ADF Code Corner

001. How to Access Attributes of a Declarative Component from a Managed Bean

twitter.com/adfcodecorner

Abstract:

In an earlier how-to document I explained how to build and work with declarative components in ADF such that it displays data passed from the containing page. As you would have guessed to this time already, there are more use cases to cover for declarative components. The use case in this article describes how to access an attribute within a declarative component from a managed bean that is part of it. For example, to write a generic component that allows the containing page to pass in e.g. a download URL to process, the attribute of the declarative component doesn't need to be bound to a UI component. This however means that developers need to find a way to access the hidden attribute programmatically - and of course there is.

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
26-JUN-2008

Oracle ADF Code Corner is a loose blog-style series of how-to documents that provide solutions to real world coding problems.

Disclaimer: All samples are provided as is with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.

Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: http://forums.oracle.com/forums/forum.jspa?forumID=83

Introduction

Declarative components in ADF allow ADF Faces developers to build new composite ADF Faces UI components out of existing components. Oracle JDeveloper 11 then creates all the tag library files and deployment artifacts that are needed to re-use this component in other web projects. To communicate with the outer world, declarative components can have attributes and method references exposed as properties to the containing page

Solution

When you develop a declarative component, you define the attributes and attribute's Java type that should be exposed by the declarative component.

The declarative component that I built for this how-to document contains a command button that, when pressed, prints the value of the hidden attribute "userInput" to the console.

The page source of the component looks as follows.

```
<?xml version='1.0' encoding='windows-1252'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
          xmlns:af="http://xmlns.oracle.com/adf/faces/rich">
  <jsp:directive.page contentType="text/html;charset=windows-1252"/>
  <af:componentDef var="attrs" componentVar="component">
    <af:commandButton text="Press Me"
       actionListener="#{DeclarativeComponentHelper.onButtonPressed}"/>
    <af:xmlContent>
      <component
        xmlns="http://xmlns.oracle.com/adf/faces/rich/component">
        <display-name>ButtonAccessToAttribute</display-name>
        <attribute>
          <attribute-name>userInput</attribute-name>
          <attribute-class>java.lang.String</attribute-class>
          <required>true</required>
        </attribute>
        <component-extension>
          <component-tag-namespace>component</component-tag-namespace>
          <component-taglib-uri>/custom/buttons</component-taglib-uri>
        </component-extension>
      </component>
    </af:xmlContent>
```

```
  </af:componentDef>
</jsp:root>
```

Note the af:componentDef element has two attributes defined **var="attrs"** and **componentVar= "component"**. The "**attrs**" string is what you use to reference a declarative component from Expression Language on a visual UI component: ${attrs.firstName} for example references the firstName attribute of a declarative component.

The "**component**" value however grants you access top the declarative component as a whole and is what we will use to access and read from a hidden attribute. While you also could use the "attrs" attribute to access the attribute, using the more generic "component" handle is less code to write if there is more than one attribute to interact with.

In the above example, the hidden attribute is "userInput", which is of type String. While this attribute is not hidden from the containing page, it is not displayed on the declarative component UI - thus hidden.

The declarative component is deployed together with a faces-config.xml file that allows you to create and configure a managed bean. In this example, the managed bean is named "DeclarativeComponentHelper" and contains an ActionListener method,which is referenced from the commandButton

```
<af:commandButton text="Press Me"

actionListener="#{DeclarativeComponentHelper.onButtonPressed}"/>
```

The managed bean code looks as follows:

```
 1  package adf.sample;
 2
 3
 4  import javax.el.ELContext;
 5  import javax.el.ExpressionFactory;
 6  import javax.el.ValueExpression;
 7
 8  import javax.faces.context.FacesContext;
 9  import javax.faces.event.ActionEvent;
10
11  import oracle.adf.view.rich.component.fragment.
           UIXDeclarativeComponent;
12
13
14  public class DeclarativeComponentHelper {
15    public DeclarativeComponentHelper() {
16    }
17
18    public void onButtonPressed(ActionEvent actionEvent) {
19      UIXDeclarativeComponent  _this = (UIXDeclarativeComponent)
          getValueObject("#{component}",UIXDeclarativeComponent .class);
20
21        String s = (String)_this.getAttributes().get("userInput");
22        System.out.println("The provided value was: "+s);
23     }
24
25     private Object getValueObject(String expr, Class returnType){
26       FacesContext fc = FacesContext.getCurrentInstance();
27       ELContext elctx  = fc.getELContext();
28      ExpressionFactory elFactory =
                          fc.getApplication().getExpressionFactory();
```

```
29      ValueExpression valueExpr =
          elFactory.createValueExpression(elctx,expr,returnType);
30          return valueExpr.getValue(elctx);
31      }
32  }
```

As mentioned before, the bean prints the content of the attribute to the console. In Oracle JDeveloper, the console is the message window. The two lines of code that access the attribute value are

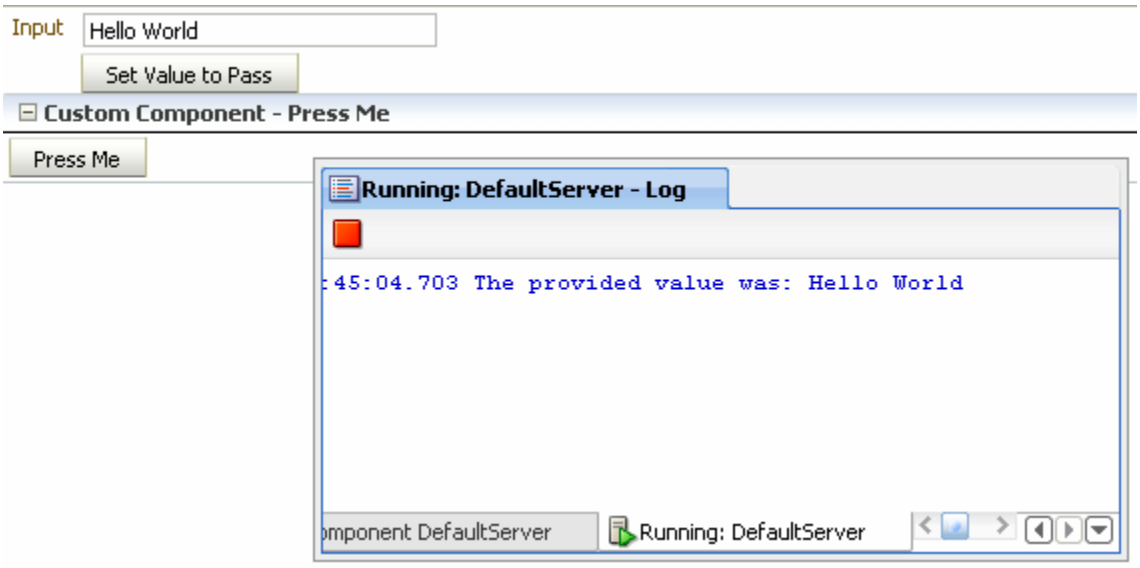
```
19  UIXDeclarativeComponent  _this = (UIXDeclarativeComponent)
        getValueObject("#{component}",UIXDeclarativeComponent .class);
20
21  String s = (String)_this.getAttributes().get("userInput");
```

All declarative components are subclasses of **UIXDeclarativeComponent**, so that it isn't needed to know the exact class name of the new declarative component to type cast the object.

The UIXDeclarativeComponent contains a method getAttributes() that returns a HashMap of all attributes defined for a given component. Thus, to access the "userInput" attribute, the managed bean uses a call to get("userInput").

As mentioned earlier, the declarative component itself is accessible through the "component" string, which can be used in a JSF 1.2 value expression, as shown in the getValueObject() method above.

Running the sample, the following screen is displayed for you to play with



Type a value into the input text field and press the "Set Value to Pass" button to write the information into the ProcessScope. Pressing the "Press Me" button of the declarative component then reads it from there for printing to the console.

Download

You can **download** the Oracle JDeveloper workspace **from the ADF Code Corner** site:
http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html

The workspace contains two projects, one for the custom declarative component and one for testing.