Chapter

14

Integrating BIRT Spreadsheet Engine with Java applications

This chapter contains the following topics:

- About BIRT Spreadsheet Engine and J2SE
- Writing an application class that extends JFrame
- Accessing the BIRT Spreadsheet API using JavaScript
- Using an add-in function

About BIRT Spreadsheet Engine and J2SE

The Actuate BIRT Spreadsheet Engine class library gives you spreadsheet functionality in a Java application or applet. This chapter describes how to:

- Write a Java Swing application that is also an applet.
- Access the BIRT Spreadsheet API using JavaScript.
- Deploy the license file in each deployment environment.

All of the example programs in this chapter use the BIRT Spreadsheet API. For more information about the BIRT Spreadsheet API, see the Javadoc.

Writing an application class that extends JFrame

If you do not want your Actuate BIRT Spreadsheet Engine application to double as an applet, you can write the application class to extend the Java Swing class JFrame. In the HelloWorldApp2 example that follows, the main( ) method instantiates an object of the application class and set its visible property to true. The constructor prepares itself by performing standard Java Swing tasks, including:

- Setting the layout method of the content pane
- Setting the frame's size and title
- Creating a WindowAdapter object and passing it to the addWindowListener( ) method

The HelloWorldApp2 class constructor then does a few operations specific to the BIRT Spreadsheet API, including:

- Instantiating a JBook object
- Adding the JBook object to the frame's content pane
- Creating a BookModel object from the JBook object
- Passing the BookModel object to the doSpreadsheetTasks( ) method that does all the spreadsheet-related tasks

```
import com.f1j.swing.engine.ss.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import com.f1j.ss.*;
```

```
public class HelloWorldApp2 extends JFrame {
   public HelloWorldApp2() {
      getContentPane().setLayout(null);
      setSize(450, 275);
      setTitle("Swing application");
      SimpleWindow sw_window = new SimpleWindow();
      addWindowListener(sw_window);
      // Create a JBook object, add it to the content pane
      JBook jb_jbook1 = new JBook();
      jb_jbook1.setBounds(10,5,400,200);
      getContentPane().add(jb_jbook1);

      // Create a BookModel object,
      // pass it to doSpreadsheetTasks
         BookModel bm_book = jb_jbook1.getBookModel();
      doSpreadsheetTasks(bm_book, new Object());
   }
   public static void main(String args[ ]) {
      (new HelloWorldApp2()).setVisible(true);
   }
   class SimpleWindow extends WindowAdapter {
      public void windowClosing(WindowEvent event) {
         Object object = event.getSource();
         if(object == HelloWorldApp2.this)
            SimpleApp_WindowClosing(event);
      }
   }
   void SimpleApp_WindowClosing(WindowEvent event) {
      setVisible(false);
      dispose();
      System.exit(0);
   }
   private void doSpreadsheetTasks(BookModel bm_book,Object obj) {
      try{
         bm_book.setText(1, 0, "Hello World");
      } catch(Exception e){}
   }
}
```

The JBook class implements the BookModel interface. Therefore, you can pass a JBook object to any method that takes a BookModel argument. Almost all of the spreadsheet-specific functionality in JBook is also in the BookModel interface.

The doSpreadsheetTasks( ) method in the HelloWorldApp2 example takes a BookModel argument and an Object argument. This means that doSpreadsheetTasks( ) has the same signature as both the start method and the end method of a callback class. For more information about callback classes, see *Designing Spreadsheets using BIRT Spreadsheet Designer.* Using the same signature for doSpreadsheetTasks( ) as the signatures of the start( ) and end( ) methods of a callback class is useful for the following two reasons:

- You can easily reuse your doSpreadsheetTasks( ) code in a callback class.
- There are numerous examples of callback class code that apply equally well to applications and applets.

When you compile and run the HelloWorldApp2 application, a window appears, similar to the one in Figure 14-1.

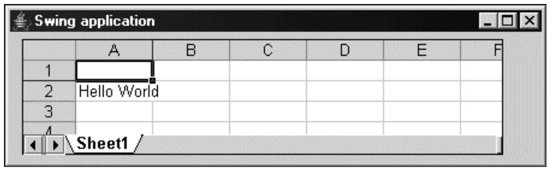

Figure 14-1 HelloWorldApp2 example application

Accessing the BIRT Spreadsheet API using JavaScript

You can access the BIRT Spreadsheet API using JavaScript. The JBookApplet class has a method, getJBookEx( ), that returns a JBook object. You can access the JBook object in a JavaScript script using code similar to the following line:

```
d_document.japp.getJBookEx()
```

In this snippet, japp is the name of the JBookApplet object, as assigned in the NAME attribute of the APPLET element. In the following example, the function startMe( ) runs every time the web page loads:

```
<HTML>
   <HEAD>
      <TITLE> Live Worksheet Page </TITLE>
      <SCRIPT>
         function startMe() {
            document.japp.getJBookEx().messageBox("Just press
            OK","Testing Message Box", 1);
         }
      </SCRIPT>
   </HEAD>
   <BODY onload="startMe()">
      <APPLET CODE="com.f1j.swing.engine.ss.JBookApplet"
            ARCHIVE="essd11.jar, derby.jar, license/"
            NAME="japp" WIDTH=500 HEIGHT=500>
         <PARAM name="Workbook" value="myWorkbook.xls">
      </APPLET>
   </BODY>
</HTML>
```

The JavaScript function startMe( ) in the previous example contains the following line of code that gets a JBook object and calls its messageBox( ) method:

```
d_document.japp.getJBookEx().messageBox("Just press OK",
   "Message Box", 1);
```

When you open this HTML file in a browser, it displays a message box containing the message, "Just press OK". You can access the entire BIRT Spreadsheet API in this way.

You can use this technique with any applet, including those you write yourself, as long as you include a method in your applet to return the JBook object. The following is a modification of the HelloWorld applet that includes a method to return the JBook object:

```
import java.awt.*;
import com.f1j.swing.engine.ss.*;

public class HelloWorld2 extends javax.swing.JApplet {
   JBook jb_jbook1 = new JBook();
   public void init() {
      getContentPane().setLayout(null);
      setSize(500, 500);
      jb_jbook1.setSize(500, 500);
      getContentPane().add(jb_jbook1);
      try {
         jb_jbook1.setText(1, 0, "Hello World");
      }
      catch (com.f1j.util.F1Exception e) { }
      setVisible(true);
   }
   public JBook getJBook() {
         return jb_jbook1;
   }
}
```

The following HTML code uses the added getJBook( ) method to access the BIRT Spreadsheet API and display a message box:

```
<HTML>
   <HEAD>
      <TITLE> Live Worksheet Page </TITLE>
         <SCRIPT>
            function startMe() {
               document.MyApplet.getJBook().messageBox(
               "Just press OK","Message Box",1);
            }
         </SCRIPT>
   </HEAD>
   <BODY onload="startMe()">
      <APPLET CODE="HelloWorld2.class" ARCHIVE="essd11.jar,
            derby.jar, license/"
            NAME="MyApplet" WIDTH=500 HEIGHT=500>
         <PARAM name="Workbook" value="401k.xls">
      </APPLET>
   </BODY>
</HTML>
```

When you open this HTML file in a browser, a web page appears, similar to the one in Figure 14-2.

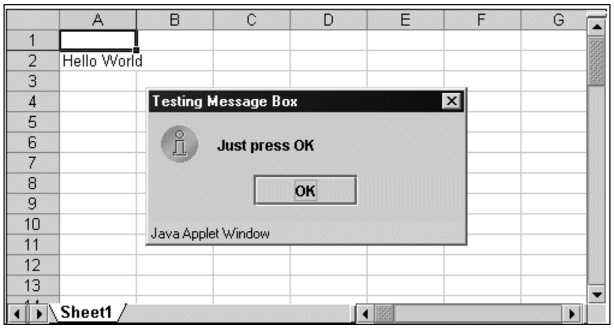

Figure 14-2 Web page containing HelloWorld2 applet

Using an add-in function

An add-in function creates a spreadsheet function, like SUM( ), that you can reference in the spreadsheet with an equals sign followed by the function definition, like =myFunction( ). An add-in function is supported for Actuate file formats, .sod and .soi, and in BookModel instances. Add-in functions are not supported in Excel. Loading an add-in function in Excel displays #FORMULA instead of the resulting calculated value.

To implement an add-in function in BIRT Spreadsheet Engine, create a class that extends com.f1j.addin.Func. The new class must:

- Extend the com.f1j.addin.Func class.
- Have the same case-sensitive name as the add-in function that the application uses to call it.
- Support only one instance of itself. To do this, the class must:
    - Have a private constructor.

      The constructor calls the superclass constructor and passes the name of the add-in function and the function's minimum and maximum number of parameter, as in the following snippet:

```
   private MyConcat() {
      super("MyConcat", 1, 30);
   }
```

   - Contain a static initializer such as:

```
   static {
      new MyConcat();
   }
```

- Override the evaluate( ) method of com.f1j.addin.Func. The evaluate( ) method provides the functionality of the add-in. Use the synchronized modifier with the evaluate( ) method to provide thread safety, as in the following snippet:

```
public synchronized void evaluate(
   com.f1j.addin.FuncContext fc_context)
```

- Be in the application's class path.
- Have essd11.jar, derby.jar, and the license file, eselicense.xml, in the application's class path.

Understanding the FuncContext object

The evaluate( ) method contains a FuncContext parameter, which provides access to the parameter values that the application passes to the add-in function. The FuncContext object has the following methods that the add-in can use to get the parameters:

- getArgumentCount( ) returns the number of parameters that the application passes to the add-in function.
- getArgument( ) takes a parameter number argument and returns the specified parameter as a com.f1j.addin.Value object. For more information on the Value object, see "Understanding the Value object," later in this chapter.
- setReturnValue( ) comes in five varieties, each variety having a parameter of a a different type. The five parameter types are:
   - Boolean
   - Double
   - Short
   - String
   - StringBuffer

Understanding the Value object

The Value object contains methods to evaluate and extract the value of a parameter that the application passes to the add-in function. The Value object contains the following kinds of methods:

- Methods to check the type of the parameter, including:
  - checkLogical( ), which checks whether the value is a Boolean
  - checkText( ), which checks whether the value is a text string
  - checkNumber( ), which checks whether the value is numeric

  All the preceding methods check for their specific type and attempt to convert the value to the specified type if it is not. All the preceding methods return false if the value is not and cannot be converted to the specified type.
- Methods to check the type of the value without attempting to convert it if it is not. These methods include:
  - isArea( ) tests if the value is a cell range.
  - isCell( ) tests if the value is a single cell reference.
  - isEmpty( ) tests if the parameter is empty.
  - isLogical( ) tests if the value is a Boolean values.
  - isNumber( ) tests if the value is a number.
  - isText( ) tests if the value is a text string.
  - isTrue( ) tests if the value is a Boolean true.
- Methods to get the parameter value. These methods include:
  - getCol1( ) gets the column of the beginning of a range if the value is a cell range.
  - getCol2( ) gets the column of the end of a range if the value is a cell range.
  - getRow1( ) gets the row of the beginning of a range if the value is a cell range.
  - getRow2( ) gets the row of the end of a range if the value is a cell range.
  - getColCount( ) gets the number of columns in the range if the value is a cell range.
  - getSheet( ) gets the sheet for a cell range.
  - getText( ) gets the value as a text string.
  - getText(StringBuffer) gets the value in a string buffer.

About an example of an add-in function

The following example demonstrates an add-in function that concatenates text:

```
public class MyConcat extends com.f1j.addin.Func {
// The following variable is shared
StringBuffer sb_accum = new StringBuffer();

static { //static initializer
   new MyConcat();
}

private MyConcat() { // private constructor
   super("MyConcat", 1, 30); // Specify 1-30 arguments
}

public synchronized void evaluate(com.f1j.addin.FuncContext fc) {
   sb_accum.setLength(0);
   int argCount = fc.getArgumentCount();
   for (int ii=0; ii < argCount; ii++) {
      com.f1j.addin.Value v_val = fc.getArgument(ii);
      if (v_val.checkText()) {
         m_accum.append(v_val.getText());
      }
      else {
         fc.setReturnValue(v_val.eValueInvalidValue);
         return;
      }
   }
   fc.setReturnValue(sb_accum.toString());
}
}
```

Making add-in functions determinant

Functions in BIRT Spreadsheet are either determinant or non-determinant. A determinant function, such as ABS(n), consistently returns the same value for a given parameter. BIRT Spreadsheet only recalculates a determinant function if the method has a parameter value that BIRT Spreadsheet has not encountered before. A non-determinant function, such as RAND( ), returns a value that varies with every execution. BIRT Spreadsheet always recalculates the return value for a non-determinant method.

Although BIRT Spreadsheet add-in functions are non-determinant by default, you can change this default behavior by calling the setDeterminant( ) method of the com.f1j.addin.Func object. After calling this method, the add-in function becomes determinant. Calling the setDeterminant( ) method can make a significant improvement in recalculation performance.