Using Java Components in Oracle Forms Applications

An Oracle Technical White Paper
January 2000

ORACLE

INTRODUCTION

Oracle Forms Server is an internet application development tool and runtime platform. Using a unique, scaleable and highly optimized architecture, Oracle Forms Server enables you to develop and deploy applications using thin client technology in a scaleable and efficient way without sacrificing the capabilities of a rich user interface. Moving applications to the web no longer means you have to downgrade the user interface - Oracle Forms Server supports all the rich user interface components such as poplists, list-of-values, auto-reducing lists, etc. typically found and used in more traditional desktop bound applications in Web based applications.

Oracle Forms Server 6i extends the richness and capabilities of the user interface by allowing you to embed and use your own custom Java components in the application. This integration of Oracle Forms Server 6i and custom Java code is achieved through the use of the Pluggable Java Components (PJC) interface that has been exposed in the 6i release. Using the PJC interface it is possible to incorporate off the shelf JavaBeans into an application, to modify the default behavior of one of the standard Developer Java UI components, or to completely replace one of the standard Developer Java UI components with a custom Java component that displays and behaves in a completely different way.

This paper focuses on how to build Pluggable Java Components (PJC) and to use them within Oracle Forms Server 6i. applications.

ORACLE FORMS SERVER ARCHITECTURE

Oracle Forms Server enables applications to be run as internet applications through the use of a unique Java based client and application server architecture. This architecture employs the use of a powerful application server to execute the application logic and a generic Oracle Forms Server Java client that is able to render the user interface for any Oracle Forms application. The Oracle Forms Server Java client facilitates the interaction between the end user and the application running on the application server using an intelligent message passing mechanism which utilizes caching and compression to reduce network traffic.

This paper will only address the relevant portions of the architecture as they pertain the use of PJCs within Forms.

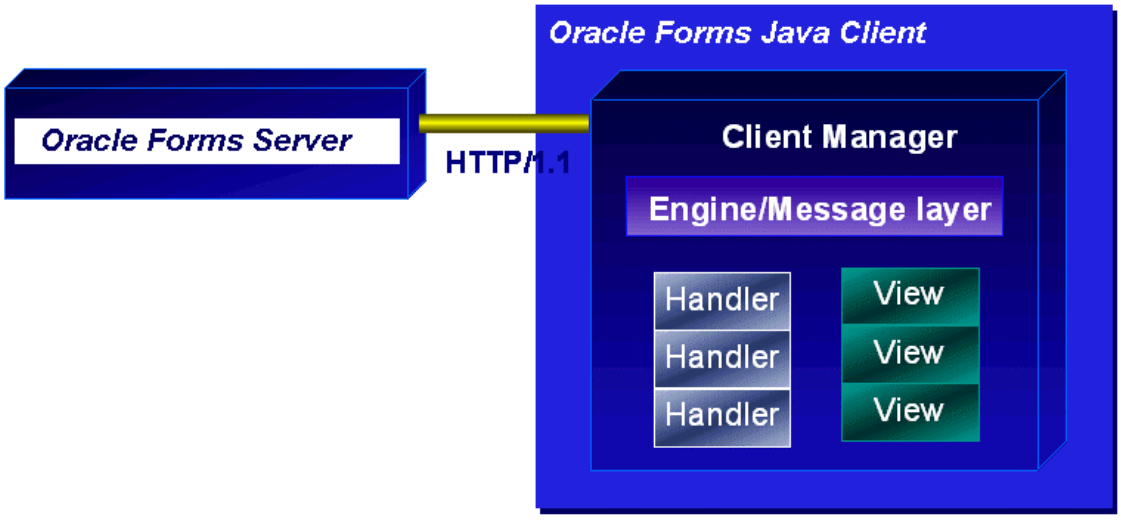

Figure 1: Oracle Forms Server Architecture

THE ORACLE FORMS SERVER JAVA CLIENT

The Oracle Forms Server Java client is a generic Java applet that is able to display the user interface of any Oracle Forms application and respond to user interactions in a highly optimized manner. A description of the user interface is transmitted from the Oracle Forms Server to the Oracle Forms Server Java client when the application is first started and as it is run. The Oracle Forms Server Java client renders itself in accordance with the user interface description contained in the messages. As the user interacts with application, messages are passed back and forth between the Oracle Forms Server Java client and the Oracle Forms Server using an intelligent synchronization mechanism. This intelligent synchronization mechanism enables the messages that are being created by both the Oracle Forms Server Java client and the Oracle Forms Server to be buffered and then transmitted as a compressed bundle at appropriate synchronization points in the application. The messages are cached on both the Oracle Forms Server Java client and the Oracle Forms Server which enables only the changes made to the messages to be passed rather than the entire message.

To provide this message buffering and caching capability, the Oracle Forms Server Java client contains a sub component called the dispatcher that handles the receipt and transmission of all messages between the Oracle Forms Server Java client and the Oracle Forms Server. The dispatcher will receive messages sent to the client from the application running on the Oracle Forms Server and then route those messages directly to the specified user interface component. The user interface component will then interpret and act on the message. When a user interface component needs to

send changed data or the details of an event occurrence to the running application on Oracle Forms Server it does so by creating a message and handing it to the dispatcher. The dispatcher will place the message into the message buffer and cache for storage and diff'ing until it is forwarded to the server at an appropriate synchronization point.

ORACLE FORMS JAVA UI COMPONENTS

Each Oracle Forms native UI component has an equivalent representative in Java. This enables the same application to be run in either client/server mode or in internet mode unchanged. The Java representations of the Oracle Forms native UI components are created using the Java lightweight component model. Using the lightweight component model means that the Oracle Forms Java UI components are rendered completely, rather than relying on the peer UI objects provided by the windowing system of the client operating system. This means that the Oracle Forms Java UI components appear visually the same and have similar behavior across different client operating systems.

Each Oracle Forms Java UI component is implemented using two different classes; a Handler class and a View class. This two class representation is a variation of the standard Model-View-Controller design (MVC) pattern but still adheres to the general goal of separating the data storage aspect from the visual display aspect. In the Oracle Forms Server architecture the Handler class acts both as the Model and Controller whilst the View class acts as the View.

The Handler class is responsible for both maintaining the current value of any data and controlling the visual representation of the data. All server based interaction with the View class is conducted through the Handler class. The Handler class may register itself as an event listener for events that are generated by the View class. The Handler class itself interacts directly with the message dispatcher to send and receive messages to and from the Oracle Forms Server.

The View class is singularly responsible for presenting the data to the user in some manner and handling user input. The View class may allow the data to be changed by the user. This is dependent on the type of UI component the View class is representing and the properties it has set. The View class propagates any data changes made back to the Handler class using the Java event model.

THE IVIEW INTERFACE

To enable the Handler class to interact with and control the View class, the View class implements a

public interface; `oracle.forms.ui.IView`. This interface describes all of the methods the Handler class uses to manage and interact with a View class including lifecycle, property manipulation, event handling and component display methods.

Any Java class that is to be used within the Oracle Forms Server Java client must provide an implementation of this interface. All Oracle Forms Java UI components implement this interface. Because a Pluggable Java Component is just a different View class, it must also provide an implementation of this interface.

Figure 2 contains the definition of the IView interface.

```
public void init(IHandler handler);
```

This method is called immediately after the object is constructed. This method passes the object a reference to it's Handler and gives it a chance to perform any initialization that it requires.

```
public void destroy();
```

this method is called when the object is no longer required. This method gives the object a chance to free up any system resources, that it holds.

```
public Object getProperty(PropertyID id);
```

This method returns the value of the requested property. Each View class must support the properties listed in the following sections. If the requested property is not supported by this Object, this method must return null.

```
public boolean setProperty(PropertyID id, Object value);
```

This method sets the value of the specified property. Each View must support the properties listed in the following sections. If the requested property is not supported by this Object, this method must return false, otherwise it must return true.

```
public void addListener(Class type, EventListener listener);
```

This method adds a listener of the specified type. The types of Listener that each View type must support is listed in the following sections.

```
public void removeListener(Class type, EventListener listener);
```

This method removes a listener of the specified type. The types of Listener that each View type must support is listed in the following sections.

```
public void paint(Graphics g);
```

In this method, the View must paint itself, using the AWT Graphics object provided. For subclasses of Component, this method is called by the Component's Container. For other Objects this method will be called by the Object's Handler

```
public void repaint(Rectangle r);
```

In this method, the View must invalidate the rectangle provided. If the rectangle is null, the entire object should be invalidated.

Figure 2: the `IView` interface definition

COMPONENT PROPERTIES

Properties are used to specify the required behavior and state of a component. The behavioral aspect defines how the object interacts with the running environment. An example of this is the `CAN_TAKE_FOCUS` property for a Textfield. If this is set to true then the component will be able to accept the user input focus. If this is set to false then the component will not accept the focus.

Other properties can be specified to define and manipulate the state of the component. Using these properties controls how the object displays. Examples of this type of properties would be properties such as `BACKGROUND_COLOR` which would define how the component looked and `FONT` which would specify the font used when the component displays text strings.

A subset of the property set exists that is common to all Oracle Forms UI components. In addition to this subset of common properties, each individual UI component type has additional properties that it supports that specify additional behavior and state for that specific component type.

An Oracle Forms UI component must support the common set of properties as well as those properties that are specific to the type of UI component it is.

Property values are manipulated through the use of a setter method. The values of properties are inspected using a getter method. These conventions are taken from the JavaBean naming standard. All Oracle Forms Java UI components allow the properties that are defined for it to be set and get. This is done through the use of the `setProperty` and `getProperty` methods defined in the `IView` interface.

Each property registered for a Oracle Forms Java UI component is created and stored within the ID class. This class, `oracle.forms.properties.ID` stores a static instance of the property which can be passed around and accessed from the different methods in the `IView` interface.

The ID class contains a static `registerProperty` method that allows developers to create and register additional properties. When a new property is required by a PJC, the `ID.registerProperty` method is invoked with the name of the new property as a parameter. The `registerProperty` method will register the new property of the given name with the ID

class and return a reference to it that can then be used within the Java class to identify the property at a later point.

```
/**
 * Forms property registration - used to set the clock face color
 */
public static final ID p_FACE = ID.registerProperty("FACE");
```

Figure 3: Registering a new property

The name used to create the new property is used to identify the property from the PL/SQL environment of Forms using the Forms built-ins, allowing it to be programmatically manipulated by a running application.

```
SET_CUSTOM_ITEM_PROPERTY('CLOCK_BEAN_ITEM', 'FACE', 'WHITE');
```

Figure 4: Using new properties from PL/SQL

A reference to the ID class is passed to the `setProperty` and `getProperty` methods defined by the IView class to identify which property is to be set and/or get on the Oracle Forms Java UI component. The `setProperty` or `getProperty` method performs the required actions to set or get the property from the PJC.

```
/**
 * Set the property referenced in pid to the Object value
 *
 */
public boolean setProperty(ID pid, Object value)

/**
 * Get the value of the property referenced in pid.
 *
 */
public Object getProperty(ID pid)
```

Figure 5: The setProperty and getProperty methods

When the new property is required to be set/get, the ID class identifying the additional property is passed to the setProperty/getProperty method on the JavaBean or PJC just as if it were a default property. This enables PJCs to be operate in exactly the same manner as the Oracle Forms Java UI components.

EVENTS

The Java event model is used by Oracle Forms Java UI components to inform their Handlers (and other interested parties) when something of interest has occurred. The Oracle Forms Java UI component will act as the event source and will send (or fire) the event off to all event listeners. The Handler (or other interested party) will act as an event listener and will receive (or handle) the event. When an user event occurs, the Oracle Forms Java UI component will fire the event on the event listener by invoking a method on the listening object. This same event model is used by PJCs to interact with their Handler classes.

The `IView` interface defines a generic method for handling the addition and removal event listeners.. When a component wishes to be notified of events that are generated by the Oracle Forms Java UI components, it registers as an EventListener with the component..

`IView` defines a single event listener registration method, `addListener(Class type, EventListener listener)` that components use to register themselves to be notified of events when they occur. The type of events that the event listener is listening for is indicated by the first parameter. When this method is invoked the Oracle Forms Java UI component stores the listener object in some internal storage object. When the event type for which the listener has registered occurs, the component fires the event by invoking the appropriate method on all listeners that have registered for that type of event.

`IView` defines a single event listener removal method, `removeListener(Class type, EventListener listener)` that components use to remove themselves from being notified when an event occurs. When this method is invoked the Oracle Forms Java UI component removes the listener object from the list of registered listeners it fires events on.

A common set of event listeners are used by all of the Oracle Forms Java UI components and PJCs. In addition to the common set of event listeners, additional event listeners which are specifically related to the type of the UI component may need to be supported. For example, a UI component that is to be used as a `TextField` item should provide support `TextListener` event listeners, while a UI component that is to be used as a `CheckBox` item should provide support for `ItemListener` event listeners.

USING JAVABEANS IN ORACLE FORMS APPLICATIONS

Oracle Form Builder 6i enables you to integrate and use standard JavaBeans in your applications. The JavaBean is instantiated by the Oracle Forms Server Java client at runtime which enables it to be used by the application. A JavaBean is simply an unspecialized type of PJC. Unlike other PJCs (such as a Button which knows about being pressed) the JavaBean has no assumed behavior.

A JavaBean is specified for use in the Form Builder using a new Forms item; BeanArea. A BeanArea item is a special instance of a custom item. A BeanArea item has a property, Implementation Class. This property is used to specify the fully qualified name of the Java class that the BeanArea item should instantiate. When the BeanArea has the implementation class specified, the Form builder displays the JavaBean within the BeanArea item in the Layout Editor.

A JAVABEAN BECOMING A MANAGED COMPONENT

A JavaBean is a PJC which is of an unspecialized type. Like all other PJCs, in order to be able to be managed and controlled by the Forms application, an implementation of the `IView` class must be provided. This can be done directly by the JavaBean class itself or for the JavaBean through the use of a helper class as outlined below:

- The JavaBean can implement the IView interface and provide implementations for each of the methods defined within it. This suits situations where you are creating the JavaBeans from scratch (or have access to the source code) and where the JavaBean will only ever be used in a Forms environment. A convenience class has been provided to make this task easier. The `oracle.forms.ui.VBean` class provides an empty implementation of the IView interface. You can derive the JavaBean from this class and provide additional method implementations which override the default method implementations to customize the behavior.

- You provide a wrapper class can which acts as a broker between the JavaBean and Forms. The wrapper class provides the implementation of IView using either of the two methods above and in turn invokes the appropriate methods of the JavaBean when required. This situation is the most suitable for the use of 'off the shelf' JavaBeans where the source code is not available and the JavaBean is intended for general use.

IMPLEMENTING THE IVIEW INTERFACE DIRECTLY

When you are creating or modifying a JavaBean for the express purpose of using it in a Forms environment, then the JavaBean class itself may provide the implementation of the IView interface.

This can be achieved in the following two ways.

Declaring that the class implements the IView interface when specifying the class name for the JavaBean. This requires that a method implementation be provided for each of the methods defined in the IView interface.

```
public class ClockWrapper
       implements oracle.forms.ui.IView
```

Figure 6: class definition implementing IView interface

Making the JavaBean a subclass of the `oracle.forms.ui.VBean` class. The `VBean` class provides an implementation of the IView interface. You only need to provide implementations of methods where the default functionality is to be overriden.

```
public class ClockWrapper
       extends oracle.forms.ui.Vbean
```

Figure 7: class definition using the VBean convenience class

By creating the class as a subclass of the `VBean` class the amount of work you are required to do to provide an implementation of the `IView` interface is reduced since only those methods that need to be overridden must be developed. This may not always be possible if the JavaBean itself is a subclass of another class.

USING A JAVABEAN WRAPPER CLASS

If you wish to integrate an existing JavaBean into your application, the best way to approach this is to create a simple wrapper class which acts as an intermediary between Forms and the JavaBean itself. The wrapper class provides the glue to map the JavaBean to the Oracle Forms Server Java client. The wrapper class instantiates the JavaBean and provides the implementation of the IView interface as required. The wrapper class itself may directly implement the IView interface or it may subclass from the VBean class and simply provide the required methods to override the default implementations contained in the VBean class as discussed above.

The wrapper class acts as a delegation component by sitting between the Forms applet and the JavaBean. When the wrapper class is initialized, the wrapper class creates and stores a reference to

an instance of the JavaBean. The wrapper class provides the implementation of the getProperty and setProperty method so that when Forms calls them to set a property on the JavaBean, the wrapper class delegates the operation to the JavaBean by calling the appropriate getter and/or setter method on the JavaBean.

The Java code snippet below shows the constructor for the wrapper class and the actions it performs to create and add an instance of the JavaBean to the wrapper.

```
/**
 * The constructor for the wrapper class which instantiates the
 * JavaBean component and adds it to the wrapper container.
 **/
public ClockWrapper()
{
  try
  {
     mClock = new Clock();
     mClock.setVisible(true);
     mClock.setEnabled(true);
     add("Center",(Component)mClock);
     //
     // start the clock running ...
     //
     mClock.start();

     //
     // register this wrapper as a listener for the clock Alarms
     //
     mClock.addAlarmListener(this);
   }
   catch(Exception e)
   {
     e.printStackTrace();
   }
}
```

Figure 8: the ClockWrapper initialization tasks

SETTING AND GETTING PROPERTIES ON THE JAVABEAN

The properties of the JavaBean will be controlled via the setProperty and getProperty[1] methods defined in the IView interface and implemented by the JavaBean or the wrapper class. The setProperty method will be invoked when a property on the JavaBean is to be modified or initially set. When a running application needs to get the value of a JavaBean property then the getProperty method will be invoked. The desired property will be specified as parameter to the setProperty or getProperty of type ID.

When setProperty is invoked the method call will be passed two parameters; the first parameter of type ID indicates the desired property that is to be set. The second parameter, of type Object contains the value the desired property needs to be set to. The method call returns a boolean value indicating the result of the desired operation. The value of the desired property is declared as an object of type Object. The setProperty method implementation must cast this Object type to the type required for the JavaBean property.

The setProperty method can determine what property is to be modified by using an if based statement, comparing the input parameter property ID with the list of default and additionally registered properties that it can handle.

```
public boolean setProperty(ID pid, Object value)
{
  boolean result=true;

  if(pid == p_CLOCKRADIUS)
  {
    mClock.setRadius(Integer.parseInt((String)value));
  }
  else if(pid == p_BACKGROUND)

  {
    mClock.setColor((mClock.BACKGROUND),
    Utils.getColorFromString((String)value));
  }
  else if(pid == p_FACE)
  {
    mClock.setColor((mClock.FACE),
    Utils.getColorFromString((String)value));
  }

  //
  // handle other custom properties here
  //
```

[1] The getProperty method for BeanArea items is not available in Oracle Forms Server release 6.0.5. since there is no PL/SQL built-in. It will be enabled from Oracle Forms Server Release 6.0.6 onwards with a corresponding PL/SQL built-in.

```
  return result;
}
```

Figure 9: setting JavaBean properties from the wrapper class

When getProperty is invoked the method call will be passed a single parameter of type ID which indicates the property for which the value should be returned. The method call will return a value of type Object which represents the value of the desired property on the JavaBean. If the JavaBean property is a Java primitive such as in int, char, boolean, etc. then the getProperty method must convert these to a reference type for return to the calling object.

```
/**
 * Method in the IView interface that
 * allows the value of a specific property to be set to a
 * specific value by the Forms application when component
 * is required to be initialized.
 * @param     pid     the properties value that is to be returned
 * @return    the value of the property that was specified
 */

public Object getProperty(ID pid)
{
  Object result = null;

  if(pid == p_CLOCKRADIUS)
  {
    result = new Integer(mClock.getRadius());
  }
  else
    result = super.getProperty(pid);

  return (Object)result;
}
```

Figure 10: getting JavaBean properties from the wrapper class

INVOKING JAVABEAN METHODS FROM FORMS

The invocation of methods contained in the JavaBean is achieved using a similar approach to setting the properties of the JavaBean. Using the ID class, the methods are registered as properties with an identifying name. The identifying name is used from the PL/SQL built-in to indicate to the JavaBean that a specific method is to be invoked using the SET_CUSTOM_ITEM_PROPERTY built-in. The setProperty method is called with the ID of the registered method. The setProperty

method determines what JavaBean method to call based on the ID parameter.

The following Java code snippet shows the calling of a method on the JavaBean from the setProperty method.

```
public boolean setProperty(ID pid, Object value)
{
  boolean result=true;

  else if(pid == m_CLOCKSTOP)
  {
    mClock.stop();
  }
  else if(pid == m_CLOCKSTART)
  {
    mClock.start();
  }
  …
  // handle other properties here
}
```

Figure 11: mapping JavaBean methods in the wrapper class

The following PL/SQL code snippet shows how to invoke the JavaBean method from Forms. The PL/SQL built-in SET_CUSTOM_PROPERTY takes the name of the Forms item containing the JavaBean, the character string representing the desired property and the value of the property.

```
SET_CUSTOM_ITEM_PROPERTY('CLOCK_BEAN_ITEM','CLOCKSTOP',0);
```

Figure 12: invoking JavaBean methods from PL/SQL

CONTROLLING THE JAVABEAN FROM FORMS

A JavaBean that is used in an Oracle Forms application can be programmatically controlled from the application using the PL/SQL `SET_CUSTOM_ITEM_PROPERTY` built-in. The built-in requires you to specify the name of the property and the value to be assigned to that property. The built-in uses the name that the additional property was registered with to identify the property to be set. The built-in causes the `setProperty` method to be invoked on the JavaBean with the matching `ID` class for the property name and the value the property is to be assigned.

The following code shows the use of the PL/SQL built-in, SET_CUSTOM_ITEM_PROPERTY to change the value of the animation rate of the JavaBean.

```
SET_CUSTOM_ITEM_PROPERTY('CLOCK_WRAPPER_BEAN', 'FACE',:FACE);
```

This results in the setProperty method being invoked on the JavaBean wrapper. The parameters passed to the method will be a reference to the property instance that was registered with the name 'FACE' and a String representation of the value of the FACE item. The JavaBean setProperty implementation converts the Object parameter to the required type and invokes the appropriate method on the JavaBean.

COMMUNICATING WITH FORMS FROM THE JAVABEAN

A JavaBean can communicate with a running Forms application by creating and dispatching custom events. This is facilitated by a Forms application registering an event listener of type CustomListener with the JavaBean through the JavaBeans' Handler class. The JavaBean must provide support for this event listener if it is to fire events on the form (the VBean class provides support for this.) When a CustomEvent is created, it is used when invoking the customActionPerformed method on the registered CustomEvent listeners or when invoking the dispatchCustomEvent provided by the VBean class.

To indicate to Forms which event has occurred, an additional property should be registered with the ID class for each of the different events that the JavaBean will send back to Forms. The new ID reference that represents the event is used when constructing a new CustomEvent object.

JavaBeans pass data with the event back to forms via the Handler object. The Handler object contains a setProperty method which the JavaBean uses to set values of any identified properties. Any number of properties can be set on the Handler to return event data.

To notify the event listeners that an event has occurred, the VBean class provides a dispatchCustomEvent method. This method invokes the customActionPerformed method on each of the event listeners that have registered with the JavaBean. If the IView interface is being implemented directly, the you must provide event listener registration and removal methods for the CustomListener class. A method to fire the event on each of the registered event listeners is also required.

The following Java snippet of code shows the construction of a CustomEvent. The Handler is

populated with the event data dispacthed using the method supplied with the VBean class.

```
/**
 * Method in the AlarmListener interface that is
 * called when an Alarm is fired on the Clock JavaBean.
 * @param    ae    the details of the alarm that was fired
 **/

public void alarmFired(AlarmEvent ae)
{
  //
  // create the Forms custom event and set it to the alarm date
  //
  try
  {
    CustomEvent ce = new CustomEvent(mHandler,e_ALARMFIRED);
    mHandler.setProperty(p_ALARMTIME,ae.toString());
    dispatchCustomEvent(ce);
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
}
```

Figure 13: Creating and Dispatching a CustomEvent to Forms

When a CustomEvent is fired from the JavaBean it causes the WHEN_CUSTOM_ITEM_EVENT trigger on the BeanArea item in Forms that houses the JavaBean to be raised. The name of the event is stored in the system variable :SYSTEM.CUSTOM_ITEM_EVENT. The Event data sent by the JavaBean is available to the Forms application via the PL/SQL built-ins GET_PARAMETER_LIST and GET_PARAMETER_ATTR.

The PLSQL snippet code below shows the WHEN_CUSTOM_ITEM_EVENT trigger code used to handle CustomEvents dispatched from a JavaBean. The associated event data is also extracted from the CustomEvent. In this case the JavaBean is sending information about the mouse button that was pressed.

```
PROCEDURE handleEvent IS
  eventName varchar2(20) := :system.custom_item_event;
  eventValues ParamList;
```

```
  eventValueType number;

  tempString varchar2(4000);


BEGIN



  IF (eventName = 'ALARMFIRED') then
    eventValues                                          :=
get_parameter_list(:system.custom_item_event_parameters);

get_parameter_attr(eventValues,'ALARMTIME',eventValueType,temp
String);
    displayAlert('Alarm Fired @ ' || tempString);
  END IF;



END;
```

Figure 14: handling Custom Events in PL/SQL

CREATING AND USING PLUGGABLE JAVA COMPONENTS

A PJC can be used in place of a default Oracle Forms Java UI component in an application and be treated as if it were the original component. This type of PJC is of a specialized nature; it has requirements it must meet such as the types of properties it supports and the event listeners it operates with. The PJC is used as a replacement View class for a Forms item. This enables the default functionality and behavior of the standard UI component type to be changed to suit the application. This can be achieved through the modification of one of the Oracle Forms Java UI component through subclassing or by using a separate Java class.

The PJC when used in an application is treated in exactly the same manner as the default Oracle Forms Java component. From the running application's perspective, there is no change. All interaction with the Oracle Forms Server and the Oracle Forms Server Java client is handled as if the PJC were a standard Oracle Forms Java UI component. The running Forms application will pass pass properties to the PJC via it's Handler class which defines how the PJC should look and behave. The PJC must raise events on the running Forms application via it's Handler class when the user

interacts with the application, for example, when a checkbox item value is changed.

SPECIFYING THE USE OF A PJC IN THE APPLICATION

Each Oracle Forms UI component has a property named Implementation Class. This property by default has a value set to be the Oracle Forms UI Java class supplied with the product that represents the specific item type. The default Implementation Class value is not displayed in the property palette. To use a PJC, the Implementation Class property for an item should be set to the fully qualified name of the PJC Java class that is to be used for that item.

PLUGGABLE JAVA COMPONENTS REQUIREMENTS

All PJCs that replace standard Oracle Forms Java UI components act in place of the native control that they replace. The PJC is treated in exactly the same manner as the standard Oracle Forms UI Java component. As was discussed in earlier sections, in order to be a PJC, an implementation of the IView interface must be provided and it must support a defined set of properties and event listeners that are specific to the type of user interface item it is being used for.

PROPERTIES

As with the JavaBean implementation, all PJCs must also provide support for the setting and getting of properties. The setting and getting of the properties are done through the `setProperty` and `getProperty` methods defined in the `IView` interface.

The properties a PJC must provide support for are very specific to the type of user interface item the PJC is to become when it is deployed in an Oracle Forms Server application. The Oracle Forms online documentation accessible from the Oracle Forms Builder contains the definitive list of the properties supported for each user interface item.

EVENTS

For PJCs, events are very important since this is the way that it communicates with the running Forms application on Oracle Forms Server via its Handler class. The Oracle Forms Server Java client uses that standard Java event model; event listeners register interest in the events that a class generates. When an event occurs, the class informs each of the listeners that have registered interest via the invocation of a known method. The Handler registers as a listener for events it is interested in observing with the View class, in this case a PJC .

The View class must provide methods to allow for the registration and removal of event listeners. The Handler class calls the `addListener` method to notify the class that it is interested in an event type and it calls the `removeListener` method to notify the class that it is no longer interested in the event type.

The `addListener` and `removeListener` methods are generic listener methods. They provide a reference to the class which is to be registered as a listener and also a reference to the type of events the listener is interested. This enables the PJC to have multiple listeners of different types to registered for different event types.

The types of events a PJC must provide support for are very specific type the type of user interface item the PJC is to become when it is deployed in an Oracle Forms Server application. For example, an Oracle Forms checkbox item will register a listener of type Item with the View class. If a PJC is to be used as a checkbox item, it must provide support for listeners of type Item and must notify the registered listeners when an ItemSelected event occurs. The Oracle Forms online documentation accessible from the Oracle Forms Builder contains the list of events supported for each user interface item.

```
/**
 * Method in the IView interface that
 * allows a listener object to be registered to listen
 * for events that occur of a certain type specified
 * by the type parameter.
 * @param type the type of the Listener that is being registered
 * @param listener the object to be registered as the listener
 */
public void addListener(Class type, EventListener listener)
{
  if (type == ItemListener.class)
    this.addItemListener((ItemListener)listener);
  else
    super.addListener(type, listener);
}
```

Figure 15: Handling Listener Registration in a PJC

CREATING PLUGGABLE JAVA COMPONENTS

There are two ways a Java you can create a PJC for use within Oracle Forms applications. You can

customize the functionality of one of the standard Oracle Forms Java UI components through the use of subclassing or you can create a completely custom PJC by providing an implementation of the IView interface and support for the necessary properties and events.

CREATING PJCS VIA SUBCLASSING

When you wish to modify the behavior of a native Oracle Forms Java UI component then this can be achieved by creating a new PJC as a subclass of the component you wish to modify. Through the use of subclassing, the PJC will inherit all of the behaviors and properties of the native component. It is then able to modify them by providing different implementations of the methods from the superclass.

All of the Oracle Forms Java UI components may be subclassed. The Oracle Forms Java UI components are all prefixed with the letter 'V' as seen in Figure 16.

ORACLE FORMS JAVA UI CLASSES

```
oracle.forms.ui.Vbean
oracle.forms.ui.VButton
oracle.forms.ui.Vcheckbox
oracle.forms.ui.VComboBox
oracle.forms.ui.VImage
oracle.forms.ui.VPopList
oracle.forms.ui.VRadioButton
oracle.forms.ui.VRadioGroup
oracle.forms.ui.VTextArea
oracle.forms.ui.VTextField
oracle.forms.ui.VTList
```

Figure 16: Oracle Forms Java UI Classes

To modify the behavior of the native Oracle Forms Java UI component, the PJC provides alternate implementations of the methods in the native component.

An example of a PJC that uses subclassing is where a Web style rollover button is desired to be used

within an Oracle Forms Server application. The Oracle Forms Java UI Button component provides support for displaying an image on a button but it does not provide the support to have the image change when the mouse moves on and off of the button. For this example, a new class `RolloverButton` would be created as a subclass of the `oracle.forms.ui.VButton` class. The `RolloverButttton` class would inherit all of the default behavior from the `VButton` class. Additional functionality would be provided in the class to detect the mouse moving over the button component using the `MouseListener` interface and to change the image which was being displayed on the button.

PJCs that are created as subclasses of Oracle Forms Java UI components do not need to provide support for all of the properties that are utilized by the item type since this has already been implemented by the parent class. If the PJC needs to override some of the default behavior that occurs when a specific property is set then the PJC can provide an implementation of the `setProperty` method and intercept the setting of the property. The developer need only provide an implementation for the specific property they wish to work with and can implement whatever functionality is required. The developer does not need to provide an implementation for each of the propeties. A PJC should call the `setProperty` property method of the super class to handle the properties in which it is not interested in.

```
/**
    ** Implementation of IView interface
    **
    ** @param id    - property to be set.
    ** @param value - value of the property id.
    ** @return - true(if the property could be set)
    ** @see IView
    */
    public boolean setProperty(ID pid, Object value)
    {
        boolean success = true;

        if ( pid == IMAGE_NAME_OFF )
        {
          mImageNameOff = (String) value;
          loadImage(OFF,mImageNameOff);
        }
        else if ( pid == IMAGE_NAME_ON )
        {
          mImageNameOn = (String)value;
```

```
        loadImage(ON, mImageNameOn);
      }
      // let VButton class handle all other properties
      else
      {
        success = super.setProperty(pid, value);
      }
      return success;
  }
```

Figure 17: Setting Properties in the PJC

CREATING CUSTOM PJCS

In addition to modifying the behavior of existing Oracle Forms Java UI components, you can create PJCs that have no relation to any existing component and look and operate in any way that you want. You can create an entirely new style of View class that suits your particular applications requirements and have it operate within the application as a native control. The PJC will be of a specific Forms UI item type, such as a Checkbox, Textfield, etc.

In this case, you are entirely responsible for managing the interactions between the PJC and its Handler class, since there is no default behavior inherited from a parent class which can be utilized. The PJC must provide a full implementation of the IView interface. The PJC must operate within the standard Forms model of setting and getting of properties and of notifying the handler when events occur using the standard event model. The list of properties and events that must be supported by the PJC are determined by the type of UI item the PJC is replacing.

An example of a custom PJC is the case of an image style CheckBox, where the states of the CheckBox are represented by images rather than the standard box with a cross in it. The custom CheckBox PJC provides functionality to display an image, to detect mouse clicks and to swap the image when the mouse is clicked on the image area. The custom checkbox PJC needs to support the properties used by the native Oracle Forms Java CheckBox item. These properties would define the look and initial state of the checkbox item. The custom CheckBox PJC also needs to notify the Handler class when the state of the item was changed. For the CheckBox type of UI component, the Handler class registers ItemListeners with the PJC. The PJC invokes the itemStateChanged method on the registered ItemListener classes when the checkbox state is changed.

The code snippet in Figure 18 shows the event handler that is called when the mouse is clicked over

the image. As a result of the mouse being pressed, the fireItemListeners method is called.

```
/**
  * Private class to handle the mouse clicks on the PJC.
  * When the mouse is clicked, the itemstate is 'flipped'
  * to the other state and the ItemListeners are notified via the
  * fireItemListeners method
  *
  */
 class MyMouseAdapter
 extends MouseAdapter
 {
   /**
    * Override the mouse clicked method to flip the itemstate
    * and fire the item listeners
    * @parm me the mouse event that occurred
    */
   public void mouseClicked(java.awt.event.MouseEvent me)
   {
     flipItemState();
     fireItemListeners();
   };
 }
```

Figure 18: Handling moused events in a PJC

The code snippet in Figure 19 shows the code used to notify the ItemListeners that have registered interest in Item events that an event has occurred. The ItemListeners will have been registered by the Handler class and by notifying the ItemListener the Handler will effectively be notified that a state change has occured. This change will be sent to the application running on Oracle Forms Server.

```
 /**
  * Notify all reigstered itemlistener objects that an item
  * event has occured.
  *
  * @param e  the item event that is sent to registered listener
  */
 public void fireItemListeners()
 {
```

```
    ItemListener listener;
    Enumeration enum;
    Vector clone;
    ItemEvent ie = new
      ItemEvent(this,0,this,ItemEvent.ITEM_STATE_CHANGED);

    synchronized(this)
    {
      clone = (Vector)mItemListeners.clone();
    }
    enum = clone.elements();

    while(enum.hasMoreElements())
    {
      listener = (ItemListener)enum.nextElement();
      listener.itemStateChanged(ie);
    }
  }
```

Figure 19: Firing events on the item listeners

DEPLOYING APPLICATIONS USING JAVA COMPONENTS

All Java classes that are used in an Oracle Forms Server application must be downloaded to the running browser before they can be instantiated. If you are using Oracle JInitiator then the JAR caching feature stores downloaded JAR files on the actual client to avoid the download when next the JAR file is referenced. This means that the PJCs used within a Forms application must also be available for download from the same place as the Oracle Forms Server classes.

The Oracle Forms Server Java classes are stored in JAR files. These JAR files are specified in the ARCHIVE tag in the HTML page that is used to launch the application.

To deploy your PJCs there are really only two options. The first option is to place the classes that make up the PJC in the physical directory that is mapped to the CODEBASE virtual directory. When the classloader tries to load the classes you have specified it will check the CODEBASE directory. If the classes are found there, it will load them from over the network into the client. The second option is to put all of the classes that make up the PJC into a JAR file and then specify the name of the JAR file in the ARCHIVE tag along with the Oracle Forms Server JAR files. The class loader will load all the classes from the JAR files specified before the application is run. If you are performing actions such as accessing the local file system from the PJC then it will need to be signed

and the Java signing model will only sign JAR files and not individual class files.

The easiest way to work with PJCs during development and testing is to place them into the CODEBASE directory. The classes must be stored in a directory structure that reflects the package definition of the class.

TIPS AND TRICKS

You will find that creating PJCs and JavaBeans and using them in Oracle Forms applications is relatively straight forward once you have read the online documentation and have taken a look at some of the samples that ship with the product. This section contains some little tips and tricks that I have used during my investigations that might provide some additional help for you as you embark on the road to UI freedom.

USEFUL TIPS

- Try to develop as much of the PJC as possible as a standalone component. This will allow you test and fine tune the component without introducing another layer of potential problems on top. Once you have the component fairly well developed and tested then create the additional code/components/wrappers required to run it in an Form application.

- Use copious amounts of trace writes to help track what is going on. There is no currently no way from the Form Builder to debug a PJC once inserted into an application. Using trace writes from the Java class itself is the easiest way to determine what is happening. Also having some form of class variable which can be set to turn debugging on and off in the code saves a lot of code scrubbing once the class is complete. Logging to a persistent file can also be quite helpful if you are generating a lot of debug messages.

- When debugging your PJCs, run the Forms application using the appletviewer or a certified browser which allows the viewing of the Java console. This will let you see the debugging statements you have in the code. If you run the Form from the Form Builder using the 'Run Web Preview' button then you will not see the messages that are written out since there is no standard out.

- Put all code that you regularly use into a common class that can be used from any PJC. This will save you a lot of rework. Things that are good to reuse are the debugging routines, type conversion routines, etc.

- Type conversion can be a difficult process since the known property values that are set by Forms are mapped to their Java equivalents by the Handler but custom properties registered via the ID.registerProperty() method are not automatically mapped. The setProperty method received the property value as an Object. If you are getting type casting exceptions, using the toString() method on the Object to help pinpoint the type of the Object.

- Check the exceptions that are thrown very carefully for hints as to where the problem lies.

Remember that the default Oracle Forms Java UI components all work so if there's a problem it's usually somewhere in your code.

- The Oracle Forms Server Java class libraries such as f60all.jar have been signed to allow them to run in a trusted mode. Any additional PJCs that you write and include in your applications will not be trusted and therefore run inside of the Java sandbox. If you wish to escape the sandbox to do things such as access the client filesystem then you will need to put your classes in a JAR file and sign it. The client machines will need to install the certificate you use in the signing process in order to verify the integrity of your signed JAR file.

- The IHandler class which is passed to the init method implements the java.applet.AppletStub interface. This class can be used to get access to things such as the document base and code base virtual directories and to generally do applet types of things.

- Take a look at JDeveloper 3.0 which contains a PJC wizard to assist in the construction of Oracle Forms Server 6i PJCs. This wizard provides information about the properties that are available to the different Oracle Forms Java UI components and creates skeleton code methods to perform the accessing of the selected properties.

- Keep it as simple as is possible. Try to keep the environment as simple as you can make it whilst you are developing and debugging your Java code.

CHANGES IN THE ORACLE FORMS 6*i* RELEASE

As this paper was being completed, Oracle Forms Server 6*i* was being finalized. This section contains some additional details of the changes made in the 6*i* release with respect to PJCs.

- The ID class now contains a getName() method which returns the actual name of the property that was used at property registration time. This is a very handy method to use when debugging code to see which properties are being accessed.

- There are a few new built-ins that allow for the manipulation of PJCs at runtime. The new built-ins will now allow for the getting of custom properties (in addition to the setting of them) from JavaBeans and PJCs and will allow for the setting and getting of custom properties on standard PJCs when they are used as replacement items for the default Oracle Forms Java UI components. With the initial release of Oracle Forms Server 6.0 , any additional properties that were registered by a PJC could not be set nor get. Likewise, for JavaBeans running inside of the BeanArea in the initial release of Oracle Forms Server 6.0, properties could only be accessed via a set method and not a get method.

  The new builtins are labelled set_custom_property and get_custom_property. Please consult the Oracle Forms online documentation in the 6*i* release for more information on these built-ins and how to use them.

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
+1.650.506.7000
Fax +1.650.506.7200
http://www.oracle.com/