SwingML Renderer

Developer's tutorial

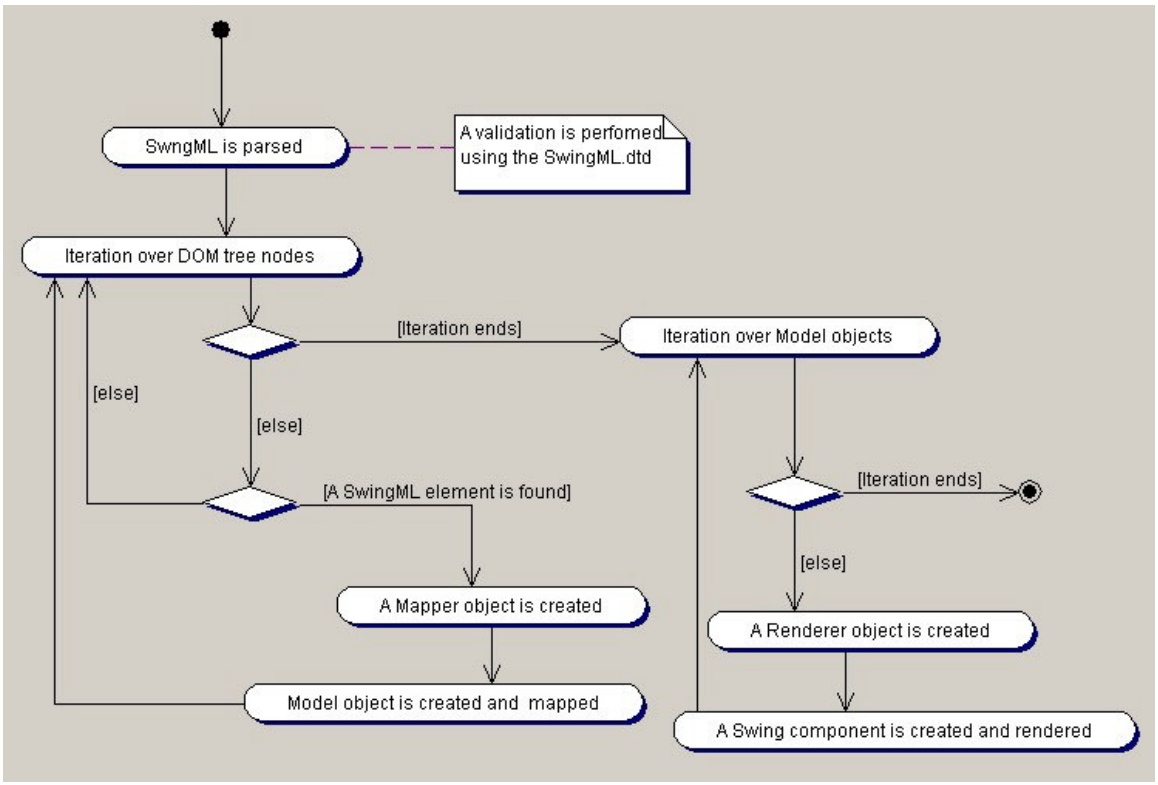
Ezequiel Cuellar

September 2004

SwingML Renderer.

This document explains how the SwingML Renderer works and how it can be extended to add support for new components.
In order to achieve this we will explain step by step how the support for the JButton component was done.

The way the SwingML Renderer works is depicted in the following active diagram.



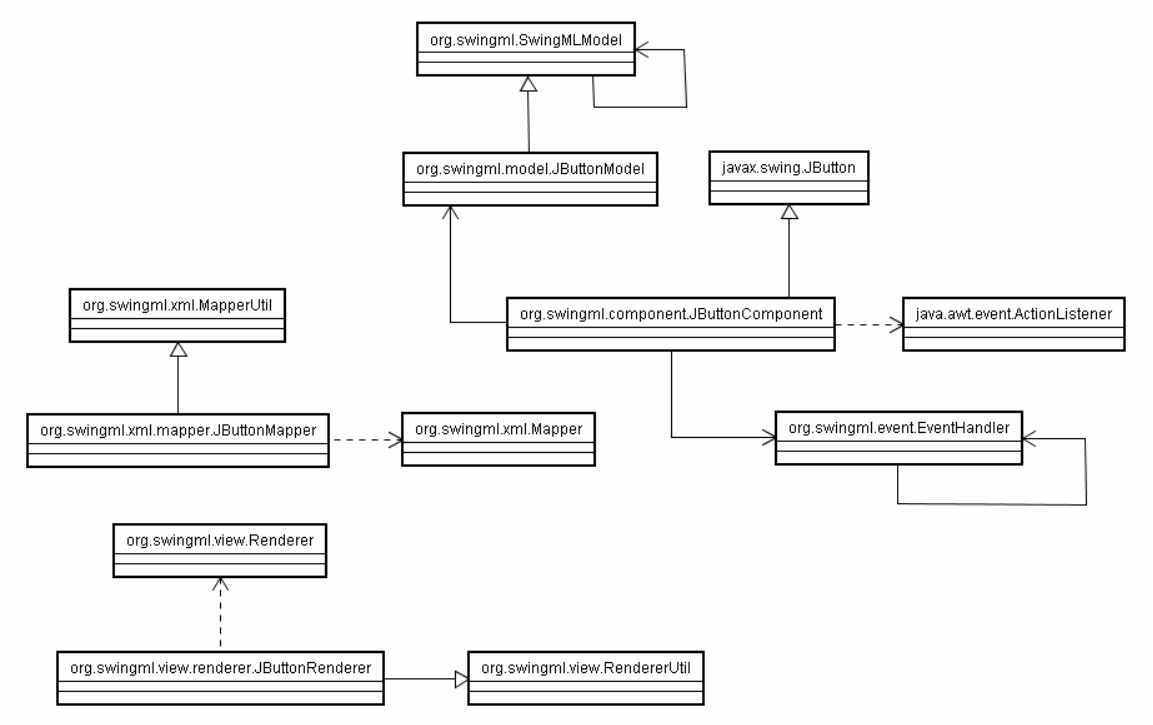
The SwingML is parsed and validated using the SwingML.dtd, once the parsing is done a tree structure representation of the SwingML form is stored in a DOM tree.
The SwingML Renderer iterates on each node of the DOM tree and when a node that complains with a SwingML element name is found an instance of a Mapper is created using a factory pattern. A Mapper is a class that encapsulates the details regarding to the mapping of a DOM node to a Model.
Then Mapper creates an instance of the Model that corresponds to the Swing component and maps the values taken out of the attribute nodes to it. The Mapper is responsible for setting the current Model into its parent. This process continues until all the nodes from the DOM tree have been mapped to a Model objects.
Once the first iteration ends a second iteration starts, this time over the Model objects. The Model objects follow the same tree structure of the DOM tree.
The SwingML Renderer iterates on each Model object and for each one a Renderer is created using a factory pattern. A Renderer is a class that encapsulates the details regarding to the creation and rendering of the Swing component. Then the Renderer creates an instance of the Swing element and sets its Model. The Renderer is responsible of adding the current Swing component to the parent component.

Adding support for JButton components.

To add support for any Swing component the following steps have to be done.

1. Define element specification.
2. Register element specification.
3. Create a Model.
4. Create a Mapper.
5. Register a Mapper.
6. Create a Renderer.
7. Register a Renderer.
8. Create a Component.

This is the class diagram for the JButton support.



Step 1. Define element specification.

This is how the JButton will be represented in a SwingML form. The word BUTTON was selected to represent the element and the following properties were selected to be part of its specification:
NAME, TEXT and ORIENTATION.

All String constants are defined as static variables in the class org.swingml.xml.Constants, new constants have to be defined in this class. By convention NAME is used as the id of the element, TEXT is used as any text that will be displayed by the Swing component and ORIENTATION is used to specify the location if the LAYOUT of the parent is BorderLayout.

The specification for the BUTTON element is

```
<BUTTON NAME="" TEXT="" ORIENTATION="">
        [LISTENER elements]
</BUTTON>
```

Note [LISTENER elements]. This means that the BUTTON can contain LISTENER elements that will specify the events that the BUTTON supports. For more information on the LISTENER element refer to the SwingML document specification.

In the SwingML form the BUTTON may look like this:

<BUTTON NAME="TestButton" TEXT="Accept" ORIENTATION="North"/>

Step 2. Register element specification.

The element specification has to be defined in the SwingML.dtd.

```
<!ELEMENT BUTTON (LISTENER)*>
<!ATTLIST BUTTON
        NAME       ID                                      #REQUIRED
        TEXT       CDATA                                   #REQUIRED
        ORIENTATION (North|South|Center|East|West)   #IMPLIED
>
```

Step 3. Create a Model.

Model classes must be placed in the "org.swingml.model" package. By convention the name of the Model should be the name of the Swing element plus the word "Model" therefore the name of our model is "JButtonModel". The Model must be a subclass of "org.swingml.SwingMLModel. This class encapsulates all the attributes that are common for all components as well as their setter and getter methods. These attributes are: name, orientation, text, tooltip, icon, listeners, layout, children.

The JButtonModel must define additional attributes that are going to be supported by the element specification as well as their accessor methods.

The "org.swingml.SwingMLModel" class defines the method validate(), this method should be overwritten by the "JButtonModel" subclass containing code to perform any validation that couldn't be performed at parsing time by the SwingML.dtd. This method is used at runtime by the SwingML Renderer.

To verify the source code please refer to the class "org.swingml.model.JButtonModel" in the src.jar archive that comes along with the SwingML renderer distribution.

Step 4. Create a Mapper.

Mapper classes have the task to map xml elements to their models. Mappers must be placed in the "org.swingml.xml.mapper" package. By convention the name of the mapper should be the name of the Swing component plus the word "Mapper" therefore the name of our mapper is "JButtonMapper."
The "JButtonMapper" must subclass "org.swingml.xml.MapperUtil" and should implement the "org.swingml.xml.Mapper" interface. The "swing.xml.MapperUtil" implements all the necessary code that maps the attributes that are common for all components, these are the attributes that were previously mentioned in the Step 3. Create a Model.
The JButtonMapper must implement the methods "mapModel()", "getModelToMap()" and "mapModelAttributes()". The "mapModel()" method is called by the SwingML Renderer at runtime to initiate the xml element mapping, this method should call the "getModelToMap()" and "mapModelAttributes()" methods.
The method iterate() from the superclass "org.swingml.xml.MapperUtil" should be called at the end of "mapModel()" to continue the iteration over the DOM tree nodes. The "getModelToMap()" method has three main tasks, first it should create a new instance of the Model, then it has to establish a bidirectional relationship between the new Model and its parent using the "addChild()" and "setParent()" methods and then it has to return the new Model instance. Finally the "mapModelAttributes()" should contain code that

maps the values of each attribute node to the Model. The “getAttribute()” method from the superclass “org.swingml.xml.MapperUtil” should be used inside “mapModelAttributes()” to retrieve attribute nodes by name and the method “mapCommonAttributes()” should be called to set the values of the attributes that are common for all components.

To verify the source code please refer to the class “org.swingml.xml.mapper.JButtonMapper” in the src.jar archive that comes along with the SwingML renderer distribution.

Step 5. Register a Mapper.

The “org.swingml.xml.mapper.JButtonMapper” should be registered in the “getMapper()” method in the “org.swingml.xml.MapperUtil”. The Renderer uses the factory pattern to instantiate Mappers at runtime, this is done by passing the name of a node as a parameter and returning a Mapper instance that corresponds to it.

Step 6. Create a Renderer.

Renderer classes have the task of instantiate JFC/Swing components and set their models. Renderer classes must be placed inside the “org.swingml.view.renderer” package. By convention the name of the Renderer should be the name of the Swing component plus the word “Renderer” therefore the name of our renderer is “JButtonRenderer”
The “JButtonRenderer” must subclass “org.swingml.view.RendererUtil” and should implement the “org.swingml.view.Renderer” interface.
The “JButtonRenderer” should provide the implementation for the “renderer()” method, this method is called a runtime by the Renderer to initiate rendering of the Swing component, it should contain code that creates an instance of the Component passing its Model as a parameter and adds the new Component to its parent using the "add()" method.

To verify the source code please refer to the class “org.swingml.view.renderer.JButtonRenderer” in the src.jar archive that comes along with the SwingML renderer distribution

Step 7. Register a Renderer.

The “org.swingml.view.renderer.JButtonRenderer” should be registered in the “getRenderer()” method in the “org.swingml.view.RendererUtil”. The Renderer uses the factory pattern to instantiate Renderers at runtime, this is done by passing an instance of the model as a parameter and returning a Renderer instance that corresponds to it.

Step 8. Create a Component.

Component classes must be placed inside the “org.swingml.component” package. By convention the name of the Component should be the name of the Swing component plus the word “Component” therefore the name of our component is “JButtonComponent”.
The Component should subclass the Swing component that is going to be rendered and implement any listener that will be supported. The “swingml” package contains the “XMLTranslatable” interface. The “XMLTranslatable” interface should be implemented by any Component whose value will be posted back to the server side component. The “XMLTranslatable” defines the method “getXMLValue()”, this method should be implemented containing code that will retrieve the single value of the Component or a SwingML representation for more complex Components like JTree or JTable. In our case the “JButtonComponent” doesn’t post any value back to the server side component therefore the “XMLTranslatable” interface shouldn’t be implemented. The “JButtonComponent” should define a constructor that takes an instance of the “JButtonModel”. An instance of the “JButtonModel” will be passed when the “JButtonRenderer” creates an instance of the “JButtonComponent”. This constructor should contain code that sets the model to the Swing component that is extended, adds any action listener supported and sets the values for its properties like name, text, etc.

The package "org.swingml.event" contains the "EventHandler" class. The “Event Handler” class it is used to provide support for events. If the Component supports events it must get a reference to it by using the “getInstance()” method then the event is handled by calling the method “handleEvent()” and passing the name of the event previously defined in the “org.swingml.xml.Constants” as a parameter. The “handleEvent()” method should be called from each method that has to be implemented to support every required Listener.

To verify the source code please refer to the class “org.swingml.component.JButtonComponent” in the src.jar archive that comes along with the SwingML renderer distribution