MyfacesBuilderPlugin

Myfaces Builder Plugin

In short, the objectives of this plugin are:

- Generate tag classes
- Generate faces-config.xml file
- Generate .tld file
- Generate facelets-taglib.xml file
- Generate component classes (optional)

The previous work and thought about this topic can be found here:

Code Generation

The main guidelines for achieving the objectives are:

1. Use annotations while preserving java1.4 compatibility 2. Possible use a template tool like velocity to generate the files (for avoid recompile of the code when errors found)(in discussion). 3. Simplify the code based on myfaces-faces-plugin.

The tool proposed to achieve (1) above is the qdox library (qdox.sourceforge.net). This is a java source parser that is used as the base for the well-known xdoclet project. QDox can be used to process either "real" source-retention java1.5 annotations (even on java1.4), or doclet tags embedded in javadoc comments with just a very minor alteration to the plugin code. This therefore means that the same plugin can be used to handle JSF1.1-compatible and JSF1.2-compatible code. Therefore both of the two syntaxes can fairly easily be supported concurrently by the same plugin:

```
/**
 * Some Foo property.
 * @JSFProperty literalOnly="true"
 */
public abstract String getFoo();
```

and

```
/**
 * Some Foo property.
 */
@JSFProperty(literalOnly=true)
public abstract String getFoo();
```

Note that processing source annotations with the Sun APT tool is far more difficult; it is a command-line only tool, is a sun-specific extension, requires java1.5, and does not (AFAIK) give any access to the javadoc comments attached to the annotated node. The Qdox library is portable, and supports doclet-style annotations too.

Guidelines and relevant points per module

Metadata generation

- Use qdox to generate the model and save it to an xml file in a neutral way.
- Possibility to construct a model from several metadata files (tomahawk or sandbox), scanning all dependencies for myfaces-metadata.xml
- There is one special case with tomahawk. Tomahawk contains its own tag class hierarchy copied from myfaces core. So it's necessary to define a mechanism to generate this hierarchy (Maybe it's better an option of a mojo or a specific mojo for do that).
- The component should follow a convention similar to this:

```
package org.apache.myfaces.component;

/**
 * Brief desc of component (goes into tld brief desc).
 *
 * More desc of component (goes into tld long desc). And some more desc. Whatever.
 *
 * @JSFComponent
 *   name = "x:somecomponent"
 *   class = "org.apache.myfaces.component.SomeComponent" //(optional; the default is this class)
 *   parent = "org.apache.myfaces.component.SomeComponentParent" //(optional; default is parent class)
 *   type = "org.apache.myfaces.Component"
 *   family = "org.apache.myfaces.Component"
```

```
 *   defaultRendererType = "org.apache.myfaces.Component"
 *   tagClass = "org.apache.myfaces.component.SomeComponentTag" //Required for generation
 *   tagSuperClass = "javax.faces.webapp.UIComponentTag" //default is parent by type tagClass.
 *   tagHandler = "org.apache.myfaces.component.TagHandler"
 *   bodyContent = "JSP" (JSP|empty)
 *   desc = "short description"
**/
public class SomeComponent extends SomeComponentParent {


  /**
   * The Foo property does something (short desc).
   *
   * An example of declaring a property using a non-abstract method.
   * Of course, there will normally be a concrete getter method too.
   *
   * @JSFProperty
   *   fieldName = "foo"
   *   jspName = "foo" (optional; by default derived from method name)
   *   propertyValues = null //FOR ENUMS (optional)
   *   rtexprvalue = "false" //Used on 1.1 only
   *   tagExcluded = "true"
   *   literalOnly = "false"
   *   required = "false" //Used on tld only
   *   transient = "false"
   *
  **/
   public void setFoo(String foo) {
     _foo = foo;
   }

  /**
   * The Bar property does something (short desc).
   *
   * An example of declaring a property using an abstract method; presumably
   * a concrete component subclass will be generated by the plugin, and that
   * will define the real implementation of this method (and the matching
   * setter).
   *
   * @JSFProperty
   *   jspName = "bar" (optional; by default derived from method name)
   *   propertyValues = null //FOR ENUMS (optional)
   *   rtexprvalue = "false" //Used on 1.1 only
   *   tagExcluded = "true"
   *   literalOnly = "false"
   *   required = "false" //Used on tld only
   *   transient = "false"
   *
   **/
   public abstract String getBar();

  /**
   * In some very special cases, it is necessary to know when a value of a component is
   * set on the component class or is retrieved from a ValueBinding/ValueExpression,
   * so proper body methods should be generated, since inner fields are defined with
   * private scope.
   *
   * Notes:
   * <ul>
   * <li>The scope of isSetProperty, getLocalProperty and isLocalProperty is the scope of
   * the abstract method that requires.
   * <li>the inner property is isGetLocalMethodScope and isSetMethodScope
   * </ul>
   *
   * @JSFProperty
   *   name = "property"
   *   fieldName = "baz"
   *   class = "java.lang.String"
   *   isSetFieldMethod = "false" //Create method isSetProperty
   *   isGetLocalMethod = "true" //Create method getLocalProperty or isLocalProperty
   *   stateHolder = "false"
```

```
   *   defaultValue = ""
   **/
   public abstract String getBaz();

   public abstract String getLocalBaz(); //Do not evaluate ValueBinding/ValueExpression


  /**
   * @JSFProperty
   *   isSetFieldMethod = "true"; //Create method isSetProperty
   *   defaultValue = "false"
   **/
   public abstract boolean isBuu(); //If is not defined return false

   public abstract boolean isSetBuu(); //If is not set a value returns false, otherwise true

  /**
   * Disable the getBlah property (just for instances of this type).
   * <p>
   * Sometimes a component inherits from another component, but does NOT want to expose a property
   * that the parent component exposes. For example, t:div extends t:htmlTag, but does not want to
   * expose the "value" property.
   * <p>
   * Doing this is very bad OO design; doing this breaks the Liskov substitution principle, and
   * therefore OO usage is completely stuffed. A subclass *must* provide all the features of the
   * parent type.
   * <p>
   * Nevertheless, some existing code does this so this feature must be supported. Providing a
   * JSFProperty annotation with the exclude attribute causes the .tld etc to NOT define this
   * particular property, for this class only (child classes of this class get it again).
   *
   * Use tagExcluded = "true" do this, but another suggestion proposed (not implemented) is use exclude =
"true".
   *
   * @JSFProperty tagExcluded= "true"
   **/
   public String getBlah() {
     if (this.getClass() == SomeComponent.class) {
        // this component does NOT support this property
        throw new UnsupportedOperationException();
     } else {
       return super.getBlah();
     }
   }

  /**
   * Override the documentation for this property which was inherited from the
   * definition on an ancestor class.
   * <p>
   * This property is used here for .....
   * </p>
   * <p>
   * Where the component behaviour changes, then there will of course be a new
   * implementation of the method so there is somewhere to put the new docs.
   * </p>
   * <p>
   * Unfortunately in some cases the component is just a dumb holder for the property,
   * and what changes is how this component class expects the corresponding renderer
   * class to use this property. That's rather non-OO, but it's how JSF works. In this
   * case we need new docs for the property even though the implementation has not
   * changed; the best proposal so far is to write a trivial method body that just
   * delegates to the parent method.
   * </p>
   * <p>
   * Yes this is inelegant; it would be nice to find a better way to do this...
   * </p>
   *
   * @JSFProperty
   **/
  public Object getValue() {
    return super.getValue();
```

```
  }

}
```

Note that as described in an earlier section, an equivalent syntax with java1.5 annotations can also be supported concurrently by the plugin. For simplicity, only the doclet-annotation form is shown above.

Component class generation

There are three possible scenarios for component generation:

1. Annotated class is the same component file:

No component code is generated. All methods are created manually. The annotations are used in generation of other classes.

```
/**
 * @JSFComponent
 * //NO class and superClass attributes defined
 *
**/
public class Component extends ComponentParent {

}
```

2. Annotated class is an abstract class, generation of a child component class:

One child class are generated based on the annotations of the parent (getter, setter, save and restore).

```
/**
 * @JSFComponent
 *   class = "org.apache.myfaces.component.Component" //The class is a different class (By default is the same).
 *   superClass = "org.apache.myfaces.component.AbstractComponent" //The superclass is the same class.
Generation without copy pattern.
**/
public class AbstractComponent extends ComponentParent {

  public void myMethod(){
    /*.........*/
  }

}

Generates:

/**
**/
public class Component extends AbstractComponent{

}
```

3. Annotated class is a private class that is used as template (never instantiated, just the code is copied inside the generated class) This pattern can be used for myfaces core(HtmlDataTable on myfaces 1.1). The advantage over a template is that this class is inside the source code, so code completion and other IDE features are available. Anyway,if option 2 can be used then is suggested to be used.

```
/**
 * @JSFComponent
 *   class = "org.apache.myfaces.component.Component" //The class is a different class (By default is the
same). Generation applies with copy
 *   superClass = "org.apache.myfaces.component.ComponentParent" //The superClass is a different class
**/
class _AbstractComponent extends ComponentParent {

  public void myMethod() throws MyException{
    /*.........*/
  }

}

Generates:

/**
**/
public class Component extends ComponentParent{

  public void myMethod() throws com.xxx.yyy.MyException{
    /*.........*/
  }

}
```

Component tag class generation

The tag class can be generated like this:

```
/**
 * @JSFComponent
 *    tagClass = "org.apache.myfaces.component.ComponentTag" //Required for generation
 *    tagSuperClass = "javax.faces.webapp.UIComponentTag" //default is parent by type tagClass.
 *
**/
public class Component extends ComponentParent {

  /**
   * @JSFProperty
   *   tagExcluded = "true" //no generation on ComponentTag
   *
  **/
  public void setProperty(String property){
     _property = property;
  }

}
```

General guidelines:

- A component can or cannot have a generated component tag class.
- If a class extends from UIComponentTag and is 1.2, it is replaced to UIComponentELTag and viceversa.

faces-config.xml generation

The equation to generate faces-config.xml is this:

```
faces-config.xml = src/main/conf/META-INF/faces-config-base.xml + "info form component annotations" + "info
from renderer annotations" + "info from renderkit annotations" + "info from validators and converters"
```

The renderer should follow this minimal notation:

```
/**
 * @JSFRenderer
 *   renderKitId = "HTML_BASIC"
 *   family = "org.apache.myfaces.Component"
 *   type = "org.apache.myfaces.Component"
 *
**/
public class ComponentRenderer extends Renderer{

}
```

.tld generation

The equation for tld is this:

.tld = src/main/conf/META-INF/customtld-base.tld" + "info form component annotations"

For each component, the tld contains:

- Description about this component
- Properties defined in the component class
- Properties defined on the parent(s) component following the type hierarchy.

General guidelines:

- A component that inherit properties from its parent can exclude some properties.

Code completion and IDE support

This is a xml file that you can use with Doclipse to add doclet code completion and comments to make easier the develop of custom components

```
<?xml version="1.0" encoding="UTF-8"?>
<doclipse>
        <description>Code review tags</description>
        <tag name="@JSFComponent" target="class"
                doc="Define that this class is a component class.">
                <attribute name="name" required="true"
                        doc="The name of the component in a page (ex: x:mycomp)." />
        <attribute name="class" required="false"
            doc="The class that implements this component. If not exists this is generated." />
        <attribute name="parent" required="false"
            doc="The parent class which inherits this class. If not set, the superclass of the current class is
used" />
        <attribute name="family" required="false"
            doc="Family that belongs this component. If not defined, it try to get the value of the field
COMPONENT_FAMILY."/>
        <attribute name="type" required="false"
            doc="Type of component. If not defined, it try to get the value of the field COMPONENT_TYPE."
/>
        <attribute name="defaultRendererType" required="false"
            doc="Renderer type used to identify which renderer class use. If not defined, it try to get the
value of the field DEFAULT_RENDERER_TYPE."/>
        <attribute name="canHaveChildren" required="false"
            doc="Indicate if the component can have children"/>
        <attribute name="tagClass" required="false"
            doc="Tag class that match this component"/>
        <attribute name="tagSuperclass" required="false"
            doc="Tag super class that inherits the tag class"/>
        <attribute name="tagHandler" required="false"
            doc="Indicate the tag handler of the tag class."/>
        <attribute name="bodyContent" required="false" allowed="empty JSP"
            doc="Indicate if the element accept inner elements or not."/>
        <attribute name="desc"
            doc="Short description"/>
        </tag>
        <tag name="@JSFProperty" target="method"
```

```
                doc="Define that this getter method define a property.">
        <attribute name="required" allowed="true false"
            doc="(true|false) Define if the property is required or not. Default:false" />
        <attribute name="transient" allowed="true false"
            doc="Indicate if the property is not saved and restored its state."/>
        <attribute name="stateHolder" allowed="true false"
            doc="Use saveAttachedXXX and restoreAttachedXXX to save and restore state"/>
        <attribute name="literalOnly" allowed="true false"
            doc="Indicate that the getter and setter does not evaluate EL or ValueBinding expressions."/>
        <attribute name="tagExcluded" allowed="true false"
            doc="Define if this tag is excluded from tld."/>
        <attribute name="returnSignature"
            doc="The full name of the return type for MethodBinding or MethodExpression it uses"/>
        <attribute name="methodSignature"
            doc="CSV full names of the types that are params for methods using this MethodBinding or
MethodExpression param"/>
        <attribute name="defaultValue"
            doc="The default value to set if this property is generated."/>
        <attribute name="desc"
            doc="Short description"/>
        </tag>
        <tag name="@JSFJspProperty" target="class"
                doc="Define a property that belongs to this component but does not have getter defined. There
exists two main scenarios: 1. Add a property that exists on tld but not on component(binding on 1.1). 2.
Exclude an already defined property from the tld. THIS ONLY SHOULD BE USED IN VERY SPECIAL CASES (Use
@JSFProperty instead).">
        <attribute name="name" required="true"
            doc="The name that identifies this property. (ex:border, id, value)"/>
        <attribute name="returnType" required="true"
            doc="The full type or primitive that this property has defined on tag class."/>
        <attribute name="required" allowed="true false"
            doc="(true|false) Define if the property is required or not. Default:false" />
        <attribute name="tagExcluded" allowed="true false"
            doc="Define if this tag is excluded from tld."/>
        <attribute name="longDesc"
            doc="Long description"/>
        <attribute name="desc"
            doc="Short description"/>
        </tag>
        <tag name="@JSFConverter" target="class"
               doc="Indicate the converterId which identifies this class. If not defined, it try to get the
value of the field CONVERTER_ID.">
        <attribute name="name" required="false"
            doc="The name of the component in a page (ex: x:mycomp)." />
        <attribute name="tagClass" required="false"
            doc="The tag class used for this converter, if applies."/>
   <!-- <attribute name="class" doc="The class which is associated this tag."/> -->
        <attribute name="bodyContent" required="false" allowed="empty JSP"
            doc="Indicate if the element accept inner elements or not."/>
        <attribute name="desc"
            doc="Short description"/>
        </tag>
        <tag name="@JSFValidator" target="class" doc="">
        <attribute name="id" required="false"
            doc="Indicate the validatorId which identifies this class. If not defined, it try to get the value
of the field VALIDATOR_ID."/>
        <attribute name="name" required="false"
            doc="The name of the component in a page (ex: x:mycomp)." />
        <attribute name="tagClass" required="false"
            doc="The tag class used for this validator, if applies."/>
   <!-- <attribute name="class" doc="The class which is associated this tag."/> -->
        <attribute name="bodyContent" required="false" allowed="empty JSP"
            doc="Indicate if the element accept inner elements or not."/>
        <attribute name="desc"
            doc="Short description"/>
        </tag>
        <tag name="@JSFRenderer" target="class"
               doc="Indicate that this class is a Renderer. This is registered on faces-config.xml.">
        <attribute name="renderKitId" required="true"
            doc="The renderKitId that belongs this renderer. If no custom renderkit defined you should use
HTML_BASIC."/>
```

```
        <attribute name="family" required="true"
            doc="The component family that this renderer should be applied."/>
        <attribute name="type" required="true"
            doc="The renderer type that is applied this renderer."/>
        <attribute name="desc"
            doc="Short description"/>
        </tag>
        <tag name="@JSFRenderKit" target="class"
               doc="Identifies a RenderKit class">
            <attribute name="renderKitId" required="true"
            doc="The renderKitId that identifies this RenderKit. This is registered on faces-config.xml."/>
        <attribute name="class"
            doc="The class which is associated this tag."/>
        </tag>
        <tag name="@JSFJspTag" target="class" doc="">
        <attribute name="name" required="true"
            doc="The name of the component in a page (ex: x:mycomp)." />
        <attribute name="bodyContent" required="true" allowed="empty JSP"
            doc="Indicate if the element accept inner elements or not."/>
   <!-- <attribute name="class" doc="The class which is associated this tag."/> -->
        <attribute name="desc"
            doc="Short description"/>
        </tag>
        <tag name="@JSFJspAttribute" target="method"
                doc="Define a tag attribute. This doclet should be used inside a @JSFJspTag class, to define
individual tag classes used in JSF, like f:verbatim or f:actionListener">
        <attribute name="required" allowed="true false"
            doc="(true|false) Define if the property is required or not. Default:false" />
        <attribute name="rtexprvalue" allowed="true false"
            doc="(true|false) This value is put on the tld when applies." />
        <attribute name="desc"
            doc="Short description"/>
        </tag>
    <tag name="@JSFJspAttribute" target="class"
        doc="Define a tag attribute. This doclet should be used inside a @JSFJspTag class, used define
individual tag classes used in JSF, like f:verbatim or f:actionListener">
        <attribute name="name" required="true"
            doc="The name of the attribute. This is the name on the tld"/>
        <attribute name="className" required="true"
            doc="The class or type the component must refer on the tag class. On 1.1 is java.lang.String always
and on 1.2 is javax.el.ValueExpression or javax.el.MethodExpression."/>
        <attribute name="required" allowed="true false"
            doc="(true|false) Define if the property is required or not. Default:false" />
        <attribute name="rtexprvalue" allowed="true false"
            doc="(true|false) This value is put on the tld when applies." />
        <attribute name="longDescription"
            doc="Long description. By default, it takes what is inside comment area."/>
        <attribute name="desc"
            doc="Short description"/>
    </tag>
        <tag name="@JSFExclude" target="method"
                doc="Indicate that this code must be excluded from copying when generation using a template
package scoped class is used. THIS ONLY SHOULD BE USED IN VERY SPECIAL CASES (Use abstract pattern to add
custom code on component class instead).">
        </tag>
        <tag name="@JSFExclude" target="field"
                doc="Indicate that this code must be excluded from copying when generation using a template
package scoped class is used. THIS ONLY SHOULD BE USED IN VERY SPECIAL CASES (Use abstract pattern to add
custom code on component class instead).">
        </tag>
</doclipse>
```