Copyright Information

Disclaimer of Warranties

The software and users manuals are provided "as is" and without express or limited warranty of any kind by either IISC or anyone who has been involved in the creation, production, or distribution of the software, including, but not limited to the implied warranties of the merchantability and fitness for a particular purpose. The entire risk as to quality and performance of the software and users manuals is with you. Should the software and users manuals prove defective, you (and IISC or anyone else who has been involved in the creation, production, or distribution of the software) assume the entire cost of all necessary servicing, repairs, or correction.

Some states do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

Limitation of Liability

In no event will IISC or any other person involved in the creation, production, or distribution of the software be liable to you on account of any claim for any damages, including any lost profits, lost savings, or other special, incidental, consequential, or exemplary damages, including but not limited to any damages assessed against or paid by you to any third party, arising out of the use, inability to use, quality, or performance of such software and users manuals, even if IISC or any other such person or entity has been advised of the possibility for such damages, or for any claim by any party. In addition, IISC or any other person involved in the creation, production, or distribution of the software shall not be libel for any claim by you or any other party for damages arising out of the use, inability to use, quality, or performance of such software and users manuals,

based upon principals of contract warranty, negligence, strict liability for the negligence of IISC or other tort, breach of any statutory duty, principals of indemnity or contribution, the failure of any remedy to achieve its essentials purpose, or otherwise.

Some states do not allow the limitation or exclusion of liability for incidental or consequential damages, so the above limitation may not apply to you.

TABLE OF CONTENTS

INTRODUCTION

Welcome to the *JSheet Programmer's Guide*. This Guide targets a variety of users and discusses how the JSheet applet can be used at several levels. If you are:

- familiar with HTML and want to include the JSheet applet in your web page, or
- an experienced JavaScript programmer who wants HTML controls to interact with the JSheet applet, or
- an experienced Java programmer who wants to develop a servlet or Java application using the JSheet Application Programming Interface (API),

this Guide contains information to help you.

About This Manual

This Guide contains this *Introduction*, three chapters, and an appendix.

Chapter 1, The JSheet Applet, explains how to include the JSheet applet in your web page. In addition to standard HTML attributes, the JSheet applet also can accept specialized parameter tags that are native to the applet and allow you to customize the default appearance and behavior of the JSheet applet.

Chapter 2, Using JavaScript with the Applet, explains how to use JavaScript to access the JSClient functionality, once the JSheet applet has been included in an HTML file.

Chapter 3, Using Java with JSheet, explains how to use the API to create applications that use the power of the JSServer engine. This chapter discusses how to connect to a JSheet server, display data, interact with the worksheet grid, respond to events, modify permissions, and manipulate charts.

The *Appendix, JavaScript Callback Functions*, contains a complete list and description of the JavaScript functions used to handle callbacks generated from the JSheet Server.

Before You Begin

Chapter 1 assumes that you have a basic understanding of HTML tags and attributes and know how to build a web page. *Chapter 2* assumes that you have a basic understanding of JavaScript and the document object model (DOM). *Chapter 3* assumes that you have a basic understanding of Java, are experienced as a Java developer, and that you have either a JDK installed, or a visual Integrated Development Environment (IDE), such as JBuilder by Borland or Visual Café by WebGain.

NOTE: If you are using the JDK with a text editor, you must include the **JSheet.jar** file in your **CLASSPATH** environment variable. If you are using a visual IDE, you should refer to your documentation to learn how to add it to your project. The examples in *Chapter 3* use the JDK/text editor option.

CHAPTER 1: THE JSHEET APPLET

The JSheet applet is displayed in a web browser by including an **<applet>** tag in a page of HTML. Similar to the **<img>** tag, the **<applet>** tag provides the browser with the source for the object (the applet) to be displayed, and specifies the dimensions of the object. When the JSheet applet is loaded, a spreadsheet grid and interface are displayed in the browser. The web page user interacts directly with a JSheet workbook that resides on the server. The following example shows an **<applet>** tag that is used to load the JSheet applet.

Example

```
<applet
      code="com.iisc.jwc.jsheet.JSClient"
      archive="jsheet.jar"
      width="600"
      height="400"
      mayscript="true"
      codebase="."
>
</applet>
```

The code and archive attributes tell the browser to load the Java applet contained in the class **com.iisc.jwc.jsheet.JSClient** within the Java archive (**.jar**) file named **JSheet.jar**. The **codebase** attribute tells the applet the

directory in which the **.jar** file resides. In the example above, the **.jar** file resides in the same directory as the HTML file.

The **width** and **height** attributes instruct the browser to display the applet in a rectangular area that is 600 pixels wide and 400 pixels tall. The **mayscript** attribute is a Netscape-only option that enables scripting (via JavaScript) of the applet from the HTML page.

In addition to standard HTML attributes, the **<applet>** tag also can accept specialized parameter tags that are native to the applet. These **<param>** tags allow you to customize the default appearance and behavior of the JSheet applet. The following example expands the earlier **<applet>** tag example to include **<param>** tags.

Example

```
<applet
      code="com.iisc.jwc.jsheet.JSClient"
      archive="jsheet.jar"
      width="600"
      height="400"
      mayscript="true"
      codebase="."
>
<param name="host" value="moondance">
<param name="bookname" value="pmtcalc.jss">
<param name="ShowFormulaBar" value="true">
</applet>
```

In the example above, parameter tags instruct the JSheet applet to connect to a host machine named **moondance** and open a workbook file named **pmtcalc.jss** from the server-side workspace of the user. The **ShowFormulaBar** parameter tag is set to **true** to display the optional formula bar in the applet.

<APPLET> Tag Attributes

The following attributes are required when constructing a JSheet **<applet>** tag.

Attribute	Description	Example
code	The name of the file containing the source code for the applet.	code="com.iisc.jwc.jsheet.JSClient"
archive	The name of the Java source code archive file to be pre-loaded.	archive="JSheet.jar"
width	The width, in pixels, of the applet.	width="600"
height	The height, in pixels, of the applet.	height="400"
mayscript	A Netscape-only flag enabling scripting (via JavaScript) of the applet instance.	mayscript="true"

The following attributes are optional for the **<applet>** tag.

Attribute	Description	Example
codebase	The path (relative or absolute) to the file containing the source code (.**jar** files) for the applet. Relative paths start from the base URL default of the document, "."	codebase="../../"
name	The name for a specific instance of the applet. The default is **none**.	name="JSheetApplet"
alt	Text to be displayed if the browser understands the **<applet>** tag but cannot run Java applets. The default is **none**.	alt="Payment Calculator"
align	The alignment of the applet: **left, right, top, texttop, middle absmiddle, baseline, bottom, absbottom**. The default is **bottom**.	align="left"
hspace	The pixel offset on either side of the applet. The default is **0**.	hspace="5"
vspace	The pixel offset above and below the applet. The default is **0**.	vspace="10"

JSheet Applet Tag Parameters

The JSheet applet can be changed dynamically through the use of parameter tags. The table below describes each of the available parameter tags.

Parameter	Default	Description
background	Oxffffff (white)	The background color to be used for the visible frame area. The user provides the standard HTML-style RGB reference.
bookName	none	The name of the workbook to create or open on launch.
bookPassword	none	The password for the workbook that is created or opened on launch. If no password is specified, it is assumed the workbook does not have a password.
connect	true	Attempts to connect to the server on startup.
hideExceptions	false	Disables exception message windows.
hideInfoDialogs	false	If set to true, dialog boxes are displayed when receiving error messages; otherwise, an error is displayed in the status bar and the window.
hidePopupMenu	false	Displays a popup menu when the user right-clicks over the applet.
host	current host	Sets the host name to which to connect. If the value is blank, attempts to connect to the web server's host machine. If that is not available, a dialog box is displayed, allowing the user to specify a name.
openMode	none	The mode in which you want an existing book to be opened.
openNewBook	false	Creates a new workbook if set to **true**, or opens an existing workbook if set to **false**. If the **bookname** parameter is set, the name specified by that parameter is used for the new book name.

password	none	The user password to use when connecting to the host machine.
publicRead	true	Read permissions are granted for all new workbooks created on the client.
publicSave	true	Save permissions are granted for all new workbooks created on the client.
publicWrite	true	Write permissions are granted for all new workbooks created on the client.
saveable	true	Eliminates the prompt that asks if you want to save.
sheetName	none	The name of the sheet within the workbook on which the chart is located. By default the sheets are labeled **Sheet1, Sheet2, Sheet3**, and so forth.
showFormatBar	true	Displays the format bar, to allow for the easy formatting of fonts, styles, colors, and alignments for the data entered into the worksheet.
showFormulaBar	true	Displays the formula bar, to allow for the easy creation or editing of cell formulas and named ranges.
showStatusBar	true	Displays the status bar at the bottom of the applet.
templateName	none	The name of the template that you want to use for the new workbook.
throwExceptions	true	Enables throwing exceptions.
tracking	none	Sets the tracking method you want to use for updating cells.
trackingDown		The selected color fills the cell when the cell value decreases.
trackingDuration	-1	The number of seconds to display the celltracking changes.
trackingUp		The selected color fills the cell when the cell value increases.
user	none	The user name to use when connecting to the host machine.

Hiding the JSheet Applet

There may be times when it is preferable to hide the applet. To hide the applet, specify **1** as the value for both the **width** and **height** attributes. Alternatively, you can use the **hidegridcomponents** applet parameter to hide the applet. The code in the following example hides the applet.

Example

```
<applet
      code="com.iisc.jwc.jsheet.JSClient"
      archive="JSheet.jar"
      width="1"
      height="1"
      mayscript="true"
      codebase="."
>
<param name="hidegridcomponets" value="true">
</applet>
```

IMPORTANT: Do not attempt to create an applet of zero width and zero height. When using Netscape, the resulting behavior is unpredictable.

The JSChart Applet

There are two main access points to the charting facilities of JSheet. Via the JSheet interface, you can select **View/Charts** from the menu to display snapshot images of a chart. Or, you can use the JSChart applet to display and manipulate charts.

A chart is displayed in a web browser by including a JSChart **<applet>** tag in a page of HTML. The following example shows a simplified JSChart applet element.

Example

```
<html>
<body>

    <applet
        name="jschart"
        code="com.iisc.jwc.jschart.JSChart"
        archive="JSchart.jar"
        width="400"
        height="600"

            <param name="host" value="zeus">
            <param name="user" value="johndoe">
            <param name="password" value="test">
            <param name="bookname" value="/simple.jss">
            <param name="sheetname" value="sheet1">
            <param name="chartname" value="Chart1">
    </applet>

</body>
</html>
```

JSChart <applet> Tag Parameters

The JSChart applet can be changed dynamically through the use of parameter tags. The table below describes each of the available parameter tags.

Parameter	Default	Description
background	Oxffffff (white)	The background color to be used for the visible frame area. The user provides the standard HTML-style RGB reference.
bookname	none	The name of the book that contains the chart.

bookpassword	none	The password for the book that contains the chart. If no password is specified, it is assumed the book does not have a password.
chartname	none	The name of the chart to display. If a user-defined name is not specified, charts are named **Chart1, Chart2, Chart3**, and so forth by default.
host	current host	Sets the host name to which to connect. If the value is blank, attempts to connect to the web server's host machine. If that is not available, a dialog box is displayed, allowing the user to specify a name.
password	none	The user password to use when connecting to the host machine.
sheetname	none	The name of the sheet within the workbook on which the chart is located. By default, the sheets are labeled **Sheet1, Sheet2, Sheet3**, and so forth.
smartfit	true	If the frame area provided is smaller than the image, when **smartfit** is set to **true**, the image is scaled down to fit the frame. However, the image is not stretched to fit a frame area that is larger than the image. When **smartfit** is set to **false**, the image is stretched or scaled down to fit the frame, regardless of the size of the frame or the image.
throttle	5 seconds	The period of time, in seconds, that JSChart throttles back update requests. If a value of **0** is entered, the chart never updates.
user	none	The user name to use when connecting to the host machine.

CHAPTER 2: USING JAVASCRIPT WITH THE APPLET

This chapter explains how to construct the JSheet **<applet>** tag, the various parameters that can be used with the **<applet>** tag, and how to work with the applet in JavaScript. A basic understanding of JavaScript and the document object model (DOM) is assumed.

Once the **<applet>** tag has been included in an HTML file, the JSClient functionality can be accessed using JavaScript in an HTML page. Specifically, calls are made to JSClient methods through the applet. While not all browsers support making calls to Java methods from JavaScript, this functionality is supported in both Internet Explorer and Netscape Navigator.

The Internet Explorer and Netscape Navigator browsers provide built-in software that allows JavaScript to interact with Java applets. The basic procedure is to:

- insert an applet into an HTML page using an **<applet>** tag.
- use the DOM to extract the applet object and assign the applet object to a JavaScript variable.
- use the following JavaScript syntax to access any of the Java methods of the applet:

```
<applet variable name>.<applet method>
```

The following HTML example demonstrates this procedure.

Example

```
<applet name="JSApplet" . . . >
<param value="bookname" value="testWorkbook.jss">
</applet>

<script language="JavaScript">
var myApplet = document applets["JSApplet"];
alert ("Number of sheet in workbook: " + myApplet.getSheetCount());
</script>
```

In the above example, **getSheetCount** is a method that retrieves the number of sheets in the currently open workbook. When the HTML page loads, a message is displayed that describes the number of sheets in the currently open workbook.

getSheetCount is a Java method found in the **JSClient** class. **JSClient** is the Java class that corresponds to the JavaScript **myApplet** variable. (For more information on using JSClient methods from within JavaScript, see the section *Using Applet APIs*, later in this chapter.)

A JavaScript Example

A complete JavaScript example is provided below. At this point, you can simply skim the code; explanations that reference the code are presented in the sections that follow.

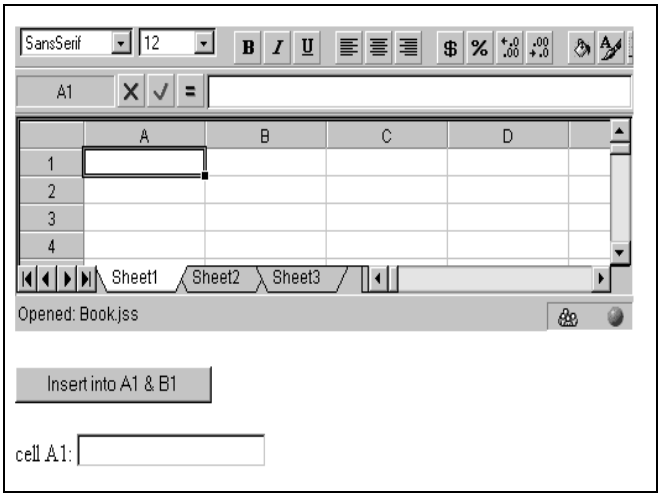
Example

```
<html>
<head>
<title>JavaScript Example</title>
<script src="../Client/jsmllib.js"></script>
<script language="JavaScript">
<!--
function doIt()
{
	var myApplet = document.applets["JSApplet"];
	var cell = javaCell(myApplet, 0, 1, 1);
	myApplet.setCellEntry(cell, "Hello");
	cell.col = 2;
	myApplet.setCell Entry(cell, "World! ");
	cell.col = 1;
	alert(myApplet.getCellEntry(cell));
}
//-->
</script>
</head>

<body>
<applet
	name="JSApplet"
	codebase="../Client"
	code="com.iisc.jwc.jsheet.JSClient"
	archive="JSheet.jar"
	mayscript="true"
	width="500" height="200">
	<param name="OpenNewBook" value="true">
	<param name="user" value="demo">
	<param name="password" value="demo">
</applet>

<form>
<input type="button" value="Insert into A1 & B1" onclick="doIt()">
<br><br>
```

```
<script language="JavaScript">jsRegList.setDefaults("","JSApplet", "book.jss", 0);
</script>
cell A1:
<input type="text" name="myTextControl" onchange="jsVerify(this)">
<script>
jsRegList.add("JSML_TEXT", 0, new Cell(1,1), "myTextControl", "", "", TRUE,
TRUE, "", "");
</script>
</form>

</body>
</html>
```

The HTML code in the example above renders as shown in the following illustration.

Figure 2-1



When you click the **Insert into A1 & B1** button, **Hello** appears in cell A1 and **World!** appears in cell B1. **Hello** is displayed in the text box and a pop-up dialog that contains **Hello** is also displayed. If you modify the text box and then click elsewhere, the modified value appears in cell A1.

The Button Control

In the previous code example, the **<applet>** tag with an **OpenNewBook** parameter causes a new JSheet grid to appear. When the **Insert into A1 & B1** button is clicked, the **doIt()** function is called. In the **doIt()** function, a JavaScript-accessible version of the applet is put into the **myApplet** variable when the following line is executed:

```
var myApplet = document.applets["JSApplet"];
```

The following line uses the **myApplet** variable to call the **setCellEntry** method of JSClient:

```
myApplet.setCellEntry(cell, "Hello");
```

NOTE: JavaScript can access the **setCellEntry** method only because it is defined as **public**. View javadoc for the **JSClient** class to verify that the method is public.

The **setCellEntry** method expects a JSheet Cell object as its first parameter. The following line creates a JSheet Cell object:

```
var cell = javaCell(myApplet, 0, 1, 1);
```

javaCell(myApplet, 0, 1, 1) is a call to the **javaCell** JavaScript constructor. It receives four parameters:

- **applet**. The **myApplet** variable in this example.
- **sheetindex**. The sheet from within which the cell comes. **0** indicates the first sheet.
- **row**. **1** indicates the first row in the sheet.
- **column**. **1** indicates the first column in the sheet.

The **javaCell** constructor returns a JSheet Cell object. Since the cell variable describes a row 1, column 1 cell in the first sheet, the first call to **setCellEntry** puts the **Hello** string in cell A1 in the first sheet.

The **javaCell** constructor is defined in the **jsmllib.js** file. To access anything from **jsmllib.js**, you must include that file in your HTML page, using the following line:

```
<script src="../Client/jsmllib.js"></script>
```

The **"../Client/"** path indicates where the **jsmllib.js** file is stored in relation to the HTML page. For this example, the **jsmllib.js** file is found one directory level above the HTML file, and then down one directory level to the **Client** directory.

Since the **javaCell** constructor returns a copy of a Java JSheet Cell object, the cell variable can be used to access any of the public methods or fields of the **Cell** class. Since the Cell column field is public, the column value of the cell variable is changed to **2** with the following line:

```
Cell.col = 2;
```

NOTE: You can view Javadoc for the **Cell** class to verify that the column field of Cell is public.

Since the cell variable now describes a row 1, column 2 cell in the first sheet, the following call to **setCellEntry** puts the **World!** string in cell B1 of the first sheet:

```
applet.setCellEntry(cell, "World!");
```

The final two lines in the **doIt()** function change the column value of the cell back to **1** and then print the value of the cell with a pop-up dialog:

```
cell.col = 1;
alert(applet.getCellEntry(cell));
```

The Text Box Control

The text box control illustrates bi-directional interaction with the applet. Changes in the value of cell A1 cause the text box value to update; changes in the text box value cause the value of cell A1 to update.

The **jsmllib.js** library contains functions that help with the implementation of control/applet interaction: **jsRegList.setDefaults(), jsVerify()**, and **jsRegList.add()**. Call the **jsRegList.setDefaults** function one time. That call should appear above all of the controls that call the **jsVerify** and **jsRegList.add** functions.

Example

```
jsRegList.setDefaults("","JSApplet", "book.jss", 0);
```

jsRegList.setDefaults receives the following four parameters:

- **appletFrame**. The name of the frame that contains the applet. Use "" for single-frame applications.
- **applet**. The name of the applet as it appears in the **<applet>** tag.
- **workbook**. The name of the workbook file. **book.jss** is the default name for new workbooks.
- **sheetIndex**. Indicates from which sheet the cell comes. **0** indicates the first sheet.

To cause a change in the text control to affect the same change in the associated applet cell, call the **jsVerify** function. For example:

```
<input type="TEXT" name="myText Control" onchange="jsVerify(this) ">
```

jsVerify receives a control object as its single parameter. In this example, **this** refers to the enclosing text box control.

Call the **jsRegList** function, as follows, to set up an association between a particular applet cell and a particular control:

```
jsRegList.add("", 0, new Cell(1,1), "myTextControl", "", "", true, true, "", "");
```

jsRegList.add receives the following ten parameters:

- type. One of the JSML control types. (In this example, the empty string, **""**, is used.)
- sheetIndex. Indicates from which sheet the associated cell comes. **0** indicates the first sheet.

- cell. Indicates the applet cell with which the control is associated. The **Cell** constructor is a constructor in **jsmllib.js** that receives row and column parameters.
- elementName. The name of the control as specified in the **name** attribute of the HTML control tag.
- displayFunction. The name of a JavaScript function that can be used to modify the text before it is displayed in the control.
- verifyFunction**.** The name of a JavaScript function that can be used to verify the text entered into the control before it is placed in the spreadsheet.
- readFlag. If **true**, update the control each time the applet cell changes.
- writeFlag. If **true**, update the applet cell each time the control changes.
- displaySheet. The sheet that contains the applet cell; defaults to the sheet specified in the call to **jsRegList.setDefaults()**.
- displayRange. The range that contains data to be displayed. This parameter is used only for certain JSML control types (**""** in this example).

Using Applet APIs

You can access any of the public classes in your installed **JSheet.jar** file by using JavaScript in an HTML page. Specifically, you can access the public methods and fields within the public classes. These methods and fields are referred to as the application programming interface (API).

The class that you will access the most often is the **JSClient** class, which is instantiated as the JSheet applet. The JSheet applet and its public methods and fields are accessed in an HTML page, as detailed above.

Many JSClient methods require JSheet objects as parameters (e.g., **applet.copy(range)**). The **JSClient.copy** method receives a range object parameter. To create a JavaScript version of a Java object, call the **createObject()** method of JSClient. The following example creates a JSheet range object.

Example

```
range = applet.createObject("com.iisc.jwc.jsheet.Range");
```

The **createObject** method receives the full name of the target class. (To find full class names, refer to the javadoc.) After creating the range object as shown above, you must initialize it.

NOTE: Because it is common to need range and cell objects in your JavaScript code, the **javaRange** and **javaCell** constructors are included in the **jsmllib.js** library. They take care of the call to **applet.createObject** and the subsequent initialization. (Refer to the previous HTML code for an example of a call to **javaCell**.)

Use the following line to create and initialize a JSheet Range object:

```
range = javaRange(myApplet, 0, 1, 1, 3, 3);
```

javaRange(myApplet, 0, 1, 1, 3, 3) is a call to the javaRange constructor. It receives the following six parameters:

- **applet**. The **myApplet** variable in this example.
- **sheetIndex**. Indicates from which sheet the cell comes. **0** indicates the first sheet.
- **row**. **1** indicates the first row.
- **column**. **1** indicates column A.
- **bottom**. **3** indicates the third row.
- **right**. **3** indicates column C.

After creating range and cell objects using **javaRange** and **javaCell**, you can test whether the cell of the Cell object is within the range of the Range object by calling the public contains method of the range. For example:

```
if (range.contains(cell))
{
   alert("the cell is outside of the given range");
}
```

Determining the Browser Type

Unfortunately, certain code is browser-dependent in terms of how it is rendered. If you need to use browser-dependent code, you may want your code to determine the current browser type and then act accordingly. If you include the **jsmllib.js** in your HTML page, you can access the following three boolean flag variables:

IslE4orHigher isNS4 isNS5orHigher

In rendering your page and loading **jsmllib.js**, the code detects the browser type and then sets the appropriate boolean flag to true and the other two flags to false.

Array/Vector Conversion

Arrays cannot be passed between JavaScript and Java. Therefore, from JavaScript, you should not call JSClient methods that receive an array parameter or return an array. Since vectors can be passed between JavaScript and Java, you should call a parallel method that uses a vector instead of an array.

In the following examples, assume **applet** and **range** are correctly instantiated JavaScript objects, **valuesArray** is an array of values, and **valuesVector** is a vector object of values.

Examples:

Will Not Work	Will Work
applet.getNamedRanges()	applet.getNamedRangesAsVector()
applet.setRangeValuesUsingArray(range, valuesArray)	applet.setRangeValuesUsingVector(range, valuesVector)

Unfortunately, JavaScript does not have a native vector object. Thus, after receiving a Java Vector object (from a call to **getNamedRangesAsVector**), you may want to convert the vector to an array. That functionality is implemented by the **vectorToArray** function of **jsmllib.js**. To call a method that has a vector argument, you can start with an array, convert it to a vector, and then pass the vector. That functionality is implemented by the **arrayToVector** function of **jsmllib.js**.

The **vectorToArray** method receives the following two parameters:

- **rows**. The number of rows in the array.
- **vector**. A Java Vector object.

The **arrayToVector** method receives the following two parameters:

- **applet**. A JSClient object.
- **array**. A JavaScript array.

Callback Functions

If you need to handle JSheet events, include callback functions in your HTML page. Examples of typical JSheet events are **bookOpened, bookClosed, cellValueUpdated,** and **exceptionThrown.** For a complete list

and description of JSheet callback functions, see *Appendix, JavaScript Callback Functions*.

When a callback event occurs, the callback method associated with the event is called in the JSheet applet. That callback method then attempts to call the JavaScript version of the callback. Depending on whether there is a need to handle a particular event, the JavaScript developer can include or omit a JavaScript version of the callback associated with the event. As with all JavaScript functions, callback functions should be placed in the head section of the HTML page.

The following HTML code implements the **bookOpened** callback function. After the **bookOpened** event is fired, the **bookOpened** method in the JSheet applet is called. That method then finds and calls the following **bookOpened** JavaScript function. For a description of the **bookName** parameter of the **bookOpened** function, as well as descriptions of all of the callback function parameters, refer to the Appendix.

Example

```
<html>
<head>
<script language="JavaScript">
<!--
      function bookOpened(bookName)
      {
            alert("You have opened " + bookName);
      }
//-->
</script>
</head>
. . .
```

Chapter 2: Using JavaScript with the Applet

CHAPTER 3: USING JAVA WITH JSHEET

As discussed in the previous chapters, the JSheet Client (**JSheet.jar**) can be run as an applet within a web page, with or without JavaScript calls to the applet. This chapter focuses on how JSheet Client can be run as a standalone Java application, or as an application programming interface (API) for custom application development. Using the API, the Java developer can access and manipulate the JSheet applet and create custom applications using the power of the JSheet server.

To run JSheet as a standalone application, make certain you are in the directory in which **jsheet.jar** resides and execute the following command from the command line:

```
java -classpath JSheet.jar com.iisc.jwc.jsheet.JSClient
```

Getting Started with JSheet

JSClient is the main class of the JSClient API and serves as the interface to the JSheet server. **JSClient** must be instantiated (as shown in the following code) before it is used within your application.

Example

```
import com.iisc.jwc.jsheet.jsClient;
public class JSheetSample1
{
	private JSClient jsClient;
	class MySessionAdapter extends SessionAdapter
	{
		public void bookOpened(SessionEvent e)
		{
			System.out.println("A new book was opened.");
		}
	}
	public JSheetSample1()
	{
		jsClient = new JSClient();
		jsClient.setHostName("localhost");
		jsClient.setUserName("demo");
		jsClient.setUserPassword("demo");
		jsClient.setOpenNewBook(true);
		MySessionAdapter sessionAdapter = new MySessionAdapter();
		jsClient.addSessionListener(sessionAdapter);
		jsClient.start();
		}
		catch(Exception e)
		{
			e.printStackTrace();
		}
	}
	public static void main(String args[ ])
	{
		JSheetSample1 jsheetSample = new JSheetSample1();
	}
}
```

Once a JSClient instance is established, you can connect to JSheet server. Note that the code uses the API to set the connection information. Alternatively, the user will be prompted for a host name, user name, and password.

Additionally, the code includes the **setOpenNewBook()** method so that a new workbook is opened as part of the connection activity. The **setOpenNewBook()** method takes one boolean parameter and instructs JSheet server whether to open a new workbook on startup, or to wait for the user to select **Open** from the JSheet menu. In the example above, the parameter is set to **true** and the new book is opened.

The JSheet Client API provides a series of event listeners to help identify actions performed with the JSClient object. To minimize the coding effort, adapters have been provided for these event listeners. In this example, the **sessionadapter** is used. Its methods are described in the table below.

Method	Description
killClient()	Invoked when the server sends a terminate message.
bookOpened()	Invoked when this client opens a book.
bookClosed()	Invoked when this client closes a book.
messageReceived()	Invoked when the server sends a message to this client.

As shown in the previous code example, the **bookOpened()** method was used to determine if and when a new book was opened. Also, since the listener must be added to the **jsClient** object, the method **addSessionListener()** was called.

When the code is executed, the text message, “A new book was opened.” is displayed.

Using JSheet in a Window

In many circumstances, you may prefer to see a worksheet grid when you use JSheet. Since a frame is used as the display vehicle for the worksheet grid, the JSheet user interface must be added to the frame using the **this.add()** method.

Example

```
import java.awt.*;
import java.awt.event.*;
import com.iisc.jwc.jsheet.*;

public class JSheetSample2 extends Frame
{
	private JSClient jsClient;
	public JSheetSample2()
	{
		jsClient = new JSClient();
		jsClient.setHostName("localhost");
		jsClient.setUserName("demo");
		jsClient.setUserPassword("demo");
		jsClient.setOpenNewBook(true);
		jsClient.start();
		this.setSize(800, 300);
		this.add(jsClient, BorderLayout.CENTER);

		//Provide the ability to shut down the application window.
		this.addWindowListener(
			new WindowAdapter()
			{
				public void windowClosing(WindowEvent e)
				{
					System.exit(0);
				}
			}
		);
```

```
    }
    public static void main(String args[ ])
    {
        JSheetSample2 jsheetSample = new JSheetSample2();
        jsheetSample.show();
    }
}
```

When you execute the code shown in the example above, an application window is displayed with a formula bar, format bar, worksheet grid, and status bar. (Refer to the *JSheet User Guide* for information on how to identify and use the components and features of the JSheet user interface.)

Interacting with Java Components

This section explains how Java controls can be made to interact with the JSheet API. In the following example, a button is created that interacts with the JSheet API. An explanation follows the code.

Example

```
import java.awt.*;
import java.awt.event.*;
import com.iisc.jwc.jsheet.*;

public class JSheetSample3 extends Frame
{
    private JSClient jsClient;
    private Button button;
    private Panel upperPanel;       //holds the jsClient applet
    private Panel midPanel;                 //holds the button

    class ButtonListener implements ActionListener
    {
```

```
        public void actionPerformed(ActionEvent e)
        {
                try
                {
                        Cell a1 = new Cell(1,1);
                        if (jsclient.getCellEntry(a1).equals(“ “))
                                jsclient.setCellEntry(a1, “Hello”);
                        else
                                jsclient.setCellEntry(a1, “ “);
                }
                catch (JSException jsException)
                {
                        jsclient.errorMsgForException(jsException);
                }
        }
}

public JSheetSample3()
{
        jsClient = new JSClient();
        jsClient.setHostName("localhost");
        jsClient.setUserName("demo");
        jsClient.setUserPassword("demo");
        jsClient.setOpenNewBook(true);
        jsClient.start();
        this.setSize(800, 300);

        // Insert applet into a panel.
        // Insert panel into top of frame.
        upperPanel = new Panel();
        upperPanel.setLayout(new BorderLayout());
        upperPanel.add(jsclient, “Center”);
        this.add(upperPanel, BorderLayout.CENTER);

        button = new Button(“Insert/delete ‘Hello’ in cell A1”);
        button.addActionListener(new ButtonListener());
```

```
        // Insert button into a panel.
        // Insert panel into bottom of frame.
        midPanel = newPanel()'
        midPanel.add(button);
        this.add(midPanel, BorderLayout.SOUTH);

        // Provide the ability to shut down the application window.
        this.addWindowListener(
            new WindowAdapter()
            {
                    public void windowClosing(WindowEvent e)
                    {
                        System.exit(0);
                    }
            }
        );
    }

    public static void main(String args[ ])
    {
        JSheetSample3  jsheetSample = new JSheetSample3();
        jsheetSample.show();
    }
}
```

In the above code, note the instantiation of the button component in the **JSheetSample3** constructor. A listener is added to the button component so that when the button is clicked, the **actionPerformed** event handler method of the listener is automatically called.

Refer to the **actionPerformed** event handler method of the **ButtonListener**. The event handler contains code that allows the user to manipulate the value displayed in the A1 cell of the grid. Specifically, when the user presses the button, the value in cell A1 toggles between displaying

"Hello" and displaying nothing. To enable this toggling functionality, the event handler first creates an instance of the cell A1. It does this by calling the JSheet Cell constructor with **row=1** and **column=1** for its two argument values. The event handler than retrieves the contents of cell A1 by passing the newly instantiated A1 cell object (called **a1** in the above code) into the **getCellEntry** method of the jsClient object. Depending on the current contents of cell A1, the value "**Hello**" or " " is assigned to the cell. To assign a value to cell A1, the event handler passes the A1 cell object and the assigned value into the **setCellEntry** method of the JsClient object.

APPENDIX: JAVASCRIPT CALLBACK FUNCTIONS

This Appendix presents the JavaScript callback functions in alphabetical order. A brief explanation of each function is provided. In addition, the parameters required for the functions are identified and defined.

activeCellChanged(cell)

This method is called when a different cell is selected in the grid.

JavaScript function parameters:

cell - a Cell object for the newly selected cell. For Cell details, see the javadoc for the JSheet Cell class.

activeSheetChanged(sheetIndex)

This method is called when a different sheet is selected in a workbook.

JavaScript function parameters:

sheetIndex - the index for the newly selected sheet (0-based indexing)

Appendix: JavaScript Callback Functions

bookClosed(bookName)

For a given client, this method is called when that client closes its own workbook.

JavaScript function parameters:

bookName - name of the closed workbook

bookOpened(bookName)

For a given client, this method is called when that client opens a workbook.

JavaScript function parameters:

bookName - name of the opened workbook

bookSaved(userName)

This method is called when a shared workbook is saved.

JavaScript function parameters:

userName - the name of the logged-on user for the client who performed the save operation

cellStyleUpdated(range)

This method is called after the style of a cell is updated.

JavaScript function parameters:

range - a Range object for the cell specified in the **CellStyleUpdated** method call. The **range** consists of one cell. (For details on Range, see javadoc for the JSheet Range class.)

cellValueUpdated(cell, value, permissions)

This method is called when a value changes in a shared workbook.

JavaScript function parameters:

cell - a Cell object for the cell that was updated. (For Cell details, see javadoc for the JSheet Cell class.)
value - the new contents of the cell that was updated
permissions - a bitmapped integer containing read/write permissions for the updated cell.

The **permissions** parameter has the following non-zero bits:

read bit = the first bit:
if (permissions & 1) != 0, then the cell can be read
write bit = the second bit:
if (permissions & 2) != 0, then the cell can be written

clientJoined(userName)

For a given client, this method is called when a different client loads in shared mode a workbook of that client.

JavaScript function parameters:

userName – the name of the logged-on user for the client that just loaded a copy of the shared workbook

clientLeft(userName)

This method is called when multiple clients are sharing a workbook and one of the clients closes a copy of the workbook.

JavaScript function parameters:

userName – the name of the logged-on user for the client that just closed a copy of the shared workbook.

colWidthUpdated(sheetIndex, leftColIndex, rightColIndex, widths)

This method is called after the column width is adjusted in a shared worksheet.

JavaScript function parameters:

sheetIndex - the index for the sheet in which columns were changed (0-based indexing)

leftColIndex - the index for the left column in the group of columns with widths that were adjusted (1-based indexing; column A has index 1)

rightColIndex - the index for the right column in the group of columns with widths that were adjusted

widths - array of adjusted column widths; width values are in pixels

deletedLeft(range)

This method is called after a **JSClient.deleteLeft** call has been processed.

JavaScript function parameters:

range - a Range object for the range specified in the **deleteLeft** method call. (For details on Range, see javadoc for the JSheet Range class.)

deletedUp(range)

This method is called after a **JSClient.deleteUp** call has been processed.

JavaScript function parameters:

range - a Range object for the range specified in the **deleteUp** method call. (For details on Range, see javadoc for the JSheet Range class.)

doubleClicked(cell)

This method is called when a grid cell is double-clicked.

JavaScript function parameters:

cell - A Cell object for the cell that was double-clicked. For Cell details, see the javadoc for the JSheet Cell class.

exceptionThrown(errorText)

This method is called when a server-generated exception is thrown.

JavaScript function parameters:

errorText - the error message for the exception

exceptionThrown(errorNumber, errorText)

This method is called when a server-generated exception has been thrown.

JavaScript function parameters:

errorNumber - the error number for the exception
errorText - the error message for the exception

forceBookClosed()

This method is called when the server closes a shared workbook.

JavaScript function parameters: none

insertedDown(range)

This method is called after a **JSClient.insertDown** call has been processed.

JavaScript function parameters:

range - a Range object for the range specified in the **insertDown** method call. (For details on Range, see javadoc for the JSheet Range class.)

insertedRight(range)

This method is called after a **JSClient.insertRight** call has been processed.

JavaScript function parameters:

range - a Range object for the range specified in the **insertRight** method call. (For details on Range, see javadoc for the JSheet Range class.)

killClient(shutdownMode)

For a given client, this method is called when the server kills that client.

JavaScript function parameters:

shutdownMode - **true** indicates the server performed an immediate shutdown; **false** indicates the server performed a graceful shutdown

leftColChanged(sheetIndex, leftColIndex)

This method is called after a workbook is horizontally scrolled so that there is a new left column.

JavaScript function parameters:

sheetIndex - the index for the sheet with columns that were changed (0-based indexing)

leftColIndex - the index for the new left column (0-based indexing; column 1 is indicated by 0)

messageReceived(sender, message)

For a given client, this method is called when the server or a different client sends a message to that client.

JavaScript function parameters:

sender - user name of the message sender
message - the sent message

rangeStyleUpdated(range)

This method is called after the style of a range is updated.

JavaScript function parameters:

range - a Range object for the range specified in the **rangeStyleUpdated** method call. (For details on Range, see javadoc for the JSheet Range class.)

rowHeightUpdated(sheetIndex, topRowIndex, bottomRowIndex, heights)

This method is called after the height is adjusted for a row or rows.

JavaScript function parameters:

sheetIndex - the index for the sheet in which rows were changed (0-based indexing)
topRowIndex - the index for the top row in the group of rows with heights that were adjusted (1-based indexing; row 1 has index 1)

bottomRowIndex - the index for the bottom row in the group of rows with heights that were adjusted
heights - the array of adjusted row heights; height values are in pixels

sheetInserted(sheetName, sheetIndex)

This method is called when a sheet is added to a shared workbook.

JavaScript function parameters:

sheetName – the name for the newly inserted sheet
sheetIndex – the index for the newly selected sheet (0-based indexing)

sheetMoved(fromSheetIndex, toSheetIndex)

This method is called when a sheet is repositioned in a shared workbook.

JavaScript function parameters:

fromSheetIndex - index for the original position of the moved sheet (0-based indexing)
toSheetIndex - index for the new position of the moved sheet (0-based indexing)

sheetNameChanged(sheetName, sheetIndex)

This method is called when a sheet name changes in a shared workbook.

JavaScript function parameters:

sheetName - new name for the sheet
sheetIndex - index for the sheet whose name changed (0-based indexing)

sheetPropertiesUpdated()

This method is called after the properties of a shared worksheet are adjusted.

JavaScript function parameters: none

sheetRemoved(sheetIndex)

This method is called when a sheet is removed from a shared workbook.

JavaScript function parameters:

sheetIndex – the index for the sheet that was removed (0-based indexing)

styleIndexCleared()

This method is called after a formatting style has been cleared.

JavaScript function parameters: none

textModified(cell, entryValue)

This method is called when a key is pressed while a grid cell is in edit mode. A grid cell is put into edit mode when the user double-clicks inside the grid cell.

JavaScript function parameters:

cell - a Cell object for the cell that was updated. For Cell details, see the javadoc for the JSheet Cell class.

entryValue - the current content of the cell

topRowChanged(sheetIndex, topRowIndex)

This method is called after a workbook is vertically scrolled so that there is a new top row.

JavaScript function parameters:

sheetIndex - the index for the sheet with columns that were changed (0-based indexing)

topRowIndex - the index for the new top row (0-based indexing; row 1 is indicated by 0)

Appendix: JavaScript Callback Functions

INDEX