**Updating XML Serialization**

Last Updated: 5/1/2018

Simbionic uses XSLT templates to automatically generate the XML readers and writers used for XML serialization (see project directory ‘xslt’). To update the Simbionic XML representation follow the next 3 steps:

1. **Update the Simbionic model**: classes under com.stottlerhenke.simbionic.common.xmlConverters.model
   1. Handle renaming or other special cases in xslt/renames.xsl as necessary.
2. **Update the Simbionic schema**: file xslt/input/SimbionicJava.xsd
3. **Run xslt/generateAll.bat**: this will generate the needed readers and

Next we describe the first two steps above.

# Update the Simbionic model

Add new classes or update fields to the classes under the package

com.stottlerhenke.simbionic.common.xmlConverters.model.

Provide accessor for fields using a Java beam convention. For example, for a field ‘description’ have accessor’s setDecription and getDescription.

## Notes:

1. Some classes under the model package include XML tags like @XmlAccessorType, @XmlType, @XmlElement. *Those tags are ignored during the XML readers and writers generation*.
   1. TODO: 5/1/2018: Remove all the XML annotations above if not required for other XML processing purposes
2. As it turns out, the name of a field in a Java class and the name of the field in a simbionic schema file sometimes do not match. For example, the class Descriptor has a field ‘descriptors’, but in the schema this field is named ‘descrptorChildren’.
   1. The file **xslt/renames.xsl** provides means to deal with this type of situations (see function DMFn:getDMSetterName)
   2. For backwards compatibility it is not recommended to change the schema to adhere to the Java beam convention. For example, simply ‘fixing’ the Descriptor schema and changing ‘descrptorChildren’ to ‘descriptors’ will break of the Simbionic Test Engine tests.
   3. Boolean fields usually need be handled in the ‘renames.xsl’ functions. For example, the schema for an ActionNode includes the boolean field ‘isFinal’. The code generation expects the fields getIsFinal and setIsFinal. Unfortunately, the class ActionNode does not have the method getIsFinal but it has the method IsFinal. See rename.xsl.
   4. Apart from the 3 notes above, it is in general straightforward to add new fields or classes to the XML schema.
3. Adding new folder models that behave similarly to existing folder models requires manual changes to **xslt/renames.xsl**. Consider the example of adding clases ConstantFolder and ConstantFolderGroup based on the existing models for ActionFolder, BehaviorFolder, and PredicateFolder:
   1. “DMFn:isCompositeCollection” must be changed to include entry “<xsl:when test="$className = 'ConstantFolderGroup'">1</xsl:when>”; this change is needed to autogenerate XML readers and writers that correctly use ConstantFolderGroup model instances instead of attempting to write lists of ConstantFolderGroup collections (when other points in the serialization assume and provide only a single ConstantFolderGroup).
   2. “DMFn:getCompositeCollectionAccessor” must be changed to include entry “<xsl:when test="$className = 'ConstantFolderGroup'">getConstantOrConstantFolder()</xsl:when>”, where “getConstantOrConstantFolder()” is a call to the method that exposes the collection of Constant and Folder objects contained within the ConstantFolderGroup model. This change is needed to autogenerate XML readers and writers that access the members of the ConstantFolderGroupModel.

# Update Simbionic Schema

The file xslt/Simbionic.xsd describes the XML representation of the different classes in the Simbionic model. For a class X there is an XSD complexType whose name is X, representing the schema for the class X (see for example the ‘ActionNode’ schema and compare to the ActionNode class).

## Notes:

1. **Field names**: As noted before, for a field name Y in type X, the generated code expects that the class ‘X’ has the methods getY and setY. If this is not the case, see the functions getDMSetterName and getDMGetterName in the file xslt/**renames.xslt**
2. **Subclasses**: The code generation does not take into account subclasses. If A is a subclass of B, then in the schema definition all fields declared for B need to be declared for A. See for example CompoundActionNode which is a subclass of ActionNode.
3. **Collections**: a collection of object of class X should have a complex type name ‘XGroup’ (yes, add Group after the name of the class). A collection is identified by being a **xsd:sequence** with only on element identifying the XML tag used for the elements in the sequence.

For example, a Poly class has a collection of ‘conditions’. The schema for ‘Poly’ includes the declaration

|  |
| --- |
| <xsd:complexType name="Poly">  <**xsd:all>**  <xsd:element name="indices" type="IndexGroup"/>  <xsd:element name="locals" type="LocalGroup"/>  <xsd:element name="nodes" type="NodeGroup"/>  **<xsd:element name="conditions" type="ConditionGroup"/>**  <xsd:element name="connectors" type="StartGroup"/>  **</xsd:all>**  </xsd:complexType> |

And the declaration of NodeGroup is

|  |
| --- |
| <xsd:complexType name="ConditionGroup">  **<xsd:sequence>**  <xsd:element name="condition" type="Condition" minOccurs="0" maxOccurs="unbounded"/>  **</xsd:sequence>**  </xsd:complexType> |

KEY: The ConditionGroup is declared to be a ‘sequence’ (use **xsd:sequence** for collections, **xsd:all** for regular types).

1. “**Composite Collections**” Additional work is needed to create custom Folders like the ones already used for Actions, Predicates, and Behaviors. The example of ConstantFolder and ConstantFolderGroup from the Simbionic Model section is used here.

Newer \_FolderGroup types that contain elements that might either be folders or elements should copy the structure of existing folders; the choice element is emphasized as the main difference between the new model and the old, no-folder approach using xsd sequences.

|  |
| --- |
| <xsd:complexType name="ConstantFolderGroup">  **<xsd:choice minOccurs="0" maxOccurs="unbounded">**  <xsd:element name="constant" type="Constant" />  <xsd:element name="constantFolder" type="ConstantFolder"/>  **</xsd:choice>**  </xsd:complexType> |

New \_Folder types can also use a xsd structure identical to that of existing \_Folder types

|  |
| --- |
| <xsd:complexType name="ConstantFolder">  <xsd:all>  <xsd:element name="name" type="xsd:string"/>  <xsd:element name="constantChildren" type="ConstantFolderGroup"/>  </xsd:all>  </xsd:complexType> |

Finally, changes made to the SimBionicJava model to accommodate the new folder model should also be reflected.

|  |
| --- |
| <xsd:element name="constants" type="ConstantFolderGroup"/> |