选择多个源数据集



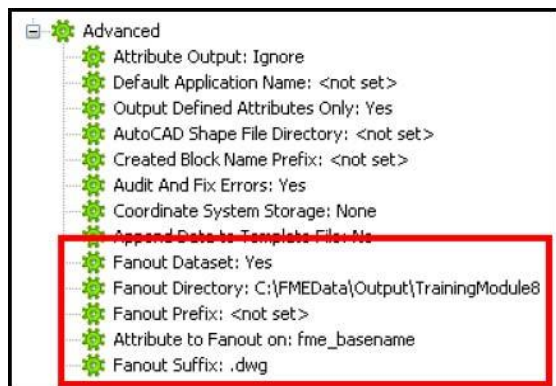
左图：在原 feature type properties 对话框中找到 fme_basename。

这个属性表示源数据集名，以及要读取的要素。

使用 “Apply to All...” 键，查找所有源要素类的格式属性。

右图：使用 `fme_basename` 将一个数据集扇出设置成要扇出的属性。

Name	Size	Type
distribution_L25.dwg	106 KB	DWG File
distribution_L26.dwg	245 KB	DWG File
distribution_L27.dwg	163 KB	DWG File
distribution_M25.dwg	207 KB	DWG File
distribution_N27.dwg	96 KB	DWG File
distribution_M26.dwg	230 KB	DWG File
distribution_M27.dwg	148 KB	DWG File
distribution_N25.dwg	85 KB	DWG File
distribution_N26.dwg	91 KB	DWG File



左图：运行工作空间，为每个输入创建一个输出数据集（这个例子中是 GML），也就是说—批量处理。

Drawbacks

许多 FME 用户会觉得这种方法比起真正的批量处理来说要更加灵活，更加简单，但是，它会产生大量的问题。

系统资源

在真正的批量处理中，FME 会分别对每个源数据集进行处理，也就是说，FME 一次只能用充足的系统资源来读取，处理一个单一的数据集。

而是用模拟批量处理，就能够同时处理所有的数据；所有，也会增加已有系统的负担。比起使用 `File > Batch Deploy`，这种转换会占据更多的空间，并且运行起来会比较慢。

Processing against each other

并且，在真正的批量处理中，FME 会分别读取，处理每个源数据集，但是，使用模拟批量处理，就会一起读取，处理所有的源数据，所有，用户一定要知道：组形式的转换并不能处理更大的组。

例如，在真正的批量处理中，最接近的 `neighbour` 处理就只会找出同一个数据集中的 `neighbour`，而是用模拟批量处理，就只找出任何一个源数据集中的 `neighbours`，这是因为同时读取了所有的源数据。

许多基于组的函数也有一个成组设置，因此通过将函数设置成成组的 `fme_basename` 属性，就能够避免这个问题了；但是，这又会影响运行速度。



通用属性



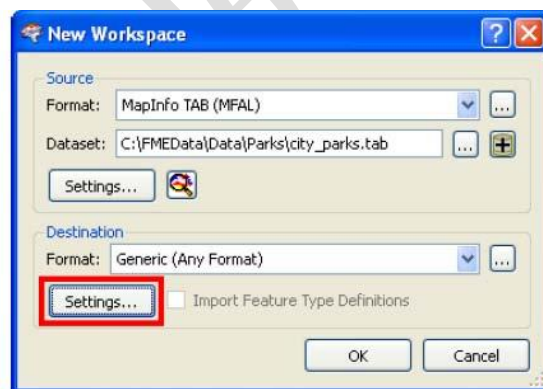
含有动态模式的编写器就能够支持任何格式的源属性，并且输出它们。

前面已经提到了，通用编写器的最大弊端就是，它需要设置一个预先定义的模式，动态模式就能够解决这个问题。

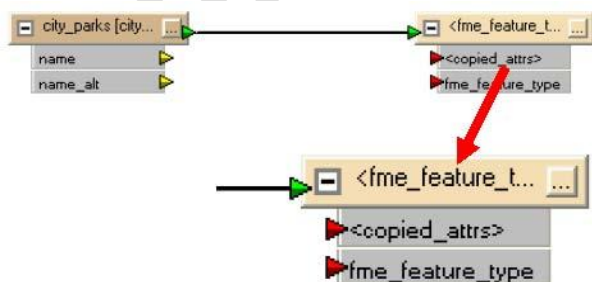
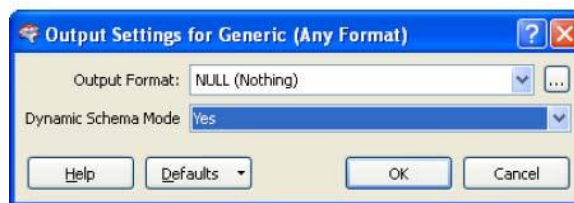
动态模式

它指的是，创建一个目标模式来自动支持任何发送给它的源属性。

右图：创建了一个带有通用编写器的工作空间。在使用动态模式之前，一定要按 **Settings** 按钮。



...在之后的对话框中，将 **Dynamic Schema Mode** 设置成 **Yes**



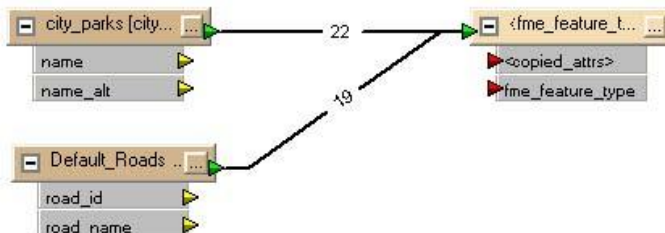
这样就会出现左图中的工作空间（如左图）。

注意目标模式属性包括 **<copied_attrs>** 和 **fme_feature_type** – 后者是必须的，因为动态模式要求设置一个扇出。

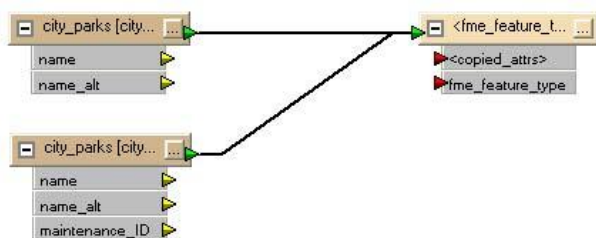
实例场景

虽然常用设置指的是一个单一阅读器，使用它就能读取相同格式的不同数据集，但是会有许多不同的情况，可能会同时用到 **Dynamic Schema** 和 **Generic Writer**。

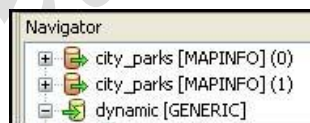
右图：一旦创建了一个动态模式，就能够将其它的阅读器（或数据集）附加到同一个通用编写器中——即使他们含有不同的模式。



当将同一种格式的多个数据集（但是含有不同的模式）发送到同一个通用编写器中，就会将输出划分成多个单一要素类，每个要素类都含有不同的属性（例如，上面的工作空间就会同时输出 **city_parks** 和 **Default_Roads** 要素类）。



左图：如果两个数据集拥有相同的要素类名，但是不同的属性，在一个动态模式中使用这两个数据集，然后从 **Navigator** 方框中最上面的 **Reader** 中获取输出属性。



Example 5-通用编写器和动态模式

开始，这个例子就说明了为什么要使用动态模式。

具体步骤

1) 开始转换

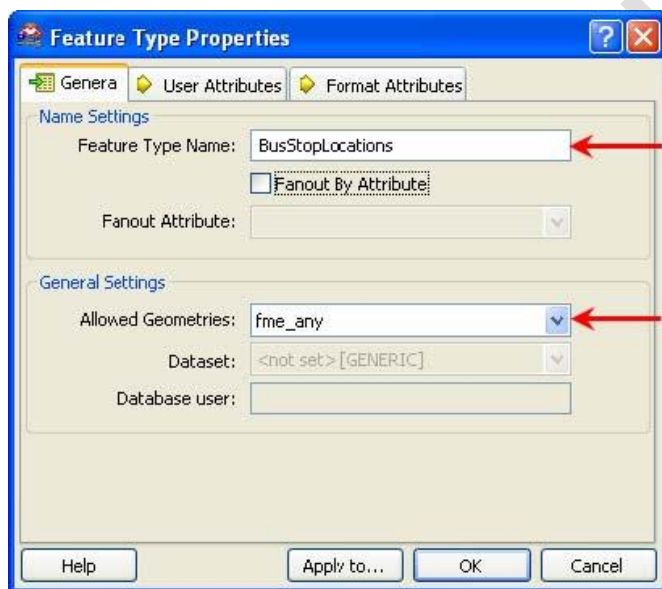
如果有需要就打开 Workbench，然后打开例 2 中的工作空间。

2) 添加源数据集

现在，我们需要将马路添加到 Bus stop 要素类，方便查询 Bus stop 数据。使用 Source Data > Add Dataset，就能够添加一个新的阅读器和数据集。

Source Format	MapInfo MIF/MID
Source Dataset	<u>C:\FMEData\Data\Roads\RoadLine.mif</u>

将源 RoadLine 连接到输出 Feature Type Default_BusStops，将目标要素类重新命名为 BusStopLocations 将 Allowed Geometries 设置成 fme_any。



3) 运行工作空间

运行工作空间，并检查输出。

注意，Bus Stop 要素含有属性，而 Road 则没有！

这就是 Generic Writer:最大的弊端：不能够编写未定义的属性。

4) 创建一个工作空间

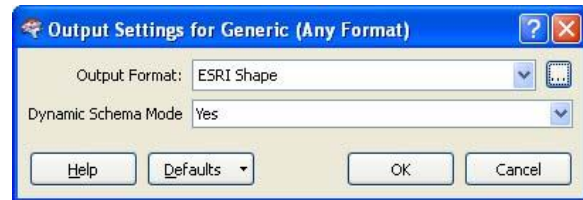
让我们使用 Dynamic Schema，来试着解决这个问题。

启动 Workbench，将 Autodesk SDF 转换为 Generic (Any Format)。

Source Format	Autodesk MapGuide Enterprise SDF
Source Dataset	<u>C:\FMEData\Data\Transit\Transit.sdf</u>
Destination Format	Generic (Any Format)

在目标设置中，你需要进行以下设置：

Dynamic Schema	Yes
Output Format	ESRI Shape



注意，这次只有一个单一的输出要素类，而不是三个。

5) 添加源数据集

再次使用 Source Data > Add Dataset，添加一个新的阅读器和数据集。

Source Format	MapInfo MIF/MID
Source Dataset	<u>C:\FMEData\Data\Roads\RoadLine.mif</u>

将 RoadLine 连接到目标数据集中。

6) 运行工作空间

发布 Feature Types to Read 参数。

使用 Prompt 和 Run，运行工作空间。

在提示对话框中，使用发布的参数 Feature Types to Read，限定 SDF 只读取 BusStop Feature Type. 检查输出。

在输出中有两种要素类：一个是马路，而另一个是公交车站，这两个要素类都有它们自己的属性设置。

在进行了以上操作后，现在我们有了一个工作空间，使用它就能够能够在任何模型中读取任何数据，将数据编写成任何格式。因为有两种不同的源格式，所有我们需要添加两个阅读器—一个通用阅读器能够解决这个问题。

通用工作空间概述



我们可以以任何连接形式使用通用工作空间功能来满足用户需求

表格综述

有大量针对通用工作空间的连接，下面的表格显示了各自的要求以及相应的优点。

输出格式	结构（属性）	
	固定	动态
固定	一个传统的工作空间，用来转换已知的格式和模式。编写器是特定格式，预先定义了所有的属性和要素类（Reader 和 Writer）	一个固定格式，但是动态的属性。使用工作空间，例如，加载不同的数据到一个数据库中。使用动态属性，就一定要用到通用编写器，但是不能发布输出格式，否则就会引起改变
	Generic Writer × Dynamic Attributes ×	Generic Writer ✓ Dynamic Attributes ✓
	Pros: 传统模式易于理解 Cons: 缺乏灵活性	Pros: 不受源属性变化的影响，易于维护 Cons: 仍然要求一个通用编写器，在转换过程中，不容易调整数据模型
生成	使用一个固定模式的工作空间，就能够动态地选择输出模式。预先定义所有的要素类，并且发布输出格式	针对不同类型用户的一个工作空间，它既是灵活的也是动态的。不需要预先定义目标要素类，并且发布输出格式
	Generic Writer ✓ Dynamic Attributes ×	Generic Writer ✓ Dynamic Attributes ✓
	Pros: 灵活，且容易排列 Cons: 源属性发生改变，则需要编辑工作空间，或更新工作空间	Pros: 不受源属性变化的影响，易于维护 Cons: 在转换过程中，不容易调整数据模型

使用需知

将 Fixed, Dynamic 以及 Generic 连接起来，使用这些连接，用户就一定要了解一些情况：

Output Format	Schema	Use Cases
Fixed	Fixed	传统的FME实例， -事先知道源数据集和目标数据集 并且 -事先知道源数据集和目标数据集的模式
Generic	Fixed	从一个中心数据库或一组文件中分配数据，这时，输出格式： -一定要是用户自定义的并且模式很少会改变 或者 -目标模式一定要与源模式不同
Fixed	Dynamic	与Merge Feature Type联合使用，就能够上传或输入到一个预先定义格式的目标数据集中。这种情况下： -源数据集要含有大量不同的模式，并且需要大量的工作空间来进行转换； 或者 -元数据集含有大量不同的模式，但是需要进行同一类型的转换（例如，重新投射，或剪辑）
Generic	Dynamic	数据分配，转换或转变（例如，重新投射，裁剪，属性处理），这种情况下输出格式一定要是自定义的： -模式容量要很大（含有许多要素类） 或者 -应该改变源模式 或者 -在创建时，并不知道模式

单元复习



这个单元帮助你创建多个数据集转换，在已有工作空间中添加新的数据集和要素类

你应该从这单元中学到什么

以下是你要学到的主要内容：

理论

- 常用工作空间能够很简单地进行复杂的操作
- 常用编写器支持任何格式的数据
- 一个动态模式可以处理含有任何源属性的数据
- 数据扇出指的是一种方法，快速地将数据分裂成多个不同的输出

FME 功能

- 能够使用通用编写器
- 能够使用动态模式
- 能够使用 **Feature Types to Read** 参数
- 能够同时处理数据集和要素类扇出
- 能够使用数据及扇出来处理多个数据集，并且能够判定在什么情况下适合使用这个方法