



XML Instance Loading

Version 1.2

May 13, 2008

PathMATE Technical Notes

Pathfinder Solutions LLC
33 Commercial Drive, Suite 2
Foxboro, MA 02035 USA
www.PathfinderMDA.com
888-662-7284

Table Of Contents

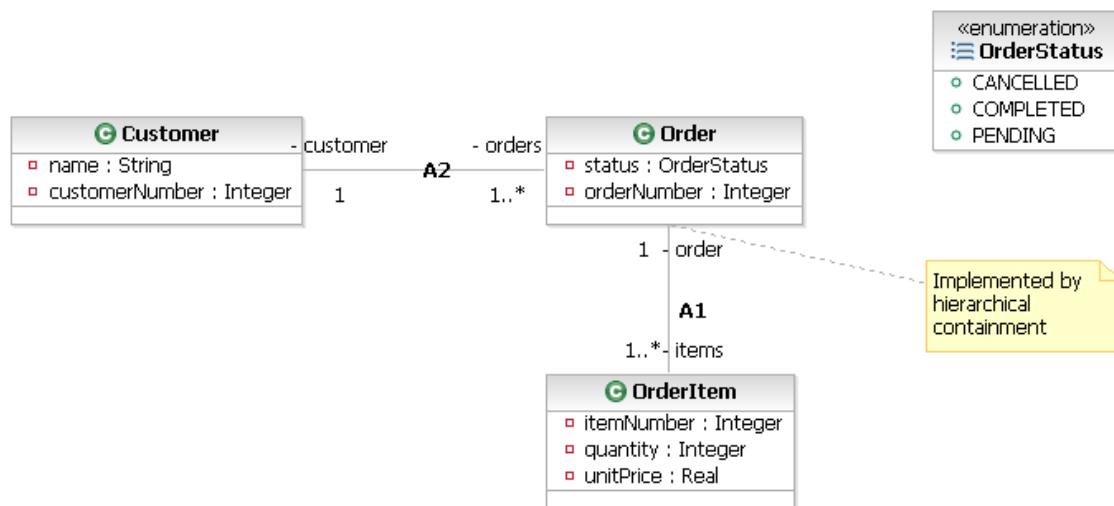
- 1. Introduction 1**
 - Example.....1
 - Scope2
- 2. Using the XML Instance Loading 3**
 - Introduction3
 - Marking the model3
 - Model Transformation5
 - Compiling and Linking.....5
 - Creating an instance population XML file5
 - Invoking the instance loader.....6
 - Example.....7

1. Introduction

This technote describes the enhancements to the Java design to support the runtime loading of an instance population from an XML document. The instance population allows object instances and associations linking the instances to be loaded at runtime.

Example

For example, given the fairly simple and straightforward class diagram:



It should be possible to load instances of these classes at runtime using an XML document that looks like:

```

<PreexistingInstances>
  <OrderProcessing_Customer>
    <customerNumber>1111</customerNumber>
    <name>Fred</name>
    <_x0040_id>1</_x0040_id>
  </OrderProcessing_Customer>
  <OrderProcessing_Order>
    <orderNumber>1111</orderNumber>
    <status>PENDING</status>
    <_x0040_id>2</_x0040_id>
  </OrderProcessing_Order>
</PreexistingInstances>
  
```

```
<A1_to_items>
  <OrderProcessing_OrderItem>
    <itemNumber>9876</itemNumber>
    <quantity>2</quantity>
    <unitPrice>9.99</unitPrice>
    <_x0040_id>5</_x0040_id>
  </OrderProcessing_OrderItem>
</A1_to_items>
</OrderProcessing_Order>
<OrderProcessing_Relation_A2>
  <Order_id>2</Order_id>
  <Customer_id>1</Customer_id>
</OrderProcessing_Relation_A2>
</PreexistingInstances>
```

Scope

The enhancements consist of both templates and mechanism support. The templates generate:

- a model-specific XML Schema Definition (XSD) file that describes the structure of the instance population data
- a model-specific Java implementation of the XML loader support classes for each modeled class and association that can be expressed in the instance population.

Supporting Java mechanisms (model-independent) are provided that use the Java SAX parser to drive the parsing and loading of the XSD file.

2. Using the XML Instance Loading

Introduction

The XML instance loading support uses markings to generate additional supporting Java classes that know how to map XML elements to class and relationship instances. The resulting instance loader class can then be invoked at runtime to process XML files containing instance populations.

Marking the model

The templates for XML loading are driven by properties applied to the model using the *properties.txt* file in the transformation working directory. The markings controlling the transformation include:

Marking Name	Applies To	Default	Description
XMLConstructorType	System	N/A	This must be set to "SAX_JAVA" to enable the Java SAX-style loader.
XMLSchemaGen	System	False	This must be set to "True" to enable XML instance loading.
XMLSchemaGen	Object	False	This must be set to "True" to enable loading of instances of this object.
XMLSchemaGen	BinaryRel	False	This must be set to "True" to enable loading of instances of this relationship.
ExcludeFromSchema	Attribute	FALSE	Setting this marking to "TRUE" excludes an object's attribute from the schema. The attribute will be assigned its default value when the instance is created.
XMLSchemaParent	Participant	FALSE	This marking is used to mark the parent participant in a parent/child relationship. Setting this value to "TRUE" causes the relationship to be expressed using hierarchical containment in the XML

			document.
CustomXMLMarshaling	Attribute	<unset>	Setting this attribute triggers specialized marshaling of the attribute as described below.
CustomXMLMarshalingLengthAttr	Attribute	<unset>	Used in conjunction with CustomXMLMarshaling to name a class attribute that holds a length field for use by the custom marshaling support.
CustomXMLEncoding	Attribute	<unset>	Used to describe the encoding used to encode the value of a custom marshaled attribute.

The CustomXMLMarshaling attribute is used to provide specialized handling of an attribute when loaded from the XML instance population file. This marking is used in conjunction with the CustomXMLEncoding and CustomXMLMarshalingLengthAttr markings to control the custom marshaling process. The following table describes the custom marshalling options:

CustomXMLMarshaling Marking	Description
IntegerVector	The attribute is marshaled as an array of integers. The CustomXMLMarshalingLengthAttr marking holds the name of the class attribute defining the length of the array. The integer vector is represented as a space-separated list of ASCII integers.
DoubleVector	The attribute is marshaled as an array of doubles. The CustomXMLMarshalingLengthAttr marking holds the name of the class attribute defining the length of the array. The data can be marshaled in one of two ways. If the CustomXMLEncoding marking is set to "ASCII", the value holds a space-separated list of ASCII doubles. Otherwise, the value holds a base-64 encoded binary dump of the vector.
DoubleMatrix	The attribute is marshaled as a square matrix of doubles. The CustomXMLMarshalingLengthAttr

	marking holds the name of the class attribute defining the dimension of sides of the matrix. The data can be marshaled in one of two ways. If the CustomXMLEncoding marking is set to "ASCII", the value holds a space-separate list of ASCII doubles in row-major order. Otherwise, the value holds a base-64 encoded binary dump of the matrix.
--	---

Model Transformation

Specify the appropriate markings in the properties.txt file. Generate the java code using the PathMATE Java Transformation Map. The schema file and any loading code will be automatically generated.

Compiling and Linking

The XML transformation map generates two distinct outputs: a XML Schema Definition that describes the structure of an instance population XML file and a Java source file/header file combination that implements the runtime instance loading using the Java SAX parser.

The generated output files are as follows:

Output File	Description
java/gc/<system_name>_preexisting.xsd	The XML Schema Definition file describing the structure of an XML instance population
java/gc/<system_name>/sys/<system_name>InstanceLoader.java	The java implementation of the type-specific instance loader classes.

The generated java files depend on mechanism files that provide the Java SAX support. These files are in the mechanisms.xml package.

Creating an instance population XML file

The generated XML Schema Definition describes the allowable structure of an XML instance population file. To allow the XML parser to validate the instance

population against the schema as it is parsed, a reference to the XSD should be included in the XML header as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<PreexistingInstances xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="gc\<system_name>_preexisting.xsd">
</PreexistingInstances>
```

The example header uses a relative path from the XML file to the XSD file. This can be replaced by a full path in case the XSD file resides at a different location. Using an XML editor that understands XML Schema Definitions, such as Altova's XML Spy, will greatly assist in creating valid instance population files. If an XSD compatible editor is not available, the Xerces SAX2Print utility can be used to ensure that the resulting XML file conforms to the schema definition. Information on SAX2Print can be found in the Xerces documentation.

Each leaf class in the source model that is marked with the "XMLSchemaGen" marking will have a corresponding element in the schema named with the convention `<domain_prefix>_<class_name>`. These elements are used to define instances of the class within the document. Each element must have child elements that describe the values of each attribute of the class, unless the attribute is marked with the "ExcludeFromSchema" marking. In addition, a special identifier attribute named `_x0040_id` must be supplied with a unique integral value. This attribute identifies the instance within the XML file and is used to implement relationships between object instances.

Relationships between instances can be expressed using relationship elements. Relationship elements are named with the convention `<domain_prefix>_Relation_A<relationship_number>`. The relationship elements contain two child elements (three in the case of association class relationships) that identify the endpoints of the relationship. The child elements refer to the endpoints using the value of the instance's `_x0040_id` attribute.

Alternatively, for relationships that have a participant marked with the "XMLSchemaParent" marking, the hierarchy of the XML document can be used to describe a parent/child relationship between two instances. This results in a more natural way to express hierarchies of objects. If a class participates in a relationships marked as a schema parent, it will allow child instances to be declared as nested elements in the schema. This nesting can extend to an arbitrary depth. Nested child objects are grouped under an XML element describing the relationship and role in order to allow reflexive parent/child relationships. The naming convention used is:

`A<relationship_number>[_to_<role_name>]`. The role name is used if the child end of the relationship specifies a role name, otherwise it is omitted.

Invoking the instance loader

The instance loader takes a well-formed and valid XML instance population and creates object and relationship instances at runtime. The loader is

invoked by creating a new instance of the `<system_name>InstanceLoader` class and invoking the `loadPopulation()` method on it.

The public APIs exposed by the instance loader are as follows:

`<system_name>InstanceLoader(String filename, boolean validate)`

This is the main constructor of the instance loader class. The file name of the XML instance population to load is passed in as the first parameter. XSD validation can be controlled by the validate parameter the constructor – if set to false, no validation of the document will occur at parse time.

`boolean loadPopulation()`

Invokes the parser to load the instances described in the population. Returns true if successful, false if an error occurred during the parse. If false is return, the error description can be retrieved using the `getLastError()` method.

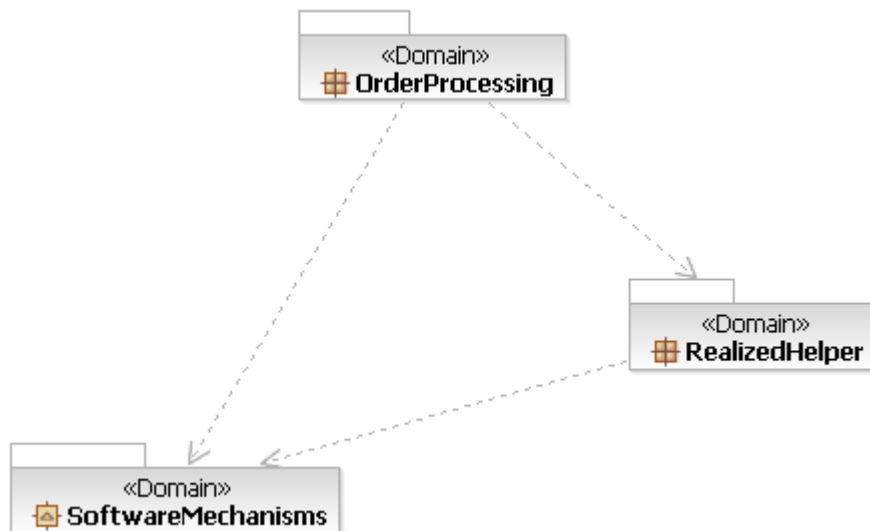
`String getLastError()`

Returns a string containing a description of the last error encountered by the population loader or null if no error was encountered.

Example

To complete the example described in the introduction, the model must be marked to enable XML instance population support and the resulting instance loader must be exposed to the rest of the system.

The domain chart for the full example is:



The model is marked using the *properties.txt* file which contains the following markings:

```
System,InstanceExample,ImplementationLanguage,cpp
System,InstanceExample,XMLConstructorType,SAX_JAVA
System,InstanceExample,XMLSchemaGen,True
Object,InstanceExample.OrderProcessing.*,XMLSchemaGen,True
BinaryRel,InstanceExample.OrderProcessing.*,XMLSchemaGen,True
Participant,InstanceExample.OrderProcessing.A1.Order.order,XMLSchemaParent,TRUE
```

These markings cause all the XML schema to contain all classes and relationships in the OrderProcessing domain. In addition, the A1 relationship between Order and OrderItem marked as a parent/child relationship, allowing items to be expressed in the schema as children of the order.

The instance loader is exposed through a new, realized, domain called RealizedHelper. The domain chart for the realized domain is:



The loadInstancePopulation() method is the most interesting one here (the rest are for printing strings and integers to the console). The java implementation of the loadInstancePopulation service looks like:

```
String RealizedHelper::loadInstancePopulation(String filename, boolean validate)
{
    InstanceExampleInstanceLoader loader =
        new InstanceExampleInstanceLoader (filename, validate);
    if (!loader.loadPopulation())
    {
        return loader.getLastErrorMessage();
    }
    else
    {
        return "";
    }
}
```

The implementation instantiates the generated instance loader class and invokes the loadPopulation() method. If it fails, the error message is returned, otherwise an empty string is returned indicating success.

In the OrderProcessing domain, a new domain service is exposed called loadAndPrint(). It loads an instance population from a file specified in a parameter and prints out the resulting instances. The PAL code that implements this service is:

```
String errorMessage = RealizedHelper:loadInstancePopulation(instancePopulation,
TRUE);
IF (errorMessage != "")
{
    RealizedHelper:PrintString("Error loading instance population: " + errorMessage);
    RealizedHelper:PrintNewline();
    RETURN;
}

FOREACH customer = CLASS Customer
{
    RealizedHelper:PrintString("Customer: " + customer.name + " (");
    RealizedHelper:PrintInteger(customer.customerNumber);
    RealizedHelper:PrintString(")");
    RealizedHelper:PrintNewline();

    FOREACH order = customer->A2->Order
    {
        RealizedHelper:PrintString("        Order ");
        RealizedHelper:PrintInteger(order.orderNumber);
        RealizedHelper:PrintString(" status ");

        IF (order.status == PENDING)
        {
            RealizedHelper:PrintString("Pending");
        }
        ELSE IF (order.status == COMPLETED)
        {
            RealizedHelper:PrintString("Completed");
        }
        ELSE IF (order.status == CANCELLED)
        {
            RealizedHelper:PrintString("Cancelled");
        }

        RealizedHelper:PrintNewline();

        FOREACH item = order->A1->OrderItem
        {
            RealizedHelper:PrintString("                Item ");
            RealizedHelper:PrintInteger(item.itemNumber);
            RealizedHelper:PrintString(" quantity ");
            RealizedHelper:PrintInteger(item.quantity);
            RealizedHelper:PrintNewline();
        }
    }
}
```

Finally, for this example, the system's initialization action was modified to retrieve the first command-line argument and invoke the loadAndPrint() service. The code looks like:

```
String populationFile = SoftwareMechanisms:GetCommandLineArg(0);  
  
IF (populationFile != "")  
{  
    OrderProcessing:loadAndPrint(populationFile);  
}
```

Running the resulting system with the instance population XML file described in the introduction produces the following output:

```
Customer: Fred (1111)  
Order 11111 status Pending  
Item 9876 quantity 2
```