Enhancements to JavaBean Objects

This document presents enhancements to JavaBean objects in RapidPLUS version 7.01:

- Checking the running status of JavaBean objects
- Implementing recordable events

CHECKING THE RUNNING STATUS OF JAVABEAN OBJECTS

JavaBean objects can be programmed to check their own running status—that is, whether they are currently in design time (Object Layout) or runtime (Prototyper or LiveManuals). This capability enables each JavaBean object to draw a default representation of itself in the Object Layout.

Step 1: Implement BeanContextChild

Each JavaBean object must implement the BeanContextChild interface (com.esim.accessibility.beans.BeanContext). This interface is part of the LiveManuals Java classes and can also be found under <RapidFolder>\java\jbWrapper\. We recommend that you place BeanContextChild.class inside the JavaBean object's .jar file, so that it will work in Rapid 7.00, and as a standalone JavaBean.

In addition, two other class files must be placed inside the JavaBean object's .jar file: com.esim.accessibility.beans.BeanContext and com.esim.util.LightweightCollection.

Step 2: Supply a BeanContext Object to Query Status

After the JavaBean object implements the BeanContextChild interface, Rapid will call the JavaBean object's setBeanContext method and supply it with a BeanContext object. This object is used to query the JavaBean about its status—whether the object is in design time or runtime—by calling its isDesignTime method.

❖ *Note: We recommend that you call the isDesignTime method from the paint method.*

An example application, CounterButton2.rpd, is provided to demonstrate a JavaBean object that draws a default representation in the Object Layout. It also includes comments about how to implement the BeanContextChild interface (located in the \\Examples\JavaBeans\BeanContext folder).

IMPLEMENTING RECORDABLE EVENTS FOR JAVABEAN OBJECTS

In both Rapid and the Scenario Authoring Tool, events (also known as user actions) performed with Rapid objects can be recorded and replayed. In Rapid, recorded events can be replayed in the Prototyper and in the Rapid Reviewer, and can be added as use and test cases in the Document Manager. In the Scenario Authoring Tool, recorded events become an integral part of scenarios/guided tours, which are used with LiveManuals simulations in Web browsers.

JavaBean objects can be programmed so that events performed with them can be recorded, just like other Rapid objects. The required code changes are explained in this section via an example JavaBean object called "CounterButton." This JavaBean object is based on the CounterButton bean from the Sun Bean Development Kit. It comprises a simple button that has a counter displayed in the center of the button. Clicking the button increments the counter and generates an event (called an action in Java).

In our example we created a new JavaBean object, "CounterButtonRecordable" (referred to in this document as the "Bean object"), which extends the functionality of CounterButton, including changes to make its *click* event recordable.

- *Note: The CounterButtonRecordable JavaBean object cannot be used in production; it is only provided for instructional purposes.*

Step 1: Implement RapidAccessible

The Bean object's class must implement the RapidAccessible interface (com.esim.accessibility.interfaces.RapidAccessible). This interface is part of the LiveManuals Java classes and can also be found under <RapidFolder>\java\jbWrapper\. We recommend that you place RapidAccessible.class inside the Bean object's .jar file, so that it will work in Rapid 7.00, and as a standalone JavaBean.

The RapidAccessible interface defines three methods:

METHOD	DESCRIPTION
addRoutableValueListener	Adds/removes listeners for recordable events. A recorder (like the Scenario Authoring Tool) usually adds a listener to a JavaBean, and the JavaBean notifies it whenever the user triggers a recordable event in the JavaBean, so that it can record it.
removeRoutableValueListener	
injectRoutableValue	Plays back an event. It is typically called from the Scenario Player applet.

These methods are used in the following steps.

Step 2: Use the Helper Class, RapidRoutingSupport

A class file called RapidRoutingSupport (com.esim.helpers.RapidRoutingSupport) helps implement recordable events in the Bean object. This class should be packed together with the Bean object's .jar file. (It's small.)

2.1 Add a RapidRoutingSupport member variable to the Bean class

```
// recordable events support
protected RapidRoutingSupport routingSupport;
```

2.2 Implement addRoutableValueListener and removeRoutableValueListener

Implement addRoutableValueListener and removeRoutableValueListener. These methods are usually implemented the same way in every JavaBean object, using the RapidRoutingSupport class:

```
/**
 * Add a listener that allows notification of value changes
 * that can be recorded or routed via the extension services.
 * @param listener java.beans.VetoableChangeListener
 */
public void addRoutableValueListener(VetoableChangeListener listener)
{
    if (routingSupport == null)
          routingSupport = new RapidRoutingSupport(this);

    routingSupport.addRoutableValueListener(listener);
}

/**
 * Remove a listener that allows notification of value changes
 * that can be recorded or routed via the extension services.
 * @param listener java.beans.VetoableChangeListener
 */
public void removeRoutableValueListener(VetoableChangeListener listener)
{
        if (routingSupport == null)
              return;

        routingSupport.removeRoutableValueListener(listener);
    }
```

Now, whenever the user triggers a recordable event in the Bean object, the Bean object should notify the Recorder about it, so that the event will be recorded.

Step 3: Implement Event Recording

A recordable event consists of a textual description (string) and zero or more numeric arguments. The RapidRoutingSupport class provides several routeEvent methods that allow you to pass events to the recorder, with or without arguments.

The recorder can prevent the event from happening, by throwing a PropertyVetoException. This exception is caught by the RapidRoutingSupport object and causes the routeEvent method to return FALSE. In the Bean object's code, you should check the result of routeEvent, and if it returned FALSE, the Bean object must not fire the event or change its state in any way resulting from the event. In the case of the CounterButton JavaBean object, this object fires the "action" event and increments its counter. Neither of these actions should happen if the recorder vetoed the event.

In the CounterButton JavaBean object, a button click occurs when the user presses and releases the mouse. This event is handled in the mouseReleased method:

```
public void mouseReleased(MouseEvent evt) {
    if (!isEnabled()) {
        return;
    }
    if (down) {
        down = false;
        repaint();
        fireAction();
    }
}
```

The "fireAction()" call fires the action event and increments the button counter. In that point, we need to pass the event to the recorder. Let's call the event "clicked" (it can be given any name of course). The code is modified as follows for the Bean object:

```
public void mouseReleased(MouseEvent evt) {
    if (!isEnabled()) {
        return;
    }
    if (down) {
        / repaint the button so it would be shown in the "released" state.
        // this is done regardless if the clicked event is triggered or not
        down = false;
        repaint();

        // route event for recording
        // routeEvent will return false if the event was vetoed by some
       // listener, in this case the event should not be fired, and the
        // change in the button state as a result of the event (that is
        // incrementing the counter) should not be performed
        // the routeEvent returns true if the event was approved (router
        // may veto the event)
        // the fireAction method will also increment the counter
        if(routingSupport == null || routingSupport.routeEvent("clicked"))
           fireAction();
    }
}
```

Note the checking of the return value of routeEvent. If routeEvent returns FALSE, it means that the recorder intercepted the event and wants to prevent it from happening. In that case, the Bean object should not trigger the event nor increment the counter.

Step 4: Implement injectRoutableValue

Now you need to implement the injectRoutableValue method in order to play back recordable events. Again, the RapidRoutingSupport class provides methods to analyze the received event. You can extract the event description and one of its arguments (if it has arguments).

In the case of Bean object, we need to check for one event type: the "clicked" event. Here is the code that checks for it:

```
/**
 * Inject a routable value. This should cause the receiver
 * to trigger a change in state, identical to that caused by
 * an event that generated the routable value.
 * A new routableValue event should not be fired.
 * @param value java.lang.Object
 */
public void injectRoutableValue (Object value)
{
        // extracts the event description
        String desc = RapidRoutingSupport.extractDescription(value);

        // handle the "clicked" event
           if (desc.equals("clicked"))
                  fireAction(); // simulate push button click
 }
```

That's it! The Bean object now fully supports one recordable event: "clicked." The button is fully usable for scenario recording, just like a regular Rapid button.

Step 5: Compare the Recordable and Nonrecordable JavaBean Objects

Several files are provided that demonstrate the differences between the recordable and nonrecordable JavaBean objects (located in the \\Examples\JavaBeans\RecordableEvents folder). The counterbuttons.jar file contains both the CounterButton and CounterButtonRecordable JavaBean objects, including source code and support classes (RapidAccessible and RapidRoutingSupport). These JavaBean objects are used in an example application, Recordable_Events.rpd. A sample document (created in the Document Manager) contains use cases recorded for the two JavaBean objects.

Note that when a JavaBean object does not support recordable events (does not implement the RapidAccessible interface), Rapid records *all* mouse and keyboard events related to the JavaBean object, so it will still work but the use case will be much larger (for example, the use case for the CounterButton JavaBean object has many more events because it does not support recordable events).