Technical Guide

Formula One™ for Java®

Robust development tool for Java developers and Webmasters.

Version 7.0

Tidestone Technologies, Inc.

Published by
Tidestone Technologies, Inc.
12980 Metcalf Avenue, Suite 300
Overland Park, Kansas 66213
phone 913-851-2200
toll-free 1-800-884-8665
fax 913-851-1390

www.tidestone.com
www.f1j.com

Printed in the United States of America
99/700.11

Contents

P R E F A C E

Overview

Computer spreadsheets--tools that help us understand, model, and manipulate relationships between sets of numerical data--further empower us by providing the ability to manipulate, extrapolate, interpret, graph, and display numeric data with the speed and convenience of the computer. Formula One for Java unites all of the powerful utility and familiar interface of the computer spreadsheet with the speed, power, and the universal presence of the Internet.

Designed for use in Java development environments, Formula One for Java provides the tools you need to design, create, and distribute custom spreadsheets over the Internet as part of your application, applet, or JavaBean. Using Formula One for Java, you may also embed high-quality spreadsheets in a Web page with just a few lines of HTML code.

Formula One for Java also works as a standalone spreadsheet application that operates with the speed and functionality of the best desktop spreadsheet.

About The Guide

This Guide, the *Formula One for Java Technical Guide*, provides an outline of the technical specifications and procedures for developers who want to include Formula One for Java in their applications, applets, or JavaBeans. The complete documentation for Formula One for Java consists of three separate publications:

1. *Formula One for Java User's Guide*

   A basic guide to using electronic spreadsheets in general and using Formula One for Java on a desktop or over a network as an end user.

2. *Formula One for Java Technical Guide*

   The Technical Guide includes more advanced concepts and uses.

3. *Formula One for Java Function Reference*

   A complete reference of the 325 functions available for use within Formula One for Java.

Using The Guide

To effectively use the Guide, you should be familiar with the basic operational features and interface characteristics of your operating system and with the concepts and applications of programming with Java.

For the most part, the Guide assumes you are using a Windows operating environment. Keyboard keys and some graphical user interface elements might vary depending upon your operating environment.

Each chapter in the Guide begins with a short introduction to its contents and a list of topics covered like the one presented below.

The following topics are covered.

This Guide and other useful information about Formula One for Java is also available online at www.tidestone.com.

Technical Support

The Tidestone technical support staff can help you with any problem you encounter installing or using Formula One for Java.

For specific information about support plans for Formula One for Java, visit www.tidestone.com and follow the Support links.

Formatting Conventions

Recognizing the typographic conventions listed in the table below will help you understand and use the documentation.

Convention example	Description
selectionChanged **setFormatPaintMode** **launchDesigner**	Names of events, properties, and methods appear in proper case and bold font.
➤ **To install Formula One for Java:**	A series of numbered instructions is preceded by an introductory line like the one shown here.
1.Type `jre -cp .`	Numbered instructions provide step-by-step directions for performing tasks. Perform the instructions in the order presented. In numbered steps, any items to enter appear in Letter Gothic font.
workbook	In general sections, italics indicate the first occurrence of a new term.
fontname	In reference sections, italics indicate variable or argument information you supply.
`import java.applet.*;`	Code examples appear in Letter Gothic font.
F1J7Swing.jar	File names appear in bold type.
Format > Sheet > Properties	The > symbol indicates a series of menu actions. This example means, “Choose the Properties option on the Sheet submenu of the Format menu.”

New and Enhanced Features in Version 7.0

New Features

Java 2 support: Formula One for Java includes support for Java™ 2 SDK v 1.2 (J2SDK).

Swing: Formula One 7.0 includes support for Java’s Swing user interface classes.

Java 2D support: Formula One for Java 7.0 takes advantage of the Java 2D classes of the Java 2 JDK, enabling users and developers to draw lines of varying weights and styles, use different patterns and gradients for background fills, rotate text, superimpose graphics, and leverage more printing options.

Model/View/Controller architecture: Formula One 7.0 includes a new API that enables developers to access the model (workbook) without loading the GUI components of the product.

2D Charting: Formula One for Java includes 2D charting for use in conjunction with Formula One spreadsheets to graphically display spreadsheet data. The charts offer Excel compatibility and include column, bar, high-low (or stack type), line, pie, area, step, combination, XY (scatter), bubble, doughnut, and studies. In the studies chart type multiple sets of data share the same category axis, but plot on separate value axes.

Multiple undos and redos: Formula One for Java's standalone application enables users to perform multiple (up to 100) undos and redos.

InfoBus: Formula One for Java plugs into the InfoBus architecture enabling it to share data with other software that supports InfoBus.

Pluggable recalculation engine: Formula One for Java allows for fine-tuning of the calculational engine with optional specialized code. Each specialized recalculation engine's performance is tailored to a different type of application and carries different size footprints and capabilities, allowing users and developers to determine which engine works best for their application.

Add-in worksheet functions: Developers have the ability to write custom functions in Java for use in Formula One for Java worksheets.

Thread pooling: In Formula One for Java, groups are now independent of threads, allowing for more efficient resource sharing among groups.

Modular JAR files: Formula One for Java offers a pre-determined set of JAR files. This way, developers and web builders only need to deploy the Formula One for Java JAR files utilized in an application, significantly reducing download times.

Enhanced Features

Read/Write Excel Files: Formula One for Java 7.0 adds support for Excel 97 and Excel 2000 formats and retains the ability to create, import, and export Excel version 5 and 7 files.

More functions: Formula One for Java provides 325 of Excel's 329 functions. This feature is aided by 7.0's added support of array formulas, which output multiple numbers to a range of cells. The added functions, including many new statistical analysis, engineering, and financial operations, were created to appeal to specialized end users such as accountants and engineers.

Extended date range: Formula One for Java supports dates through December 31, 9999.

Enlarged workbook capacity: Formula One for Java now supports more than a billion rows and 32,768 columns per worksheet. Workbooks may have up to 32,768 sheets.

Performance: Formula One for Java's enhanced performance matches or exceeds the power of traditional desktop spreadsheet applications. Enhanced performance is most apparent in the functionality of its completely rewritten calculation engine, cell-to-cell copying, reading and writing spreadsheets, and printing features.

Tidestone

CHAPTER 1

Installing Formula One for Java

This chapter provides information about obtaining, preparing to install, licensing and installing Formula One for Java. This chapter also demonstrates how to change class path settings on your system to recognize the software and run applications. The following topics are covered.

Obtaining Formula One for Java

You can obtain Formula One for Java by purchasing the CD software and documentation package or by downloading the software over the Internet from the Tidestone Technologies website at www.tidestone.com. You can also find licensing, pricing, technical support, and other information about Formula One for Java at this site.

Licensing Formula One

You can license Formula One for use on a single computer (as a user or developer); however, Formula One is usually licensed based on how an application that contains Formula One is deployed after development.

You can find specific licensing information online at the Tidestone Technologies website, www.tidestone.com, by e-mail sales@tidestone.com, or by calling the Tidestone sales department at (800) 884-8665.

Formula One for Java Files

The CD package and the Internet download packet include the following basic set of files. The Formula One for Java installer copies these files to your system.

File	Description
F1J7Swing.jar	Formula One for Java classes stored in a compressed JAR format.
F1JSplit.class	Class file required to separate F1J7Swing.jar into its component JAR files.
servlets/	Includes **F1JwriteURL.class**, **write** method servlet source, and readme files required to use the **writeURL** method.
usersguide.pdf	User's Guide in PDF format
techguide.pdf	Technical Guide in PDF format
functionref.pdf	Function Reference in PDF format
jh.jar	Java Help JAR file. Contains the JavaHelp system.
F1Help.jar	Formula One for Java documentation in JavaHelp format.
help/	API in JavaDoc HTML format. Directory of HTML files comprising the API documentation in JavaDoc format.

The PDF files for the User's Guide, Function Reference, and Technical Guide and API Guide files are also available for download from www.tidestone.com.

Platforms and Development Environments Tested

Formula One for Java has been extensively tested and is known to run reliably in Windows 95, Windows 98, Windows NT 4.0, and Solaris 2.6.

System Requirements

To run Formula One for Java, you must have the appropriate Java SDK installed on your system.

You may also need to change the class path settings variable for the development environment to work with Formula One for Java. Check the documentation for your specific development environment to determine whether changing class paths is necessary. See"Setting Class Path" on page 26 for directions on how to change class path settings. You should also be able to run your development environment from the directory where it was installed.

When installing Formula One for Java, if you want the Designer Help to be available from the Workbook Designer Help menu, you must install JavaHelp classes either by including them in your class path settings or by including the **jh.jar** and **F1Help.jar** files in the command line when running from the command prompt.

Installing Formula One for Java

For Windows 95, 98, and NT, the installation program and associated files are contained in a self-extracting executable program. For other platforms, the installation program and files are contained in a JAR file that you must uncompress before installing the software.

➤ **To Install Formula One for Java in Windows 95/98/NT:**

1. Locate the file **F1J7Setup.exe** on the CD or from the files you downloaded and double-click on the file to run the installer.
2. Read and follow the instructions that appear in the installation program windows to identify the components you want to install, and select a directory and folder to hold the program files.

The installation program creates a folder named F1Java containing all the program files and subdirectories. After installation, you can access the Designer, ReadMe file, and Uninstall program by choosing Start > Programs > Formula One.

➤ **To install Formula One for Java from the CD on UNIX:**

Note A 1.1+ JVM is required to run the installer. (1.2+ is recommended.)

1. Open a command prompt or file manager.
2. Change to the root of the CD-ROM.
3. Execute the following command to start the installation:

```
setup.sh
```

4. Follow the instructions in the installation wizard to complete the installation process.

➤ **To install Formula One on all other platforms:**

Note A 1.1+ JVM is required to run the installer. (1.2+ is recommended.)

1. Locate the file setup.class on the root of the CD or from the files you downloaded.
2. Use a JVM to execute the class:

```
java -cp . setup.class
```

3. Follow the instructions in the installation wizard to complete the installation process.

CHAPTER 2

Technical Notes for Webmasters

This chapter focuses on technical information of special interest to Webmasters and others who want to add spreadsheets or spreadsheet functionality to web pages. Chapter 3, “Technical Notes for Developers” covers information of more specific interest to software developers. The following topics are covered in this chapter:

Other topics related to developing applications but documented elswhere include:

Licensing Requirements for Server Deployment

Prior to deploying Formula One for Java or an application using Formula One for Java on a server, each Webmaster deploying the application must purchase a Server Deployment License and/or End User Licenses. Developer Licenses are also required before developing applications that use Formula One for Java. See “Licensing Formula One” on page 2.

Embedding Formula One for Java in Web pages

With Formula One for Java you can embed fully functional spreadsheets in HTML documents as high-performance Java applets.

Required Files

When embedding a Formula One for Java spreadsheet in a web page, Webmasters must place the required Formula One for Java JAR files on their server. The only file required to view and serve live spreadsheets from a web server is **F1J7Swing.jar**. For a complete JAR file listing, see “JAR File Interfaces and Classes” on page 23.

Note If download time is a concern, use a custom JAR file including only the class files required for the functions and operations you need. See “Minimizing File Size” on page 21 for details.

Required Browsers

In order to use the Swing-based workbook, the browser must comply with Sun’s requirements for Swing. Currently, this is JVM 1.1.8 or JVM 1.2.2. Since most browsers do not support those JVM’s natively, the Sun Java 1.2.2 browser Plug-in is required.

Java Plug-In?

In order to run the Swing version within the Java Plug-in, you must convert all of your APPLET tags to OBJECT tags (Internet Explorer) or EMBED tags (Netscape). A Sun-provided converter, HTMLConverter, will do this conversion automatically for you.

You can download the Java Plug-in for your platform and the HTMLConverter at http://java.sun.com.

Refer to the Tidestone website at www.tidestone.com, or contact the Tidestone technical support department at support@tidestone.com for the most recent information on web browser compatibility issues.

What's a VTS File?

Formula One for Java creates worksheets in its native VTS file format. The Formula One for Java Workbook Designer creates, reads, and saves VTS files. Files in the VTS format can also be created, read, and saved by any application or applet created with Formula One for Java.

The VTS file format is a compressed format optimized for portability and network transmission. Usage of VTS files in your application significantly reduces download and transmission times when sharing data via Formula One for Java workbooks.

Embedding Applets

Formula One for Java provides a fully functional spreadsheet applet that can be embedded inside Web pages or in an application with one line of HTML code.

Note All of the examples in this Guide run on web servers that support the Java 2 Platform (available online at java.sun.com/products) and JavaServer Pages (also available on line at java.sun.com/products/jsp).

➤ **To embed an applet in a Web page:**

1. Write an HTML file that contains at least the following code.

Note The archive path must lead to F1J7Swing.jar in this example.

The following example code embeds an applet named SimpleView in an HTML document:

```
<HTML>
<HEAD>
<TITLE> Live Spreadsheet Page </TITLE>
</HEAD>

<BODY>
<APPLET CODE="com.f1j.swing.JBookApplet.class" WIDTH=550
        HEIGHT=375></APPLET>
</BODY>
</HTML>
```

2. If the applet is in a JAR file, specify the name of the JAR file by adding the ARCHIVE option to the APPLET CODE tag.

```
<APPLET CODE="com.f1j.swing.JBookApplet.class"
        ARCHIVE="SimpleView.jar" WIDTH=550 HEIGHT=375></APPLET>
```

3. If the applet is in a CAB file, specify the name of the CAB file by adding the CABBASE option to the APPLET CODE tag.

```
<APPLET CODE="com.f1j.swing.JBookApplet.class"
        CABBASE="SimpleView.cab" WIDTH=550 HEIGHT=375></APPLET>
```

Loading Worksheets with Embedded Applets

In addition to embedding an applet in an HTML page, you can load a worksheet within the embedded applet.

To load a VTS file in an embedded applet, you must add the PARAM tag following the APPLET tag in the HTML code, as shown in the following example where **spreadsheet.vts** is the VTS file you wish to load into the applet:

```
<APPLET CODE="com.f1j.swing.JBookApplet.class" WIDTH=550 HEIGHT=375>
<PARAM NAME="workbook" VALUE="spreadsheet.vts">
</APPLET>
```

In this example, the VALUE attribute is the name of the worksheet you want to load. The PARAM NAME attribute has three settings, as shown in the following table.

Name	Description
GROUP	Calls the **setGroup** method, which sets the group name for the workbook attached to the current view. This allows external references between workbooks in the same group.
WORKBOOKNAME	Calls the **attach** method and attaches the current view to the specified view's workbook. This allows multiple workbooks to display data from the same view. Workbooks must belong to the same group before they can be attached by name.
WORKBOOK	Calls the **readURL** method and reads a workbook from the specified web page.

Reading and Writing Files Using a Browser

When attempting to read and write Formula One for Java files using a browser, you or the users of your application or applet may encounter security problems. Refer to the Chapter "Java Security Issues" on page 39 for information about security settings for your application and the required steps for digitally signing applications. More comprehensive information regarding security features and issues is available online at http://java.sun.com.

Saving Worksheets as HTML

When you are working in a workbook, you can export and save your worksheets as HTML tables. You can also use the **write** method to save an entire worksheet or a selected area of the worksheet as an HTML table.

Many of the format attributes applied to a worksheet can be saved with the exported HTML table. The following list highlights the formats that can be exported. Developers and Webmasters can specify the attributes that are to be saved with HTML tables using the **setFlags** method. The current attribute settings can be obtained using the **getFlags** method.

- **Value formats**. Formatting applied to a number, such as currency or time and date formats, is exported with the cell values.
- **Column spanning**. Data that spans columns in a Formula One for Java worksheet also spans columns in the exported HTML table.
- **Cell borders**. Cell borders applied to worksheet rows and columns are applied to HTML table borders.
- **Text alignment**. Both vertical alignment (top, center, bottom) and horizontal alignment (left, center, right) settings are saved with HTML tables.
- **Row height and column width**. The height and width settings for cells in the worksheet are maintained in the exported HTML table.
- **Font formatting**. Font attributes such as font style (bold, italic, and underline), font color, and font size are saved with the exported table.
- **Background color**. The worksheet cells background color is applied to the table.

Using the Formula One for Java writeURL Servlet

To use the **writeURL** method, you need to install the servlet F1J_writeURL.class file found in the servlets directory onto your web server. The servlets directory contains the file F1J_writeURL.readme which includes detailed directions.

The sample servlet that comes with Formula One for Java is a multi-threaded servlet that takes input from a post and writes it to a specified file.

Note All of the examples in this Guide run on web servers that support the Java 2 Platform (available online at java.sun.com/products) and JavaServer Pages (also available on line at java.sun.com/products/jsp).

Accessing the API Through JavaScript

Webmasters can also access many of the methods in the API through JavaScript.

In the following example, the **StartMeUp** function is defined in JavaScript, and a call is made to the **setShowEditBar** method in Formula One for Java. In this example, the Formula One for Java object is named **f1**, and the name is set by adding the NAME parameter to the APPLET tag.

➤ **To Access the Formula One for Java API through JavaScript**

1. Place F1J7Swing.jar in a directory on the server.

2. Write an HTML page with at least the following code.

(NOTES: The ARCHIVE path must lead to F1J7Swing.jar. You can adjust the values of the HEIGHT and WIDTH attributes to affect how large the spreadsheet will display in the browser.):

```
<HTML>
<HEAD>
<SCRIPT>
function startMeUp() {
    document.f1.getJBook().requestFocus();
    document.f1.getJBook().setShowEditBar(true);
}
</SCRIPT>
</HEAD>

<BODY onload="startMeUp()" BGCOLOR="#FFFFFF">

<CENTER><APPLET CODE="com.f1j.swing.JBookApplet.class"
        ARCHIVE="F1J7Swing.jar" NAME="f1" WIDTH=480 HEIGHT=360>
<PARAM NAME="workbook" VALUE="vtsfiles/empty.vts">
</APPLET></CENTER>

</BODY>
</HTML>
```

3. To test, access the HTML page with a browser. A live spreadsheet should appear after the JAR file downloads.

4. The focus should be on the spreadsheet and its Edit bar should be visible.

This is just one of many parts of the Formula One for Java API that can be accessed through JavaScript.

Using Formula One for Java on a Server (Demo No. 1)

Webmasters can take advantage of the page compilation feature supported in Sun's Java Web Server. Basically, this feature allows a Webmaster or developer to embed Java code in an HTML page within <java> ... </java> tags. Java Web Server then compiles the Java code on the fly into a Servlet and executes the code on the server.

In this example, you create a .jhtml file that asks the user to input a formula. The formula, passed from the HTML form to a Formula One for Java workbook, is created on the fly. Formula One for Java then calculates the formula and returns it to an HTML page.

Comments in the following code further explain how this demo works.

```
<HTML>
<java type=class>
    /*
     * Create a single spreadsheet object for the servlet. This
     * spreadsheet is used to do the calculation when a request
     * comes in.
     */
    com.f1j.swing.JBook workbook = new com.f1j.swing.JBook();
</java>
<HEAD>
<TITLE>Formula One for Java: Server/Calculate Demo</TITLE>
</HEAD>

<BODY>

<CENTER>

<!--Build a table to hold the instructions for the user-->
<TABLE WIDTH=400>
<TR>
<TD WIDTH=400 VALIGN=TOP ALIGN=MIDDLE>
<H4>Formula One for Java<BR>Server/Calculate Demo</H4>
</TD>
</TR>

<TR>
<TD WIDTH=400 VALIGN=TOP>
<FONT SIZE="-1" FACE="ARIAL, HELVETICA">
<P>This demo displays the calculational uses of Formula One for Java.
       This operation requires no download, all you must do to complete
       the test calculation is enter a value in the text box below and
       click the button. Your results will be displayed below.</P>

<CENTER>
<P><B>Enter your formula here.</B></P>

<!--Build an HTML form to capture the user's formula-->
```

```
<!--Note the ACTION statement, which points to this page where the Java
        code is-->
<FORM METHOD=GET ACTION="calculate.jhtml">

<!--The name of the value of the formula to pass to F1J 7is theFormula--
        >
<INPUT NAME="theFormula" TYPE="TEXT" Size="40"><BR>
<FONT SIZE="-2" FACE="ARIAL, HELVETICA">For example: pi() * 2</FONT>
<BR> <BR>
<INPUT TYPE="Submit" value="Calculate">
</FORM>

<FONT SIZE="-1" FACE="ARIAL, HELVETICA">
<P><B>The calculation's results are:</B></P>

<!--This is where the formula will be calculated and its
    results displayed-->


<JAVA>

/*
 * Sun's Java Web Server allows embedded Java code in an HTML page
 * inside of <JAVA>...</JAVA> tags.
 *
 * This simple Java code will lock the workbook, enter a formula,
 * get the result of the formula and return it on an HTML page.
 */

String result = ""; // Output an empty string by default

// Catch any exceptions.
try {
    // Get the URL parameter named "theFormula" if there is one.
    String theFormula = request.getParameter("theFormula");

    if (theFormula != null) {
    /*
     * Lock the workbook so that another request on another thread will
     * not conflict with this request.
     */
     workbook.getLock();
     try {
         /*
          * Enter the formula into the cell at A1. An exception will
          * be thrown if the formula is not properly formed.
          */
          workbook.setFormula(0, 0, theFormula);

          /*
```

```
            * Get the result of the formula. Formula One automatically
            * recalculates the formula for us.
            */
            result = workbook.getText(0, 0);
        }
        finally {
            /*
             * Release the lock. This must happen or no other thread
             * will be able to use the object.
             */
             workbook.releaseLock();
        }
    }
}
catch (Throwable e) {
    // Got an exception. Report it on the HTML page.
    result = "Got exception e=" + e.getMessage();
}
// Output the string.
out.println(result);

</JAVA>

</TD></TR>
</TABLE>

</FONT>
</CENTER>
</BODY>
</HTML>
```

Using Formula One for Java on a Server (Demo No. 2)

This demo builds on the concepts learned in “Using Formula One for Java on A Server (Demo No. 1).” It begins with the same procedure as the last example -- passing data from an HTML form to Formula One for Java on the server and having it return values to an HTML page. In this case, instead of passing just a formula to a Formula One for Java workbook, a pre-formatted spreadsheet to calculate amortization schedules is transferred. The calculated spreadsheet is then written out as HTML. This spreadsheet, mortgage.vts, was created with Formula One for Java and resides in the same directory as the HTML files.

This demo is a framed document. Below are the listings for the frameset, its two pages, and the .jhtml page that powers the demo. Note the comments within the code sections for more information on how this is put together.

This is the code for the frameset, amortization.htm:

```
<HTML>

<HEAD>
<TITLE>Southcreek Realty Co. - Instant Amortization</TITLE>
</HEAD>

<frameset cols=200,* frameborder="0" framespacing="0" border="0">

<frame src="loan.htm" name="left" scrolling=no noresize MARGINWIDTH=0
        MARGINHEIGHT=0>

<frame src="amorstart.htm" name="right" scrolling=yes noresize
        MARGINWIDTH=0 MARGINHEIGHT=0>

</frameset>

</HTML>
```

This is the code for the right page of the frameset, amorstart.htm:

```
<HTML>

<HEAD>
<TITLE>Southcreek Realty Co. - Instant Amortization </TITLE>
</HEAD>

<BODY>
<BR> <BR> <BR>

<CENTER>
<TABLE WIDTH=400>
<TR>
<TD WIDTH=400 VALIGN=TOP>
<CENTER><IMG SRC="topbannr.gif" WIDTH=400 HEIGHT=45 BORDER=0
         ALT="Southcreek Realty"></CENTER><BR> <BR>

<FONT SIZE="-1" FACE="ARIAL, HELVETICA">
<P>Welcome to the Southcreek Realty Instant Amortization feature of our
        Web site. Please
fill out the information at left to see your proposed amortization
        schedule.</P>
</TD></TR>
</TABLE>
</CENTER>

</BODY>
</HTML>
```

This is the code for the left page of the frameset, loan.htm:

```
<HTML>
<HEAD>
<TITLE>Southcreek Realty Company - Amortization Calculator</TITLE>
</HEAD>
<BODY BGCOLOR="#5F8090">
<!--Call loan.jhtml in the ACTION statement to process the form input
        data when submitted-->
<FORM METHOD=GET ACTION="loan.jhtml" TARGET="right">
<TABLE WIDTH=220>
<TR>
<TD WIDTH=220 ALIGN=MIDDLE COLSPAN=2>
<IMG SRC="house.gif" WIDTH=150 HEIGHT=150><BR> <BR></TD>
</TR>

<!--Ask the price of the house; name it price-->
<TR>
<TD WIDTH=120 VALIGN=MIDDLE ALIGN=RIGHT><FONT SIZE="-1" FACE="ARIAL,
        HELVETICA" COLOR="#FFFFF7"><B>List Price: </B></FONT></TD>
<TD WIDTH=100 VALIGN=TOP ALIGN=LEFT><INPUT NAME="price" TYPE="TEXT"
        SIZE="7"></TD></TR>

<!--Ask for the percent of the price being put down; name it down-->
<TR>
<TD WIDTH=120 VALIGN=MIDDLE ALIGN=RIGHT><FONT SIZE="-1" FACE="ARIAL,
        HELVETICA" COLOR="#FFFFF7"><B>Money Down %:</B></FONT></TD>
<TD WIDTH=100 VALIGN=TOP ALIGN=LEFT><INPUT NAME="down" TYPE="TEXT"
        SIZE="7"></TD></TR>
<!--Ask for the percent of the closing cost; name it closing-->
<TR>
<TD WIDTH=120 VALIGN=MIDDLE ALIGN=RIGHT><FONT SIZE="-1" FACE="ARIAL,
        HELVETICA" COLOR="#FFFFF7"><B>Closing %:</B></FONT></TD>
<TD WIDTH=100 VALIGN=TOP ALIGN=LEFT><INPUT NAME="closing" TYPE="TEXT"
        SIZE="7"></TD></TR>

<!--Ask for a 15 or 30 year term on the loan, name it years-->
<TR>
<TD WIDTH=120 VALIGN=MIDDLE ALIGN=RIGHT><FONT SIZE="-1" FACE="ARIAL,
        HELVETICA" COLOR="#FFFFF7"><B>Years:</B></FONT></TD>
<TD WIDTH=100 VALIGN=TOP ALIGN=LEFT><SELECT NAME="years">
<OPTION SELECTED>15</OPTION>
<OPTION>30</OPTION>
</SELECT></TD></TR>

<!--Ask for the rate of the loan, name it interest-->
<TR>
<TD WIDTH=120 VALIGN=MIDDLE ALIGN=RIGHT><FONT SIZE="-1" FACE="ARIAL,
        HELVETICA" COLOR="#FFFFF7"><B>Interest:</B></FONT></TD>
<TD WIDTH=100 VALIGN=TOP ALIGN=LEFT><INPUT NAME="interest" TYPE="TEXT"
        VALUE="7.0" SIZE="3"></TD></TR>
```

```
<TR>
<TD WIDTH=220 COLSPAN=2 ALIGN=MIDDLE><BR> <BR><INPUT TYPE="Submit"
        value="Calculate"></TD></TR>
</TABLE>

</FORM>

</BODY>
</HTML>
```

This is the code for loan.jhtml. This page is referenced as the ACTION in the form on amorstart.htm:

```
<HTML>
<java type=class>
    /*
     * Create a single spreadsheet object for the servlet.
     * This spreadsheet is used to do the calculation when
     * a request comes in.
     */
    com.f1j.swing.JBook workbook = new com.f1j.swing.Jbook();
    boolean needsReadWorkbook = true;
    com.f1j.ss.HTMLWriter htmlWriter = new com.f1j.ss.HTMLWriter();
</java>
<HEAD>
<TITLE>Southcreek Realty Co. - Instant Amortization</TITLE>
</HEAD>
<BODY>

<BR> <BR> <BR>

<CENTER><IMG SRC="topbannr.gif" WIDTH=400 HEIGHT=45 BORDER=0
        ALT="Southcreek Realty"></CENTER><BR> <BR>

<CENTER>
<font point-size=8>
<table><tr><td valign=top>

<java>

/*
 * Sun's Java Web Server allows embedded Java code in an HTML page
 * inside of <JAVA>...</JAVA> tags.
 *
 * This simple Java code will load a workbook, enter data into
 * the workbook, then output the workbook to an HTML page.
 */
```

```
// Get the parameters from the form
String priceParam = request.getParameter("price");
String downParam = request.getParameter("down");
String closingParam = request.getParameter("closing");
String yearsParam = request.getParameter("years");
String interestParam = request.getParameter("interest");

// Make sure the parameters have data in them
if ((priceParam != null) && (downParam != null) && (closingParam !=
        null) && (yearsParam != null)) {
    java.io.Writer writer = new java.io.OutputStreamWriter(out);
    try {
        /*
         * Lock the workbook so that another request on another
         * thread will not conflict with this request.
         */
        workbook.getBook().getLock();
        try {
            if (needsReadWorkbook) {
                /*
                 * Read in the workbook mortgage.vts from the server.
                 * This workbook must be in the directory from which the
                 * web server was run for this to work.
                 */
                htmlWriter.setFlags(htmlWriter.ALIGN_TAG |
                                    htmlWriter.VALUE_FORMATS |
                                    htmlWriter.BORDER_TAG);
                workbook.read("mortgage.vts");
                needsReadWorkbook = false;
            }
            // Enter the data into the workbook
            workbook.setEntry(1, 3, priceParam);
            workbook.setEntry(2, 1, downParam);
            workbook.setEntry(4, 1, closingParam);
            workbook.setEntry(6, 3, yearsParam);
            workbook.setEntry(10, 4, interestParam);

            /*
             * Output the first table.
             */
            htmlWriter.write(workbook, 0, 0, 0, 0, 19, 3, writer);

            /*
             * Output the amortization table.
             */
            writer.write("</td><td valign=top>");
            htmlWriter.write(workbook, 0, 31, 5, 0,
                             yearsParam.equals("15") ? 46 : 61,
                             9, writer);
        }
```

```
            finally {
               /*
                * Release the lock. This must happen or no other thread
                * will be able to use the object.
                */
               workbook.getBook().releaseLock();
            }
        }
        catch (java.io.IOException e) {
            // Got an exception. Report it on the HTML page.
            writer.write("Got exception e=" + e.getMessage());
        }
        catch (com.f1j.F1Exception e) {
            // Got an exception. Report it on the HTML page.
            writer.write("Got exception e=" + e.getMessage());
        }
    }

    </java>
    </td></tr></table>

    </CENTER>

    </BODY>
    </HTML>
```

CHAPTER 3

Technical Notes for Developers

This chapter focuses on technical information of interest to software developers and others who want to add spreadsheets or spreadsheet functionality to their applications. Chapter 2, “Technical Notes for Webmasters” covers information of more interest to Webmasters. The following topics are covered in this chapter:

Other topics related to developing applications but documented elsewhere include:

Licensing Requirements for Development

Prior to developing an application using Formula One for Java as a JavaBean, each developer working on the application must purchase a Developer License. Server Deployment and/or End User Licenses are also required before deploying any application. See “Licensing Formula One” on page 2.

Formula One for Java in Applications and Applets

Formula One for Java contains all the tools you need to create, store, analyze, manipulate, print and present data in your applications providing rich spreadsheet functionality as a high-performance component in Java development environments.

Formula One for Java leverages Sun’s JavaBean application program interface (API) to deliver Excel-compatible spreadsheet functionality to network and Internet-based applications. JavaBeans allow developers and Webmasters to create reusable, platform-independent program components. With JavaBean-compliant application builder tools like Visual Cafe, JBuilder, and Java Workshop, developers and Webmasters combine Formula One for Java and other components into applets, applications, or composite components and deploy them in any operating system or Java Runtime Environment (JRE), including Netscape and Microsoft Internet browsers.

This Technical Guide outlines and illustrates the basics of how to implement Formula One for Java in various development environments. For more information and instruction in using the Java development platform, see the online tutorials and other information at http://java.sun.com.

Swing and Formula One for Java File Size

Developers and Webmasters should take advantage of Java’s Swing technology in developing new applications, applets, servlets, and JavaBeans. Formula One for Java fully implements Swing technology including a user-selectable Look and Feel, use of custom icons and logos, and customized Swing component behavior.

Using the Java Foundation Class (JFC) Swing offers many advantages over using Abstract Windowing Toolkit. Using Swing, you can:

- choose, modify or create the appearance and behavior of your application’s GUI using the pluggable look and feel features
- gain significantly better performance--Swing uses less memory, uses fewer system resources, and produces smaller, more efficient applications

Minimizing File Size

By using only the files required to add the specific Formula One for Java functionality needed in your application, you can significantly reduce the file size of the applets and JavaBeans you create for use over the Internet.

➤ **To reduce Formula One for Java file sizes:**

1. Separate Formula One for Java into its component JAR files by running **F1JSplit.class**.

   At the command prompt in the install directory, type and enter:

   ```
   java -cp . F1JSplit
   ```

2. Formula One is split into its component JAR files, listed in the table "JAR File Interfaces and Classes" on page 23.
3. Select the JAR files Formula One needs based on the minimum functionality desired using the "JAR File Functionality Matrix" on page 22, below.
4. Attach the files to your application as explained in "Sharing Spreadsheets" on page 61.

Server Deployment

To facilitate deployment on servers that have incompatibilities with Java 2 or with GUI elements, Formula One includes four special JAR files that can be split out from the standard distribution.

JAR File	Description
F1J7Model.jar	files required to run the Formula One model only (no GUI).
F1J7JBook_JDK1.jar	all files in **F1J7JBook.jar** except six Java 2-based files
F1J7AWT_JDK1.jar	all files in **F1J7AWT.jar** except Java 2-based files
F1J7AWTDesign_JDK1.jar	all files in **F1J7AWTDesign.jar**. except Java 2-based files.

➤ **To use the special Formula One JAR files when deploying on a server:**

1. Separate Formula One for Java into its component JAR files.
2. At the command prompt in the install directory, type and enter:

Swing:

```
java -cp . F1JSplit F1J7Swing.jar
```

AWT:

```
java -cp . F1JSplit F1J7AWT.jar
```

or

```
java -cp . F1JSplit F1J7AWTDesign.jar
```

3. Load the JAR files needed for deployment as described by your server installation instructions.

JAR File Functionality Matrix

This table illustrates 14 popular ways to deploy Formula One. For Formula One to work you must include the combinations of JAR files shown as a minumum. Iterations described in the left column require the files indicated in the right columns to run properly.

	. . . you need these files:							
To run a minimized version of Formula One with this functionality . . .	**F1J7JBook.jar**	**F1J7Calc.jar**	**F1J7Graphics.jar**	**F1J7DB.jar**	**F1J7ExtFuncs.jar**	**F1J7InfoBus.jar**	**F1J7ProCalc.jar**	**F1J7Design.jar**
JBook on an applet or frame	X							
JBook with basic calculation capabilities	X	X						
JBook with charting or GRObject capabilities (no calculation engine)	X		X					
JBook with charting or GRObject capabilities w/ calculation engine	X	X	X					
JBook with ProCalc calculation engine	X	X					X	
JBook allowing access to Workbook Designer	X	X						X
Reading and writing files with JBook without charting or GRObjects	X	X						
Reading and writing files with JBook that contain Charts or GRObjects	X	X		X				X
JBook with access to JDBC and JDBCDlg classes	X	X		X				X
JBook with advanced functions	X	X			X			
InfoBusBook access	X	X				X		
HTML Writer	X							
Book with calculation capabilities (Model with no GUI)	X	X						
JBook with access to JDBC class with no JDBCDlg	X	X		X				

JAR File Interfaces and Classes

F1J7JBook.jar — Complete set of core files for creating a Jbook

Interfaces
Constants
CancelEditListener
EndEditListener
EndRecalcListener
ModifiedListener
ObjectListener
SelectionChangedListener
StartEditListener
StartRecalcListener
UpdateListener
ValidationFailedListener
ViewChangedListener

Classes
CancelEditListener
EndEditListener
EndRecalcListener
ModifiedListener
ObjectListener
SelectionChangedListener
StartEditListener
StartRecalcListener
UpdateListener
ValidationFailedListener
ViewChangedEvent
ViewChangedListener

Classes (continued)
Book
CellFormat
CellRef
FindReplaceInfo
Format
F1Exception
GRObject
GRObjectPos
HTMLWriter
NumberFormat
RangeRef
Sheet
CancelEditEvent
Dialog
EndEditEvent
EndRecalcEvent
JBook
JBookApplet
ModifiedEvent
ObjectEvent
SelectionChangedEvent
StartEditEvent
StartRecalcEvent
UpdateEvent
ValidationFailedEvent

F1J7Calc.jar — Main calculation engine plus some basic functions files

Interfaces
FuncContext
Value

Classes
Func
Database

F1J7DB.jar — JDBC class files required for database access using JDBC

Classes
JDBC
JDBCQueryObj
JDBCDlg

File	Description	Includes the Interfaces and classes
F1J7Design.jar	Swing Workbook Designer GUI for Formula One	**Classes** ClearDlg ColWidthDlg DefColWidthDlg DefFontDlg DefinedNameDlg DefRowHeightDlg DeleteDlg Designer FindDlg FormatCellsDlg FormatObjectDlg FormatSheetDlg GotoDlg InsertDlg OptionsDlg PageSetupDlg PasteSpecialDlg ReplaceDlg RowHeightDlg SortDlg
F1J7ExtFuncs.jar	All the functions not in F1J7Calc.jar	
F1J7Graphics.jar	Charting and Graphics Objects	**Classes** GRChart
F1J7InfoBus.jar	InfoBus class files required for database access using InfoBus	
F1J7ProCalc.jar	Professional Calculation Engine	

Running the Workbook Designer

You can invoke the Workbook Designer using a GUI or the command prompt.

➤ **To Launch The Workbook Designer in Windows 95/98/NT:**

- Locate and double-click on the **F1J7.exe** file.

  or

- Click Start > Programs > Formula One for Java > Formula One for Java.

  or

- Follow the directions under Any Platform, below.

➤ **To Launch The Workbook Designer on Solaris Machines:**

A shell script is included with the installer, which launches the Workbook Designer.

1. At the command prompt, switch to the installation directory.
2. Execute the following command:

```
./f1j7
```

Note Formula One for Java automatically loads a patch file for the Solaris platform that checks whether the program is launching on a Solaris machine. This very small patch, **solaris.java**, may be safely deleted by non-Solaris users.

➤ **To Launch The Workbook Designer on Any Platform from a Command Prompt:**

1. Switch to the directory that contains Formula One for Java 7.
2. Execute the following command:

```
java -classpath c:\F1J7Swing.jar com.f1j.swing.designer.Designer
```

3. Execute the following command to include the JavaHelp documentation:

```
java -classpath c:\F1J7Swing.jar;jh.jar;F1help.jar
        com.f1j.swing.designer.Designer
```

Note The Any Platform from a Command Prompt instructions assume that a supported Java Virtual Machine (JVM) is installed.

Setting Class Path

You do not have to modify your system's class path to install or run Formula One for Java. However, certain situations may require that you manually change your systems's class path settings, including:

- Running Formula One for Java on other than the default virtual machine
- Troubleshooting errors including messages about missing java-specific files
- Specifying which JAR files to run when running a limited version of Formula One for Java
- Clearing leftover settings from earlier versions of software.

Changing Class Path Startup Settings

This section provides instructions for changing class path on Windows 95, Windows 98, Windows NT, and UNIX operating systems. Examples for both changing the class path environment startup settings and using Java's `-classpath` option methods of changing class path settings are included. Changing the class path environment startup settings saves the changes in the system settings. Changing settings using the JVM's `-classpath` option only changes the settings for the current session.

➤ **On a Windows NT 4.0 machine:**

1. Select Start > Settings > Control Panel to display the Control Panel window.
2. Double-click on the System icon to display the System Properties dialog.
3. Select the Environment tab.
4. Click on the CLASSPATH variable in the User Variables list. (You may need to create the CLASSPATH variable using the variable and value frames.)
5. If you want to use **F1J7Swing.jar**, add the following path to your class path:

   ```
   <install-dir>\F1J7Swing.jar
   ```

6. Add or delete any other JAR files needed to run Formula One for Java in a limited mode (see "Swing and Formula One for Java File Size" on page 20).
7. Click Set.
8. Click OK.

➤ **On a Windows 95 or Windows 98 machine:**

1. Edit the Autoexec.bat file located in your root directory.
2. Look for the set command
3. If you are using **F1J7Swing.jar**, add the following path to your CLASSPATH:

   ```
   <install-dir>\F1J7Swing.jar
   ```

4. Add or delete any other JAR files needed to run Formula One for Java in a limited mode (see "Swing and Formula One for Java File Size" on page 20).

5. Save the **Autoexec.bat** file.
6. Reboot the computer.

➤ **On a UNIX or Solaris Machine:**

1. If you are using **F1J7Swing.jar**, at the command prompt, add the following path:

```
$ set CLASSPATH = $CLASSPATH: <install-dir>/F1J7Swing.jar
$ export CLASSPATH
```

2. Add or delete any other JAR files needed to run Formula One for Java in a limited mode (see "Swing and Formula One for Java File Size" on page 20).

Using the JVM's -classpath Option

This is the recommended option for changing class path settings since each application uses the class path it needs without interfering with other applications.

Note The java and jdb tools have a -cp (an abbreviation of class path) option. This option is used in the example below.

Further information about class path variables and their use in the Java development environment is available on the Internet at http://java.sun.com

➤ **Any Platform from a Command Prompt:**

1. Switch to the directory that contains Formula One for Java (for example, c:\F1Java):
2. Execute the following command:

```
java -cp F1J7Swing.jar com.f1j.swing.designer.Designer
```

or

Execute the following command to include the JavaHelp documentation:

```
java -cp F1J7Swing.jar;jh.jar;F1Help.jar
        com.f1j.swing.designer.Designer
```

Note On Unix machines, replace the semicolons (;) with colons (:).

Adding Formula One for Java Components to Applications

How you add Formula One for Java to a software development environment such as Symantec Visual Cafe, Borland JBuilder, and Sun Java Workshop varies from one development environment to another. Java software development environment software changes frequently. Consult the documentation for your specific software development environment software for directions.

Accessing the Formula One for Java API

When you add Formula One for Java to a form, the development environment generates much of the code required to use the component. You can tap into further functionality to customize the appearance and behavior of a workbook by accessing the Formula One for Java API as shown in the following examples.

Note All of the examples in this Guide run on web servers that support the Java 2 Platform (available online at java.sun.com/products) and JavaServer Pages (also available on line at java.sun.com/products/jsp).

Creating an Application

This example creates an application that displays a spreadsheet, accesses the API, and places a text string in a worksheet cell. Class and method names appear in bold type.

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

public class SimpleApp extends Frame
{
    public SimpleApp()
    {
        // set the size of the application window
        setSize(insets().left + insets().right + 405,insets().top +
                insets().bottom + 305);
        // give the application window a title
        setTitle("Simple Application");

        // create a new instance of Formula One for Java and add
        // it to the frame.
        JBook jb = new com.f1j.swing.JBook();
        add(jb);

        // Call the Formula One for Java setText API to place text
        // in a cell
```

```
        try {
            jb.setText(2, 0,
                       "Welcome to Formula One for Java: " +
                       "The Internet Spreadsheet");
        }
        catch (com.f1j.util.F1Exception e) {
            System.out.println(e.getMessage());
        }

        // add a window listener to handle the window closing
        SimpleWindow sWindow = new SimpleWindow();
        this.addWindowListener(sWindow);
    }

    static public void main(String args[])
    {
        // starting point for the application,
        // create an instance of our SimpleApp frame and show it
        (new SimpleApp()).show();
    }

    class SimpleWindow extends java.awt.event.WindowAdapter
    {
        // window closing event was captured
        public void windowClosing(java.awt.event.WindowEvent event)
        {
            // is the window closing event coming from the
            // SimpleApp frame
            Object object = event.getSource();
            if (object == SimpleApp.this)
                SimpleApp_WindowClosing(event);
        }
    }

    void SimpleApp_WindowClosing(java.awt.event.WindowEvent event)
    {
        hide();         // hide the Frame
        dispose();      // free the system resources
        System.exit(0); // close the application
    }

}
```

Creating an Applet

The following sample code, like the previous example, displays a spreadsheet and accesses the API to place a text string in a worksheet cell. However, this code creates an applet instead of an application.

```
import java.awt.*;
```

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

public class SimpleApp extends JFrame
{
    public SimpleApp()
    {
        // Initialize application
        setSize(600,300);
        setTitle("Simple Application");
        addWindowListener(new WindowAdapter() {
            public void windowClosing( WindowEvent e ) {
                System.exit(0);
            }
        });

        // create a new instance of Formula One for Java and add it to
        // the frame.
        JBook jb = new com.f1j.swing.JBook();
        getContentPane().add(jb);

        // Call the Formula One for Java setText API to place text in a
        // cell
        try {
            jb.setText(2, 0,
                       "Welcome to Formula One for Java: The Internet
        Spreadsheet");
        }
        catch (com.f1j.util.F1Exception e) {
            System.out.println(e.getMessage());
        }
    }

    static public void main(String args[])
    {
        (new SimpleApp()).setVisible(true);
    }
}
```

Creating a Servlet

The following sample code creates a servlet that creates a framed document. Below is the code for the frameset, its two pages, and the servlet that powers the example. Note the comments within the code sections for more information on how this is put together.

Note You can download the mortgage VTS file (54 KB) used in this example from www.tidestone.com.

This is the code for the frameset, amortization.htm:

```
<HTML>

<HEAD>
<TITLE>Southcreek Realty Co. - Instant Amortization</TITLE>
</HEAD>


<frameset cols=200,* frameborder="0" framespacing="0" border="0">

<frame src="loanservlet.htm" name="left" scrolling=no noresize
        MARGINWIDTH=0 MARGINHEIGHT=0>

<frame src="amorstart.htm" name="right" scrolling=yes noresize
        MARGINWIDTH=0 MARGINHEIGHT=0>

</frameset>

</HTML>
```

This is the code for the right page of the frameset, amorstart.htm:

```
<HTML>

<HEAD>
<TITLE>Southcreek Realty Co. - Instant Amortization</TITLE>
</HEAD>

<BODY>

<CENTER>
<TABLE WIDTH=400>
<TR>
<TD WIDTH=400 VALIGN=TOP>
<CENTER><IMG SRC="topbannr.gif" WIDTH=400 HEIGHT=45 BORDER=0
        ALT="Southcreek Realty"></CENTER><BR> <BR>

<FONT SIZE="-1" FACE="ARIAL, HELVETICA">
<P>Welcome to the Southcreek Realty Instant Amortization feature of our
        Web site. Please
fill out the information at left to see your proposed amortization
        schedule.</P>
</TD></TR>
</TABLE>
</CENTER>

</BODY>
</HTML>
```

This is the code for the left page of the frameset, **loanservlet.htm**:

```
<HTML>

<HEAD>
<TITLE>Southcreek Realty Company - Amoritization Calculator</TITLE>
</HEAD>

<BODY BGCOLOR="#5F8090">

<!--Make a path to the servlet in the ACTION statement to process the
        form input data when submitted-->
<FORM METHOD=GET ACTION="http://www.f1j.com/servlet/LoanServlet"
        TARGET="right">

<TABLE WIDTH=220>
<TR>
<TD WIDTH=220 ALIGN=MIDDLE COLSPAN=2>
<IMG SRC="house.gif" WIDTH=150 HEIGHT=150><BR> <BR></TD>
</TR>

<!--Ask the price of the house; name it price-->
<TR>
<TD WIDTH=120 VALIGN=MIDDLE ALIGN=RIGHT><FONT SIZE="-1" FACE="ARIAL,
        HELVETICA" COLOR="#FFFFF7"><B>List Price: </B></FONT></TD>
<TD WIDTH=100 VALIGN=TOP ALIGN=LEFT><INPUT NAME="price" TYPE="TEXT"
        SIZE="7"></TD></TR>

<!--Ask for the percent of the price being put down; name it down-->
<TR>
<TD WIDTH=120 VALIGN=MIDDLE ALIGN=RIGHT><FONT SIZE="-1" FACE="ARIAL,
        HELVETICA" COLOR="#FFFFF7"><B>Money Down %:</B></FONT></TD>
<TD WIDTH=100 VALIGN=TOP ALIGN=LEFT><INPUT NAME="down" TYPE="TEXT"
        SIZE="7"></TD></TR>

<!--Ask for the percent of the closing cost; name it closing-->
<TR>
<TD WIDTH=120 VALIGN=MIDDLE ALIGN=RIGHT><FONT SIZE="-1" FACE="ARIAL,
        HELVETICA" COLOR="#FFFFF7"><B>Closing %:</B></FONT></TD>
<TD WIDTH=100 VALIGN=TOP ALIGN=LEFT><INPUT NAME="closing" TYPE="TEXT"
        SIZE="7"></TD></TR>

<!--Ask for a 15 or 30 year term on the loan, name it years-->
<TR>
<TD WIDTH=120 VALIGN=MIDDLE ALIGN=RIGHT><FONT SIZE="-1" FACE="ARIAL,
        HELVETICA" COLOR="#FFFFF7"><B>Years:</B></FONT></TD>
<TD WIDTH=100 VALIGN=TOP ALIGN=LEFT><SELECT NAME="years">
<OPTION SELECTED>15</OPTION>
<OPTION>30</OPTION>
</SELECT></TD></TR>
```

```
<!--Ask for the rate of the loan, name it interest-->
<TR>
<TD WIDTH=120 VALIGN=MIDDLE ALIGN=RIGHT><FONT SIZE="-1" FACE="ARIAL,
        HELVETICA" COLOR="#FFFFF7"><B>Interest:</B></FONT></TD>
<TD WIDTH=100 VALIGN=TOP ALIGN=LEFT><INPUT NAME="interest" TYPE="TEXT"
        VALUE="7.0" SIZE="3"></TD></TR>

<TR>
<TD WIDTH=220 COLSPAN=2 ALIGN=MIDDLE><BR> <BR><INPUT TYPE="Submit"
        value="Calculate"></TD></TR>
</TABLE>

</FORM>

</BODY>
</HTML>
```

This is the code for **LoanServlet.class**. This servlet is referenced as the ACTION in the form on **loanservlet.htm.**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.f1j.swing.*;

public class LoanServlet extends HttpServlet {

    com.f1j.swing.JBook workbook = new com.f1j.swing.JBook();
    boolean needsReadWorkbook = true;
    com.f1j.ss.HTMLWriter htmlWriter = new com.f1j.ss.HTMLWriter();

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {

        res.setContentType("text/html");

        ServletOutputStream out = res.getOutputStream();
        out.println("<html>");
        out.println("<head><title>Southcreek Realty Co. - " +
                    "Instant Amortization</title></head>");
        out.println("<body>");

        out.println("<BR> <BR> <BR>");
        out.println("<CENTER>" +
                    "<IMG SRC='http://www.f1j.com/demos/topbannr.gif'" +
                    "WIDTH=400 HEIGHT=45 BORDER=0" +
                    "ALT=Southcreek Realty><CENTER><BR> <BR>");

        out.println("<CENTER>");
        out.println("<font point-size=8>");
        out.println("<table><tr><td valign=top>");
```

```
    // Get the parameters from the form
String priceParam = req.getParameter("price");
String downParam = req.getParameter("down");
String closingParam = req.getParameter("closing");
String yearsParam = req.getParameter("years");
String interestParam = req.getParameter("interest");
// Make sure the parameters have data in them
if ((priceParam != null) && (downParam != null) &&
    (closingParam != null) && (yearsParam != null)) {

    java.io.Writer writer = new java.io.OutputStreamWriter(out);
    try {
        /*
         * Lock the workbook so that another request on another
         * thread will not conflict with this request.
         */
         workbook.getBook().getLock();
         try {
             if (needsReadWorkbook) {
                 /*
                  * Read in the workbook mortgage.vts from the
                  * server. This workbook must be in the
                  * directory from which the web server was run
                  * for this to work.
                  */
                  htmlWriter.setFlags(htmlWriter.ALIGN_TAG |
                                      htmlWriter.VALUE_FORMATS |
                                      htmlWriter.BORDER_TAG);
                  workbook.read("servlets/mortgage.vts");
                  needsReadWorkbook = false;
              }
              // Enter the data into the workbook
              workbook.setEntry(1, 2, priceParam);
              if (!downParam.endsWith("%"))
                 downParam = downParam + "%";

              workbook.setEntry(2, 1, downParam);

              if (!closingParam.endsWith("%"))
                   closingParam = closingParam + "%";

              workbook.setEntry(4, 1, closingParam);
              workbook.setEntry(6, 2, yearsParam);
              workbook.setEntry(10, 3, interestParam);

              /*
               * Output the first table.
               */
              htmlWriter.write(workbook.getBook(), 0, 0, 0,
                               0, 19, 2, writer);
```

```
                    /*
                     * Output the amortization table.
                     */
                    writer.write("</td><td valign=top>");
                    htmlWriter.write(workbook.getBook(), 0, 31,
                                     5, 0,
                                     yearsParam.equals("15")?46:61,
                                     9, writer);
                }
                finally {
                    /*
                     * Release the lock. This must happen or no
                     * other thread will be able to use the object.
                     */
                    workbook.releaseLock();
                }
            }
            catch (java.io.IOException e) {
                // Got an exception. Report it on the HTML page.
                writer.write("Got exception e=" + e.getMessage());
            }
            catch (com.f1j.util.F1Exception e) {
                // Got an exception. Report it on the HTML page.
                writer.write("Got exception e=" + e.getMessage());
            }
        }

        out.println("</td></tr></table>");
        out.println("</CENTER>");
        out.println("</body></html>");
    }

    public String getServletInfo() {
        return "Southcreek Realty Co. - Instant Amortization";
    }
}
```

Loading Worksheets with Embedded Applets

In addition to embedding an applet in an HTML page, you can load a worksheet within the embedded applet.

Formula One for Java creates worksheets in its native VTS file format. Files in the VTS format can be created, read, and saved by any application or applet created with Formula One for Java. The Formula One for Java Workbook Designer also creates, reads, and saves VTS files.

The VTS file format is a compressed format optimized for portability and network transmission. Usage of VTS files in your application significantly reduces download and transmission times when sharing data via Formula One for Java workbooks.

To load a Formula One for Java VTS file in an embedded applet, you must add the PARAM tag within the APPLET tag in the HTML code, as shown in the following example:

```
<APPLET CODE="com.f1j.swing.JBookApplet.class" WIDTH=550 HEIGHT=375>
<PARAM NAME="workbook" VALUE="spreadsheet.vts">
</APPLET>
```

In this example, the VALUE option is the name of the worksheet you want to load.

The NAME attribute has three settings, as shown in the following table.

Option	Description
GROUP	Calls the **setGroup** method, which sets the group name for the workbook attached to the current view. This allows external references between workbooks in the same group.
WORKBOOKNAME	Calls the **attach** method and attaches the current view to the specified view's workbook. This allows multiple workbooks to display data from the same view. Workbooks must belong to the same group before they can be attached by name.
WORKBOOK	Calls the **readURL** method and reads a workbook from the specified web page.

Reading and Writing Files Using a Browser

When reading and writing Formula One for Java files using a browser, you and the users of your application, applet, or JavaBean may encounter security problems. Refer to "Java Security Issues" on page 39 for information about security settings for your application and the required steps for digitally signing applications. More comprehensive information regarding security features and issues is available online at http://java.sun.com for the Java 2 Platform and at http://java.sun.com for JDK 1.1.x.

Saving Worksheets as HTML

When you are working in a workbook, you can export and save your worksheets as HTML tables. You can also use the **write** method to save an entire worksheet or a selected area of the worksheet as an HTML table.

Many of the format attributes applied to a worksheet can be saved with the exported HTML table. The following list highlights the formats that can be exported. Specify the attributes to be saved with HTML tables using the **setFlags** method. Find out the current attribute settings using the **getFlags** method.

- **Value formats**. Exports formatting applied to a number, such as currency or time and date formats, with the cell values.
- **Column spanning**. Exports data that spans columns.

- **Cell borders**. Exports cell borders applied to worksheet rows and columns.
- **Text alignment**. Saves vertical alignment (top, center, bottom) and horizontal alignment (left, center, right) settings.
- **Row height and column width**. Maintains cell height and width settings.
- **Font formatting**. Saves font attributes such as font style (bold, italic, and underline), font color, and font size.
- **Background color**. Applies the background color applied to worksheet cells to the table.

Tidestone

CHAPTER 4

Java Security Issues

To read and write files from Formula One for Java with a browser, first you must resolve Java security issues found in browsers such as Internet Explorer and Netscape Navigator. Software developers and Webmasters must obtain their own digital signatures to enable their users to read and write files to and from their local disk or to print to a local printer--digital signatures are not included among the Formula One for Java product files. Please note that you use digital signatures only when Formula One for Java needs to access system resources. Simply adding a spreadsheet to your Web pages does not require a digital signature.

The following topics are covered in this chapter:

You can find more comprehensive information regarding Java security issues on-line at http://java.sun.com. Specific information about how the Java 2 Platform's security architecture works, differs from, and improves over previous versions of JDK is available at http://java.sun.com

Introduction to Java Security Issues

In developing applets for distribution over the Internet, developers must consider security issues arising from both the development environment and the browsers on which the applet will run. Java security issues work differently in the different versions of the Java development environment (JDK 1.0, 1.1 or Java 2). The Java 2 development environment adds several new security capabilities, including the option for both developers and users to set levels of security and improvements in the use of public and private keys, certificates and certification authorities.

In general, the Java platform's security architecture prevents applets downloaded from a server from:

- reading or writing to your hard drive;
- connecting to another server;
- creating independent commands;
- loading a new dynamic library;
- defining native method calls; or
- starting other programs on your computer.

And, beyond Java's general security architecture, Java 2:

- allows communication only between applets from the same Internet Protocol (IP) address, the same domain name and only on the same page
- gives users and developers the power to impose additional security measures or remove existing security restrictions using a system of security policies that defines access permissions within individual code domains.

How Digital Signatures Work

Digital signatures, created through the use of a system of private keys, public keys, and certificates, provide the framework of Java's security structure. All three of these elements must be included in a JAR file to create a "signed file" that a user can download and use with some level of security.

- Private Keys

A private key is a password-protected numeric code that identifies the individual sending the signed code or document. Private keys can be generated using Java's **keytool** or an API. To verify legitimacy, unmodified private keys match corresponding public keys to form a "matching key pair."

- Public Keys

A public key is a password-protected numeric code that identifies the entity generating the signed code or document. A public key always corresponds to a

private key. Public keys can be verified by the use of Certificates and Certificate Authorities.

- Certificates

A certificate contains a public key, information about the certified entity (entity name, organization, address), and a digital signature from (and information about) the issuer of the certificate.

Various levels of security are possible using digital signatures. The simplest security level is to check whether the private and public keys correspond (whether they are a “matching key pair”) which can be further checked by contacting the issuer of the code or file containing the keys and verbally verifying the signatures. The use of certificates generated by third-party Certificate Authorities adds another level of security. Certificate Authorities, firms that specialize in digital security (such as Verisign), issue certificates signed with their own private key which, in turn, verifies that you are the owner of your public key.

The Java 2 Development Platform adds capabilities to the Java Security Architecture through the use of policies, permissions and domains. Policies create the capability of setting your own security policies (such as not allowing system access to code downloaded from sources without third-party certification). Permissions allow users to grant applets access to system components and other “permissions” based on the level of security. Files with the same permissions are enclosed in the same “domain” and have been downloaded from the same source. Downloaded applets and applications run in their own domain based on this combination of policies, permissions, and domains.

Preparing Applets for Distribution

When you prepare an applet for distribution, you must package the applet with your certificate and key information, and compress and package the Formula One for Java classes and the classes you created for your applet into a JAR file or a CAB file.

The following sections explain and illustrate how to prepare an applet for distribution using RSA verification with Java 2, Microsoft Internet Explorer and Netscape Navigator browsers.

Preparing an Applet using RSA Verification and the Java 2 Plug-in

Using this signing procedure presents several advantages over using browser-specific procedures:

- Preparing an Applet for deployment works for both Netscape Navigator and Internet Explorer browsers.
- RSA's digital signature algorithm is used by Netscape Navigator, Internet Explorer and most other browsers and recognized across the Internet.
- RSA recognizes most common certificates, including Verisign and Thawte RSA certificates.
- RSA is used in the same way in all browsers, providing browser-independent signature verification.
- Users can validate signers and set security levels based on individual signers.

Creating an RSA Signed Applet

➤ **To create an RSA signed applet:**

1. Make sure you have:
   a. Netscape Signing Tool.
   a. A Netscape Object Signing Certificate from an RSA Certificate
   b. A certificate from an independent Certificate Authority (CA) such as BelSign, Verisign, or Thawte.
2. Install Netscape Signing Tool (download from http://developer.netscape.com) if necessary.

Note Install the Netscape Signing Tool and the Object Signing Certificates in a secure location.

---

3. Create the applet:
   a. Archive the applet's .class files into a JAR file using the "jar" command in the Java SDK.

   Type and enter the following code from a command prompt:

```
jar -cvf c:\jar_name.jar class1.class class2.class
```

Where:

`-cvf` tells the program to create a verbose file

`jar_name.jar` tells the program the output file name

`class1.class`, `class2.class` tells the program to include these class files

c. If you want to include the Formula One for Java classes in the same JAR file you need to first unjar f1j7Swing.jar using the following code:

```
jar -xvf c:\f1j7Swing.jar
```

This unjars the JAR file into a subdirectory beginning with c:\com\f1j.

4. Use Netscape Signing Tool to sign the JAR file.

See “Signing the JAR File” on page 45 for instructions.

Note The **jar.exe** program is distributed with most Java development environments, or you can download it from the Sun Microsystems website at www.java.sun.com.

Converting Netscape signed applets to RSA Signed Applets

JAR files signed using earlier versions of Netscape **signtool.exe** will not run properly in Java Plug-in.

➤ **To convert Netscape signed applets to RSA Signed Applets:**

1. Delete or remove references to netscape.security.* from the applet.
2. Archive and sign following instructions under steps 3. and 4., above.

More information about RSA and using RSA Verification with the Java 2 Plug-in is available online at http://www.rsa.com and http://www.java.sun.com.

Information about Netscape Signing Tool may be found at http://developer.netscape.com.

Preparing an Applet for Netscape Navigator

For Navigator, all of the classes for your applet must be contained in a JAR file. You can append the class files from your applet to the Formula One for Java JAR file, or you can decompress the Formula One for Java class files and create a new JAR file with your applet classes and a subset of the Formula One classes. Regardless of the method you use, for Navigator, digital signatures are applied to JAR files.

Digital Signature Information for Netscape Navigator

For Netscape Navigator, information about your software publishing certificate and key are contained in two files: **key3.db** and **cert7.db**. When you obtain software publishing certificates and keys from signing authorities, certificate and key information is placed in these files.

Typically, **key3.db** and **cert7.db** are located in the Netscape directory on your system. On UNIX systems, if you don't already have a ~/.netscape directory, you can run Netscape Communicator to create the necessary directory structure. In Windows 95 and Windows NT, these files are typically located in \Program Files\Netscape\Users\Username.

Creating the JAR File

Create the JAR file using **jar.exe**.

➤ **To add the class files into a JAR file using jar.exe:**

1. Type and enter the following code from a command prompt:

   ```
   jar -cvf c:\jar_name.jar class1.class class2.class
   ```

   Where:

   `-cvf` tells the program to create a verbose file

   `jar_name.jar` tells the program the output filename

   `class1.class`, `class2.class` tells the program to include these class files

2. If you want to include the Formula One for Java classes in the same JAR file you need to first unjar f1j7Swing.jar using the following code:

   ```
   jar -xvf c:\f1j7Swing.jar
   ```

   This unjars the JAR file into a subdirectory beginning with c:\com\f1j.

3. Reference this subdirectory when adding it to the JAR file you want to sign.

   ```
   jar -cvf c:\com\f1j\* class1.class class2.class
   ```

Note Be careful not to include directory names with the class file names. This adds directory names to the class files in the JAR file and, once placed on a web server, these files would not be found.

Signing the JAR File

To sign a JAR file, you actually create a new signed JAR file. The applet JAR file you create can be digitally signed for Navigator with **signtool.exe**. You can download this program from the Netscape website at home.netscape.com. Look for the Object-Signing Tools page in the DevEdge On-line section.

Note This procedure assumes you have obtained a digital signature and key information from a signing authority.

➤ **To digitally sign a JAR file:**

1. Find the nickname from your **cert7.db** signature file located in the Netscape directory on your system.

   At a command prompt type and enter:

   ```
   signtool -l
   ```

2. Use **signtool.exe** and the nickname in the command to create a new signed JAR file. Note that the nickname is case sensitive.

   At a command prompt type and enter:

   ```
   signtool.exe -k nickname -Z c:\newjarfilename.jar -d:\keys\
      c:\yourfiles_directory
   ```

   where:

   `-k nickname` is the nickname of the certificate in the **cert7.db** file you found in step 1.,

   `-Z c:\newjarfilename.jar` is the name of the JAR file you are signing,

   `-d:\keys\` is the name of the directory in which **cert7.db** and **key3.db** are located (they must be the same keys used in the Program Files/Netscape/Users/username directory), and

   `c:\yourfiles_directory` is the name of the directory that contains the JAR file.

   `yourfiles_directory` must contain your classes to JAR up. If you want the JAR file to include the Formula One classes, unjar F1J7Swing.jar into this directory as well.

Preparing an Applet for Internet Explorer

For Internet Explorer, all of the classes – including the classes for Formula One for Java and the classes you created for the applet – must be packaged in a CAB file. The CAB file is the file that receives the digital signature.

To begin, all class files must be extracted from JAR files.

➤ **To decompress classes in JAR files:**

At a command prompt, type:

```
jar -xvf jarfile.jar
```

where `jarfile` is the name of the file you want to decompress. The classes are decompressed and placed in the current directory.

Two programs from the Microsoft Internet SDK – **cabarc.exe** and **signcode.exe** – create and sign the CAB file. You can download these programs from the Microsoft website at www.microsoft.com/msdn/sdk. Look for the Internet Client SDK.

Creating the CAB File

Create the CAB file you will embed in an HTML document with **cabarc.exe**, using the following steps:

➤ **To create a CAB file:**

1. At a command prompt, type:

```
cabarc.exe -P -p \your_dir\ -r -s 6144 n cab_name.cab
    c:\your_dir\com\f1j\*.* + c:\your_dir\applet_classes.class
```

   where:

   `cab_name.cab` is the name of the CAB file you are creating,

   `-P` strips the specified directory structure from the Formula One for Java classes and the other classes in the applet,

   `-p \your_dir\` preserves the path name to the directory

   `-r` recurses the directories, and

   `-s` reserves space for digital signing.

   cabarc.exe creates a CAB file with the Formula One for Java classes and the applet classes you appended at the end of the command.

2. If you want to include the Formula One Java classes in the same CAB file, change the line to read:

```
cabarc.exe -p -P \your_dir\ -r -s 6144 n cab_name.cab
    c:\your-dir\com\f1j\*.* + c:\class1.class c:\class2.class
```

Signing the CAB File

The CAB file created in the previous section can be digitally signed with **signcode.exe**, using the following steps.

Note This procedure assumes you have obtained a digital signature and key information from a signing authority.

➤ **To digitally sign a CAB file:**

At a command prompt, type:

```
signcode.exe -j javasign.dll -jp low -spc myspecfile.spc
    -v mypriv.pvk cab_name.cab
```

where:

`-j` tells the program to use Java permission,

`-jp` sets the permission level for the CAB file,

`-spc myspecfile.spc` specifies the name of the file that contains your software publishing certificate,

`-v mypriv.pvk` is the name of the file that contains your private key information, and

`cab_name.cab` is the name of the CAB file you are signing.

Internet Explorer Permission Levels

The `-jp` option of the **signcode.exe** program allows you to set the permission level of the CAB file. Internet Explorer recognizes three permission levels: low, medium, and high. The following table describes how the permission level assigned to an applet corresponds with the security settings in Internet Explorer:

Applet permission set as:	Internet Explorer security setting and action required by user when applet is downloaded to browser:		
	Low	**Medium**	**High**
Low	None	Approval required	Approval required
Medium	None	None	Approval required
High	None	None	None

Checking the CAB File

After digitally signing the CAB file, you can check the permission information to make certain the file was created correctly.

➤ **To check a digitally signed CAB file:**

At the command prompt, type:

```
chkjava.exe cab_name.cab
```

where `cab_name.cab` is the name of the CAB file you want to check.

The **chkjava.exe** program is also provided in the Microsoft Internet Client SDK.

Deploying Signed Applets

➤ **To deploy the signed applet:**

1. Reference the JAR file from the HTML page using ARCHIVE=xyz.jar in the APPLET tag.
2. Copy the JAR file and HTML page to your web server.

Common Signing Mistakes & Solutions

Bad Magic Number error: One of the class files is either corrupt or the syntax for signing the JAR file was wrong. Check to make sure all filenames and paths are correct and that the appropriate "\" characters are present.

Cannot Find a .class file: Check the HTML code to make sure the files are in the correct directory. Also make sure the .class file is in the JAR or CAB file and does not have a path name (like c:) in the file.

Security errors: Either the file is not properly signed or you have not included the extra PrivilegeManager information for Netscape in your code.

Cannot find Javasign.dll: Javasign.dll is a supporting file for **signcode.exe** and must be in the same directory.

CHAPTER 5

Data Access Using JDBC and InfoBus

This chapter focuses on accessing databases with Formula One for Java using Java Database Connectivity and InfoBus technologies. The following topics are covered.

Formula One for Java and JDBC

Formula One for Java provides access to data stored in structured query language (SQL) databases through the Java Database Connectivity (JDBC) Standard. Through JDBC classes, Formula One for Java can handle database connections, SQL statements, and result sets, allowing active exchange of data between workbooks and databases.

JDBC Drivers

To connect Formula One for Java to an SQL database, you must have the JDBC drivers for your database management system (DBMS) loaded on your system. In your application or applet, Java classes are used to load the drivers. You must load the drivers before you attempt to make a connection from Formula One for Java.

Consult your development environment's documentation for information about loading and accessing JDBC drivers from your system. Sun's web site at www.java.sun.com has more information about the JDBC API, drivers and software available for downloading to support the JDBC specification.

The following example loads a JDBC driver that provides a bridge to open database connectivity (ODBC) database drivers:

```
// load a JDBC-ODBC driver
try {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
}
catch (ClassNotFoundException e) {
    jb.messageBox("Driver Class Not Found: "+ e.getMessage(),
                  "Error",jb.OK);
    return;
}
```

Connecting to a Database

To create a connection to a database, use the **getConnection** method from Java's DriverManager class. The **getConnection** method returns an instance of the Java connection class. The following example performs this task where the database name is "jdbcdb," the login is "Admin," and the password is "null."

```
// create connection to database
java.sql.Connection sqlCon = null;
try {
    sqlCon = java.sql.DriverManager.getConnection("jdbc:odbc:jdbcdb",
                                                  "Admin", "");
}
catch (Exception e) {
    jb.messageBox ("Connection Failed: "+ e.getMessage(),
                   "Error", jb.OK);
    return;
}
```

Using the JDBC Class

Formula One for Java provides a JDBC class that includes methods for querying an SQL database, directing the result set into a workbook, and updating a database from workbook data. The JDBC constructor can create an instance of the Formula One for Java JDBC class. Once the class is instantiated, you can call the methods in the class to invoke the Query dialog box, to populate a workbook with a result set, or to bind parameters.

The following example creates an instance of the Formula One for Java JDBC class:

```
com.f1j.jdbc.JDBC jdbc = new
        com.f1j.jdbc.JDBC(jb.getBook().getSheet(1));
```

Querying a Database

After creating a connection to a database, you can query the database and place the result set in a workbook. The query can be fed to the database in an SQL string or through a query object via Formula One for Java's JDBC QueryObj class.

Regardless of the method used to feed the query to a database, you must first create a statement, which prepares and allocates space for the query. To do this, use the **createStatement** method from Java's connection class. This method returns an instance of Java's statement class.

Using a Query Object

Formula One for Java's JDBC QueryObj class provides a way to submit a query to an SQL database. Query objects created by this class include a number of properties that define the query and how results from the query are displayed in a workbook. The following table lists and defines the properties of a Formula One for Java query object.

Query Object Properties and Methods

Property/Method	Type	Description
setQueryStr	String	Defines the query string for the object.
setAutoColNames	Boolean	When set to true, uses the database table field names for column headers when data is placed in a Formula One for Java workbook.
setAutoColFormats	Boolean	When set to true, the formatting of queried data (i.e., currency, dates, time) is automatically applied to that data when placed in a workbook.
setAutoColWidths	Boolean	When set to true, the width of workbook columns is automatically resized to accommodate the widest data in columns of queried data.

Property/Method	Type	Description
setAutoMaxRC	Boolean	When set to true, the maximum number of workbook rows and columns is set to match the number of table rows and columns in the queried data.
setStartRow	Integer	Sets the starting workbook row for placement of queried data: 0 sets the starting row as row 1, 1 sets the starting row as row 2, and so forth.
setStartCol	Integer	Sets the starting workbook column for placement of queried data: 0 sets the starting column as column A, 1 sets the starting column as column B, and so forth.
setColNamesInRow	Integer	Sets the row in which column names are placed: -1 places column names in the column header row, 0 places column names in row 1, 1 places column names in row 2, and so forth.

Each property of the query object has a corresponding method in the Formula One for Java JDBC class that allows you to obtain the current setting of the property.

Property/Method	Current Setting Method
setQueryStr	**getQueryStr**
setAutoColNames	**isAutoColNames**
setAutoColFormats	**isAutoColFormats**
setAutoColWidths	**isAutoColWidths**
setAutoMaxRC	**isAutoMaxRC**
setStartRow	**getStartRow**
setStartCol	**getStartCol**
setColNamesInRow	**getColNamesInRow**

Creating a Query Object

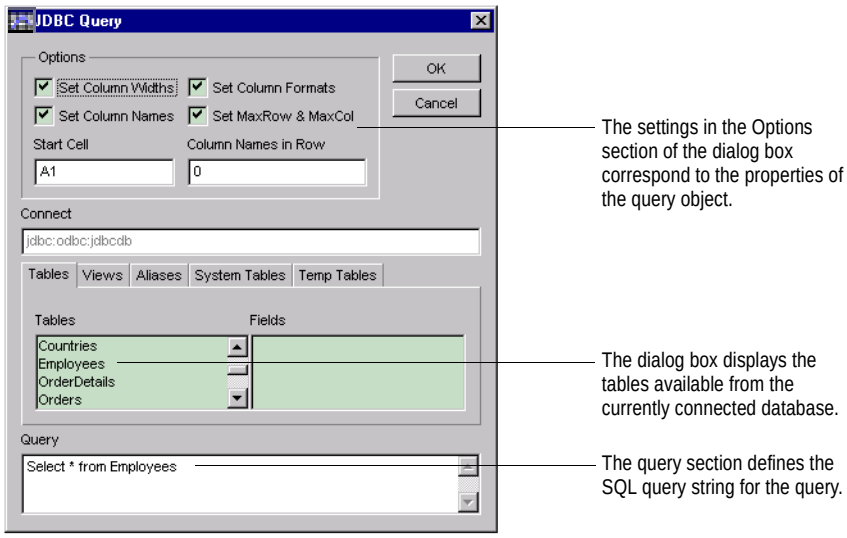
The following example shows how to create a Formula One for Java query object, set the object properties, and create a statement using the query object to define the query. The returned result set is placed in a workbook using Formula One for Java's **populateGrid** method.

```
com.f1j.jdbc.JDBCQueryObj queryObj = new com.f1j.jdbc.JDBCQueryObj();

queryObj.setQueryStr("Select * From Employees");
queryObj.setAutoColNames(true);
queryObj.setAutoColFormats(true);
queryObj.setAutoColWidths(true);
queryObj.setAutoMaxRC(true);
queryObj.setStartRow(0);
queryObj.setStartCol(0);
queryObj.setColNamesInRow(-1);
```

```
java.sql.Statement stmt = null;
java.sql.ResultSet rs = null;

try {
    stmt = sqlCon.createStatement();
    if (queryObj.getQueryStr() !=null) {
       rs = stmt.executeQuery(queryObj.getQueryStr());
       jdbc.populateGrid(rs, queryObj);
       rs.close();
    }
    stmt.close();
    sqlCon.close();
}
```

Using the Query Dialog Box

The JDBC Query dialog box provides a way to solicit user input for setting the parameters of a query. You can invoke the JDBC Query dialog box by calling the **JDBCDlg** constructor from com.f1j.swing.JDBC.JDBCDlg.



When invoked, the dialog box displays default settings. You can also use a Formula One for Java query object to set the display settings. Clicking the OK button sends any changes made in the dialog box to the specified query object.

The following example invokes the JDBC Query dialog box and uses the dialog box information to define a query that returns data to a workbook.

```
    try{
          Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      }
      catch (ClassNotFoundException e){
          System.out.println(e.getMessage());
```

```
        return;
    }

    java.sql.Connection sqlCon = null;
    try{
    sqlCon =
      java.sql.DriverManager.getConnection("jdbc:odbc:Southcreek",
      "","");

      com.f1j.ss.Sheet          sheet = m_jBook.getBook().getSheet(0);
    com.f1j.jdbc.JDBC         jdbc = new com.f1j.jdbc.JDBC(sheet);
    com.f1j.jdbc.JDBCQueryObj queryObj = new
      com.f1j.jdbc.JDBCQueryObj();
    String strQry = "Select * from CustomerOrders";
    queryObj.setQueryStr(strQry);
    queryObj.setColNamesInRow(-1);
    queryObj.setStartRow(0);
    queryObj.setStartCol(0);
    queryObj.setAutoColNames(false);
    queryObj.setAutoColFormats(false);
    queryObj.setAutoColWidths(false);
    queryObj.setAutoMaxRC(true);

    java.sql.ResultSet rs = null;
    java.sql.Statement stmt = sqlCon.createStatement();
      com.f1j.swing.jdbc.JDBCDlg jdbcDlg = new
      com.f1j.swing.jdbc.JDBCDlg(m_jBook, sqlCon, queryObj);
      jdbcDlg.setVisible(true);

      if (queryObj.getQueryStr() != null){
         String strQuery = queryObj.getQueryStr();
        rs = stmt.executeQuery(strQuery);
        jdbc.populateGrid(rs, queryObj);
          rs.close();
      }
    stmt.close();
      }
    catch (java.lang.Exception e){
        System.out.println(e.getMessage());
    }
}
```

Binding Parameters

The Formula One for Java JDBC class includes the **bindParameter** method, which allows you to take selected data from a worksheet and bind it to unknown parameters with a prepared statement. Working in conjunction with JDBC's **prepareStatement** method, the bound data is used in statements that select, insert, update, and delete data in a database. In addition to providing data for unknown parameters, **bindParameter** converts the worksheet data to a data type recognized by the database.

In the following example, the **bindParameter** method extracts data from a worksheet and updates the birth dates in the Employees table in a database. Date data, extracted from column three in the worksheet, updates employee birth dates; integer data, extracted from column one, matches employee ID numbers in the worksheet and table.

```
com.f1j.jdbc.JDBC jdbc = new
        com.f1j.jdbc.JDBC(jb.getBook().getSheet(1));

java.sql.Connection sqlCon = null;
try {
    sqlCon = java.sql.DriverManager.getConnection("jdbc:odbc:jdbcdb",
                                                  "admin", "");
}
catch (Exception ex) {
    System.out.println(ex.getMessage());
}

try {
    java.sql.PreparedStatement pSQL = sqlCon.prepareStatement(
        "Update Employees Set Birthdate = ? Where Employee ID = ?");

    for(int i = 0; i<=jb1.getLastRow(); i++) {
        jdbc.bindParameter(pSQL, 1, i, 2, java.sql.Types.TIMESTAMP);
        jdbc.bindParameter(pSQL, 2, i, 0, java.sql.Types.INTEGER);
        pSQL.executeUpdate ();
    }
    pSQL.close();
    sqlCon.close();
}
catch (java.sql.SQLException e) {
   System.out.println.(e.getMessage());
}
catch (com.f1j.util.F1Exception e) {
   jb1.messageBox(e.getMessage(), “Error”, jb1.OK);
   return;
}
```

Formula One for Java Data Access Using InfoBus

About InfoBus

InfoBus is a small Java API, designed jointly by Lotus Development Corporation and Sun Microsystems' JavaSoft division. InfoBus allows cooperating applets or Beans, on a Web page or in any other Java application, to communicate data to one another independent of the restrictions of the data accessed. Data exchange is based on information content.

Using InfoBus, developers program one type of InfoBus component (a data "*producer*") that makes a specific set of data available to other components. Another type (a data "*consumer*") waits for that particular set of data (identified by a specific data name or generic calculated name) to become available. The data consumer retrieves the data and delivers it for analysis or display within an application.

As an example, a producer JavaBean might make data from a worksheet named "results" available for retrieval. A consumer JavaBean, programmed to listen for data named "results," is sent in search of the data. When the consumer JavaBean locates the producer JavaBean with data of that name, the two JavaBeans connect and exchange the data.

InfoBus uses generic interfaces specific to InfoBus (rather than to operating systems, database types, or software) making it possible for applications that are not even aware of each other to exchange data.

InfoBus uses a number of components with specific roles.

- Data "*producers*" retrieve data from a database or another source and make it available to other components.
- Data "*consumers*" access data from the InfoBus to use in analysis or display.
- Data "*controllers*" intercept events, provide routing, and otherwise control interaction between data producers and data consumers.
- Data "*items*" contain the data retrieved by a producer while the data is on its way to the data consumer.

Note that applets can also function as both producers and consumers. One worksheet applet might both retrieve data from a database or spreadsheet as a consumer and then provide data to a charting applet, for example.

The main advantage of using InfoBus is that it separates the process of retrieving, accessing and displaying data from the restrictions of data types and locations. This enables disparate applications to freely interchange data.

Data Item Data Access Types used in Formula One for Java

InfoBus provides a number of data access types (via Java interfaces) for a data item. Data access types instruct the Java component serving as a data consumer about the attributes of the data to make available to or retrieve from the InfoBus. The following types of data items work with Formula One for Java.

ImmediateAccess

Use the ImmediateAccess type with data in the form of a single value or object. Two examples:

- You can publish a single cell (vs. a range) on the InfoBus using Formula One for Java as a producer. The consumer component will read in that single item.
- You can have another component publish one single piece of data which Formula One for Java will read in as an ImmediateAccess data type.

ArrayAccess

Use the ArrayAccess type with data in the form of multi-dimensional arrays of values. Formula One for Java can publish a 2D array of data as a data item of the type ArrayAccess.

ScrollableRowsetAccess

Use the ScrollableRowsetAccess interface type with data that represents a set of rows from a database. Formula One for Java receives the data put on the Infobus by a producer applet. ScrollablerowsetAccess allows you to perform various database operations like querying and updating data, etc.

Note Formula One for Java accepts only the ScrollableRowsetAccess interface (not RowsetAccess).

The table below illustrates which interface types Formula One for Java supports as consumers, which as producers and which as both.

Data Consumers:	ScrollableRowsetAccess, ArrayAccess, ImmediateAccess
Data Producers:	Array Access, ImmediateAccess

More information about InfoBus technology is available on line at the Lotus eSuite DevSite, http://esuite.lotus.com and Sun's JavaBeans software website, http://java.sun.com. The complete InfoBus Specification is available at http://www.javasoft.com.

Using Formula One for Java with InfoBus

To link data between InfoBus enabled applets, place the name of an InfoBus item in the applet parameters of the HTML page. All InfoBus-enabled Java components respond to requests from other InfoBus-enabled Java components.

To pass data via the InfoBus from a Formula One for Java applet to an Infobus consumer applet, you name the range containing that data in the spreadsheet. The spreadsheet automatically creates an ArrayAccess data item with the range name as the data item name. You place the consumer applet on an HTML page with a <PARAM> tag specifying the spreadsheet range from which to receive its data. The consumer applet automatically retrieves a chart data set from the data item and displays the data.

The following code presents examples of Formula One for Java in both the role of producer and the role of consumer.

Formula One for Java as an InfoBus Data Producer Example

In this example, Formula One for Java serves as the data producer. When Formula One for Java acts as the data producer, it provides data from a Formula One for Java worksheet. A charting program serves as the data consumer, retrieving and displaying the data in the chart.

When Formula One for Java is a *producer*, you need to specify the parameter tags in your HTML page. When Formula One for Java is a *consumer*, you either need to specify the parameter tags in your HTML page or use their corresponding API in code.

Note All of the examples in this Guide run on web servers that support the Java 2 Platform (available online at java.sun.com/products) and JavaServer Pages (also available on line at java.sun.com/products/jsp).

Example Code

This example uses a unique name (abc) and data range (Sheet1!$C$2:$D$4) for the data provided by Formula One for Java.

```
<HTML>
<HEAD>
<TITLE>Autogenerated HTML</TITLE>
</HEAD>
<BODY>

<CENTER>

<P> USE A UNIQUE NAME "abc" AND DEFINE A RANGE FOR IT "Sheet1!$C$2:$D$4"
        </P>
```

```
<APPLET CODE=com.f1j.infobus.InfoBusBook WIDTH=580 HEIGHT=215>
    <PARAM NAME="workbook" VALUE="test.vts">
    <PARAM NAME="outputDataItemName1" VALUE="abc">
    <PARAM NAME="outputDataFormula1" VALUE="Sheet1!$C$2:$D$4">
</APPLET>

<BR><BR>

<applet code=lotus.chart.Chart width=580 height=340>
    <PARAM NAME ="perspective" VALUE = "3D">
    <PARAM NAME ="seriesLayout" VALUE = "Beside">
    <PARAM NAME="ParseDirection" VALUE="column">
    <PARAM NAME="dataInputItemName" VALUE="abc">
    <PARAM NAME="backgroundColor" VALUE="white">
    <PARAM NAME ="seriesLabelsPosition"
        VALUE="above, below, above, below">
    <PARAM NAME="infoBusDebugEnabled" VALUE="true">
</applet>

</CENTER>

</BODY>
</HTML>
```

Note To use a defined data name, delete the outputDataFormula1 line and change "value" in the outputDataItemName1 line to the defined name of your data source. Defined names must be identical to pass from the producer to the consumer.

Formula One for Java as an InfoBus Data Consumer Example

In this example Formula One for Java, serving as the data consumer, retrieves and displays the data provided by the data producer, an Oracle database.

Example Code

```
try {
    com.f1j.infobus.InfoBusBook m_infoBook = new
        com.f1j.infobus.InfoBusBook();

     m_infoBook.setInfoBusName("oracle");
     m_infoBook.setCommitMode(
         com.f1j.infobus.InfoBusBook.kCommitAfterChanges);
     m_infoBook.setInputDataItem(
         "infobus:/oracle/Session1/rowSetInfo 1", "Sheet1!$A$1:$N$100");

}
catch (com.f1j.util.F1Exception eF1) {
```

```
    eF1.printStackTrace();
}
```

where:

`m_infoBook.setInfoBusName(“oracle”);` defines the name of the InfoBus. Both consumers and producer need to reference the identical InfoBus name.

`m_infoBook.setCommitMode(com.f1j.infobus.InfoBusBook.kCommitAfterChanges);` means that changes to Rowset will be committed after editing.

`m_infoBook.setInputDataItem("infobus:/oracle/Session1/rowSetInfo1", "Sheet1!$A$1:$N$100");` describes where the data will reside inside a Formula One for Java worksheet.

CHAPTER 6

Sharing Spreadsheets

You can connect (attach) Multiple JBook classes to the same JBook model (underlying data) within an application or applet. This flexibility allows different parts of the workbook to be displayed in different windows (JBooks) within an application.

This chapter discusses and presents examples and illustrations of controlling the display of workbook areas using attach methods and of reading a workbook directly into a JBook in an applet. The following topics are covered.

Rules for Attaching JBooks

When building applets or applications showing multiple JBooks of data from one workbook, you must **attach** JBooks. When attaching JBooks, remember these important rules:

- A JBook can be connected to only one workbook at a time.
- A workbook can be attached to multiple JBooks.
- A workbook is always attached to a JBook. It ceases to exist if it is not attached to a view.

Attach JBooks using one of the two attach methods described in this chapter, `attach(JBook jb)` or `attach(String workbookName)`.

Example Applets Using Attach Methods

The following examples create an applet showing the dog population for different breeds of dogs with all the data contained in one workbook. The first example uses the **attach**(JBook *jb*) method. The second uses the **attach**(String *workbookName*) method.

Note You could also build a similar view for each type of dog, and attach each JBook to another (hidden) JBook that opens a workbook filled with data.

The code for the following buttons is displayed in **boldface type** in the example application code.

- **Workbook Designer button.** This button launches the Workbook Designer from the hidden JBook, allowing users to open a worksheet.
- **Labradors, Poodles, and Cockers buttons.** These buttons accomplish the following tasks in the example:

  Attach the *Labrador*, *Poodle*, or *Cocker* JBook (jb2, jb3, and jb4) to the hidden JBook (jb1). The hidden view contains the worksheet data.

  Limit the area of the worksheet that is displayed.

  Prevent the formula bar from displaying, thus limiting the user's ability to change the data.

- **The setVisible method.** This method is used to hide View 1

Note Set the **setAllowDesigner** method to false to prevent the user from launching the Workbook Designer from the visible views.

Attach JBook Applet Example

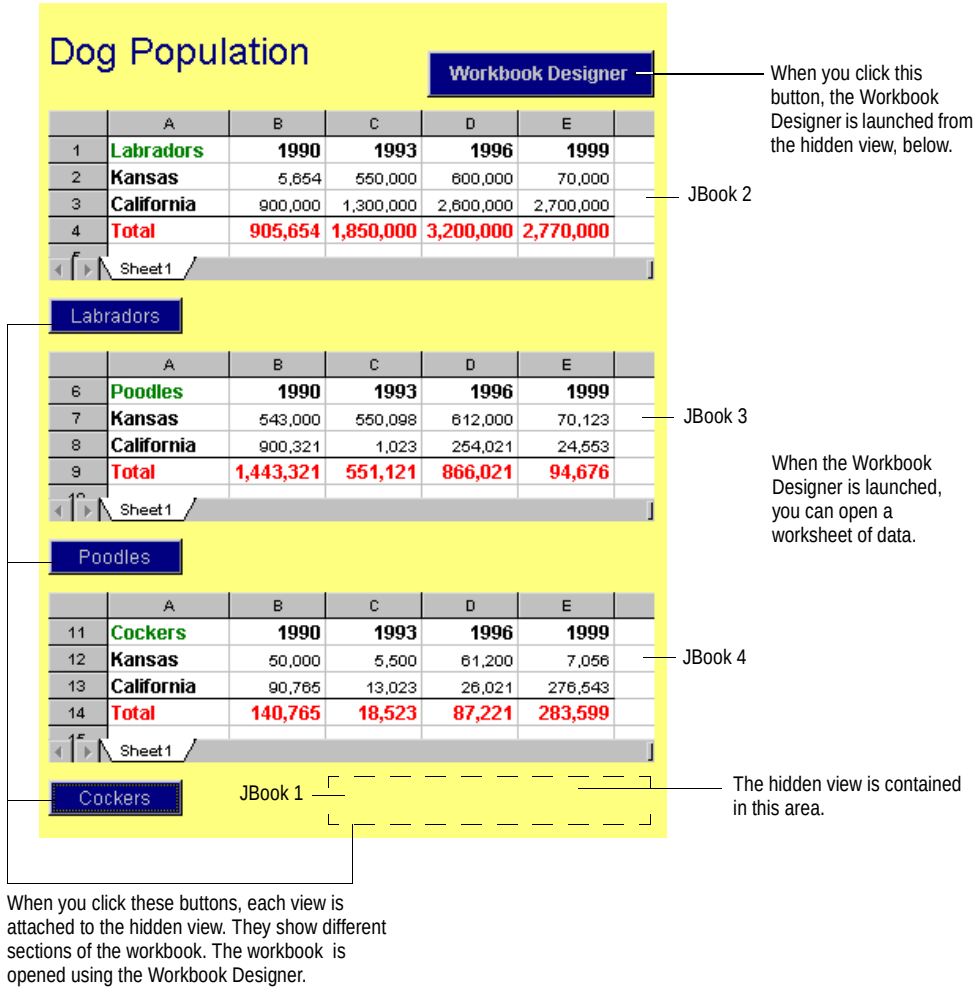
This attach method attaches the current JBook to the specified JBook's workbook using the display settings of the specified JBook's workbook. Any display properties set in the workbook after attachment are not propagated between the JBooks.

```
// HTML Required to load this applet into the Netscape Plug-in
//
// <EMBED type="application/x-java-applet;version=1.2"
//  java_CODE = "dog_popu.class" archive = "F1J7Swing.jar"
//  WIDTH=800 HEIGHT=700 ><NOEMBED>
//</NOEMBED></EMBED>
//

import java.awt.*;
import javax.swing.*;

import com.f1j.swing.JBook;

public class dog_popu extends JApplet
{
    public void init()
    {
        getContentPane().setLayout(null);
        setSize(428,546);
        setFont(new Font("Helvetica", Font.PLAIN, 12));
        setBackground(new Color(16777088));
        try {
            jb2 = new com.f1j.swing.JBook();
            jb2.setBounds(12,74,384,108);
            getContentPane().add(jb2);
        } catch (Exception e) { }
        try {
            jb3 = new com.f1j.swing.JBook();
            jb3.setBounds(12,227,384,108);
            getContentPane().add(jb3);
        } catch (Exception e) { }
        try {
            jb1 = new com.f1j.swing.JBook();
            jb1.setVisible(false);
            jb1.setBounds(120,492,276,36);
            getContentPane().add(jb1);
        } catch (Exception e) { }
        try {
            jb4 = new com.f1j.swing.JBook();
            jb4.setBounds(12,380,384,108);
            getContentPane().add(jb4);
        } catch (Exception e) { }

        label1 = new javax.swing.JLabel("Dog Population");
        label1.setBounds(12,12,180,51);
```

```
        label1.setFont(new Font("Helvetica", Font.PLAIN, 24));
        label1.setForeground(new Color(-16777088));
        getContentPane().add(label1);

        button1 = new javax.swing.JButton();
        button1.setActionCommand("button");
        button1.setLabel("Labradors");
        button1.setBounds(12,193,85,23);
        button1.setForeground(new Color(12632256));
        button1.setBackground(new Color(-16777088));
        getContentPane().add(button1);

        button2 = new javax.swing.JButton();
        button2.setActionCommand("button");
        button2.setLabel("Poodles");
        button2.setBounds(12,346,85,23);
        button2.setForeground(new Color(12632256));
        button2.setBackground(new Color(-16777088));
        getContentPane().add(button2);

        button3 = new javax.swing.JButton();
        button3.setActionCommand("button");
        button3.setLabel("Cockers");
        button3.setBounds(12,499,85,23);
        button3.setForeground(new Color(12632256));
        button3.setBackground(new Color(-16777088));
        getContentPane().add(button3);

        button4 = new javax.swing.JButton();
        button4.setActionCommand("button");
        button4.setLabel("Workbook Designer");
        button4.setBounds(252,36,144,30);
        button4.setFont(new Font("Dialog", Font.BOLD, 12));
        button4.setForeground(new Color(12632256));
        button4.setBackground(new Color(-16777088));
        getContentPane().add(button4);

        F1Action lF1Action = new F1Action();
        button1.addActionListener(lF1Action);
        button2.addActionListener(lF1Action);
        button3.addActionListener(lF1Action);
        button4.addActionListener(lF1Action);

    }
    com.f1j.swing.JBook jb2;
    com.f1j.swing.JBook jb3;
    com.f1j.swing.JBook jb1;
    com.f1j.swing.JBook jb4;
    javax.swing.JLabel label1;
    javax.swing.JButton button1;
```

```
javax.swing.JButton button2;
javax.swing.JButton button3;
javax.swing.JButton button4;

class F1Action implements java.awt.event.ActionListener
{
    public void actionPerformed(java.awt.event.ActionEvent event)
    {
        Object object = event.getSource();
        if (object == button1)
            button1_Action(event);
        else if (object == button2)
            button2_Action(event);
        else if (object == button3)
            button3_Action(event);
        else if (object == button4)
            button4_Action(event);
    }
}
void button1_Action(java.awt.event.ActionEvent event)
{
    try {
        jb2.attach(jb1);
        jb2.setShowEditBar(false);
        jb2.setMinRow(0);
        jb2.setMaxRow(5);
        jb2.setMinCol(0);
        jb2.setMaxCol(6);
    }
    catch(com.f1j.util.F1Exception e) { }
}

void button2_Action(java.awt.event.ActionEvent event)
{
    try {
        jb3.attach(jb1);
        jb3.setShowEditBar(false);
        jb3.setMinRow(5);
        jb3.setMaxRow(10);
        jb3.setMinCol(0);
        jb3.setMaxCol(6);
        }
    catch(com.f1j.util.F1Exception e) { }
}
    void button3_Action(java.awt.event.ActionEvent event)
{
    try {
        jb4.attach(jb1);
        jb4.setShowEditBar(false);
        jb4.setMinRow(10);
```

```
            jb4.setMaxRow(15);
            jb4.setMinCol(0);
            jb4.setMaxCol(6);
        }
        catch(com.f1j.util.F1Exception e) { }
    }
    void button4_Action(java.awt.event.ActionEvent event)
    {
       jb1.launchDesigner();
    }
```



	A	B	C	D	E
1	Labradors	1990	1993	1996	1999
2	Kansas	5,654	550,000	600,000	70,000
3	California	900,000	1,300,000	2,600,000	2,700,000
4	Total	905,654	1,850,000	3,200,000	2,770,000

Sheet1

	A	B	C	D	E
6	Poodles	1990	1993	1996	1999
7	Kansas	543,000	550,098	612,000	70,123
8	California	900,321	1,023	254,021	24,553
9	Total	1,443,321	551,121	866,021	94,676

Sheet1

	A	B	C	D	E
11	Cockers	1990	1993	1996	1999
12	Kansas	50,000	5,500	61,200	7,056
13	California	90,765	13,023	26,021	276,543
14	Total	140,765	18,523	87,221	283,599

Sheet1

Attach Workbook String Applet

The previous applet used the **attach(JBook *jb*)** method to attach the views. When this method is used, any display properties set in the workbook after attachment are not propagated between the views.

If you want to use data from the workbook, but you do not want the settings specified in the workbook to apply to the current view, use the **attach(String *workbookName*)** method. When you use this method, the **setGroup** method is also required. The **setGroup** method specifies the views to attach in a group. A view can find another view only if they are in the same group.

If you want to read a workbook directly into a view, use the **read** method. However, if you are creating an applet for a web page, you should use the **readURL** method. Please note that, if the filename begins with *http:*, *https:*, or *file:*, **read** calls **readURL**.

The following table lists the methods (display settings) propagated between views.

Method Name	Description
setShowFormulas	Displays formulas in place of cell values.
setShowGridlines	Determines whether gridlines are displayed.
setShowRowHeading	Indicates whether row headings are displayed.
setShowColHeading	Determines whether column headings are displayed.
setShowZeroValues	Indicates whether zero value cells are displayed.
setBackColor	Sets the background color of the view.
setExtraColor	Decides the color of the view area outside the workbook.
setMinCol	Sets the first column displayed in the active worksheet.
setMinRow	Sets the first row displayed in the active worksheet.
setMaxCol	Sets the last column displayed in the active worksheet.
setMaxRow	Sets the last row displayed in the active worksheet.
setViewScale	Determines the current display scale for a workbook.
setShowSelections	Indicates whether selections are displayed.
setShowTabs	Determines if tabs are displayed.
setShowHScrollBar	Decides whether the horizontal scroll bar is displayed.
setShowVScrollBar	Decides whether the vertical scroll bar is displayed.
setAllowResize	Determines whether you can resize rows and columns by dragging them.
setAllowFillRange	Indicates whether you can fill a worksheet range by dragging.
setAllowMoveRange	Determines whether you can move ranges by dragging them.
setAllowSelections	Indicates whether you can select ranges and objects.
setAllowObjectSelections	Determines whether you can select objects at runtime.
setAllowAutoFill	Indicates whether auto fill is enabled.
setAllowFormulas	Determines whether you are allowed to enter or edit formulas.

Method Name	Description
setAllowInCellEditing	Indicates whether in-cell editing is allowed.
setAllowArrows	Decides whether the arrow keys can reposition the active cell.
setAllowTabs	Determines whether you can use the **TAB** key to reposition the active cell in a selected range.
setAllowDelete	Indicates whether the delete key clears selections.
setAllowEditHeadings	Decides whether row, column, and top left headers can be edited.
setEnterMovesDown	Determines whether pressing the **ENTER** key moves the active cell down to the next cell.
setRowMode	Indicates the row mode status for a view.
setFixedCols	Sets how many columns to fix at the left edge of the active worksheet.
setFixedRows	Sets how many rows to fix at the top of the active worksheet.
setAllowDesigner	Decides whether the Workbook Designer can be launched at runtime.

Attach Workbook Example

This attach method searches for a workbook with the specified name and attaches the current JBook to it. This method starts with a default view using only the data from the specified workbook. Settings specified in the workbook are not applied to the current JBook. Both workbooks must be in the same group.

The following code example displays the **attach(String *workbookName*)**, **setGroup**, and **read** methods in boldface type.

```
// HTML Required to load this applet into the Netscape Plug-in
//
// <EMBED type="application/x-java-applet;version=1.2"
//  java_CODE = "dog_popu2.class" archive = "F1J7Swing.jar"
//  WIDTH=800 HEIGHT=700 ><NOEMBED>
//</NOEMBED></EMBED>
//

import javax.swing.*;
import java.applet.*;
import java.awt.*;
import com.f1j.swing.JBook;

public class dog_popu2 extends JApplet
{
    public void init()
    {
        getContentPane().setLayout(null);
        setSize(428,546);
        setFont(new Font("Helvetica", Font.PLAIN, 12));
```

```
setBackground(new Color(-128));
try {
    jb2 = new com.f1j.swing.JBook();
    jb2.setGroup("dogs");
    jb2.setBounds(12,74,384,108);
    getContentPane().add(jb2);
} catch (Exception e) { }
try {
    jb3 = new com.f1j.swing.JBook();
    jb3.setGroup("dogs");
    jb3.setBounds(12,227,384,108);
    getContentPane().add(jb3);
} catch (Exception e) { }
try {
    jb1 = new com.f1j.swing.JBook();
    jb1.setGroup("dogs");
    jb1.setVisible(false);
    jb1.setBounds(120,492,276,48);
    getContentPane().add(jb1);
} catch (Exception e) { }
try {
    jb4 = new com.f1j.swing.JBook();
    jb4.setGroup("dogs");
    jb4.setBounds(12,380,384,108);
    getContentPane().add(jb4);
} catch (Exception e) { }

label1 = new javax.swing.JLabel("Dog Population");
label1.setBounds(12,12,180,51);
label1.setFont(new Font("Helvetica", Font.PLAIN, 24));
label1.setForeground(new Color(-16777088));
getContentPane().add(label1);

button1 = new javax.swing.JButton();
button1.setActionCommand("button");
button1.setLabel("Labradors");
button1.setBounds(12,193,85,23);
button1.setForeground(new Color(12632256));
button1.setBackground(new Color(-16777088));
getContentPane().add(button1);

button2 = new javax.swing.JButton();
button2.setActionCommand("button");
button2.setLabel("Poodles");
button2.setBounds(12,346,85,23);
button2.setForeground(new Color(12632256));
button2.setBackground(new Color(-16777088));
getContentPane().add(button2);

button3 = new javax.swing.JButton();
button3.setActionCommand("button");
```

```
        button3.setLabel("Cockers");
        button3.setBounds(12,499,85,23);
        button3.setForeground(new Color(12632256));
        button3.setBackground(new Color(-16777088));
        getContentPane().add(button3);

        button4 = new javax.swing.JButton();
        button4.setActionCommand("button");
        button4.setLabel("Read Workbook");
        button4.setBounds(252,36,144,30);
        button4.setFont(new Font("Dialog", Font.BOLD, 12));
        button4.setForeground(new Color(12632256));
        button4.setBackground(new Color(-16777088));
        getContentPane().add(button4);

        F1Action lF1Action = new F1Action();
        button1.addActionListener(lF1Action);
        button2.addActionListener(lF1Action);
        button3.addActionListener(lF1Action);
        button4.addActionListener(lF1Action);
    }

    com.f1j.swing.JBook jb2;
    com.f1j.swing.JBook jb3;
    com.f1j.swing.JBook jb1;
    com.f1j.swing.JBook jb4;
    javax.swing.JLabel label1;
    javax.swing.JButton button1;
    javax.swing.JButton button2;
    javax.swing.JButton button3;
    javax.swing.JButton button4;

    class F1Action implements java.awt.event.ActionListener
    {
        public void actionPerformed(java.awt.event.ActionEvent event)
        {
            Object object = event.getSource();
            if (object == button1)
                button1_Action(event);
            else if (object == button2)
                button2_Action(event);
            else if (object == button3)
                button3_Action(event);
            else if (object == button4)
                button4_Action(event);
        }
    }
    void button1_Action(java.awt.event.ActionEvent event)
    {
        jb1.setWorkbookName("Dog1");
        String GroupName;
```

```
            GroupName = "doggroup1";
            try {
                jb2.setShowEditBar(false);
                jb2.attach("Dog1");
                jb2.setMinRow(0);
                jb2.setMaxRow(5);
                jb2.setMinCol(0);
                jb2.setMaxCol(6);
            }
            catch(com.f1j.util.F1Exception e) { }
        }
        void button2_Action(java.awt.event.ActionEvent event)
        {
            jb1.setWorkbookName("Dog1");
            String GroupName;
            GroupName = "doggroup2";
            try {
                jb3.setShowEditBar(false);
                jb3.attach("Dog1");
                jb3.setMinRow(5);
                jb3.setMaxRow(10);
                jb3.setMinCol(0);
                jb3.setMaxCol(6);
            }
            catch(com.f1j.util.F1Exception e) { }
        }
        void button3_Action(java.awt.event.ActionEvent event)
        {
            jb1.setWorkbookName("Dog1");
            String GroupName;
            GroupName = "doggroup3";
            try {
                jb4.setShowEditBar(false);
                jb4.attach("Dog1");
                jb4.setMinRow(10);
                jb4.setMaxRow(15);
                jb4.setMinCol(0);
                jb4.setMaxCol(6);
            }
            catch(com.f1j.util.F1Exception e) { }
        }
        void button4_Action(java.awt.event.ActionEvent event)
        {
            try {
                jb1.read("c:\\dogs");
            }
            catch(Exception e) { }
        }
    }
```

Tidestone

CHAPTER 7

Creating Add-In Functions

This chapter provides information about add-in functions—small Java classes that extend the capabilities of Formula One for Java. The examples presented here extend Formula One for Java's functions by adding concatenation (Example 1), concatenation with an efficient way to handle threading (Example 2), and a custom version of the SUM function (Example 3). The following topics are covered.

Implementing Add-In Functions

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

- include a static initializer that makes sure an instance of the function is created
- include a private constructor that allows only one instance to be created
- in that constructor, you must call the superclass constructor and pass the name of the function, the minimum number of arguments, and the maximum number of arguments
- override the *evaluate* method in *com.f1j.addin.Func*

To use the add-in function, include it in your class path. The function names of add-ins are case-sensitive.

Example Add-Ins

The following examples demonstrate how to implement add-in functions. Examples 1 and 2 concatenate (join together into a single string) text strings and demonstrate different ways of handling threading issues. Example 3 demonstrates how to handle cell references and ranges in an add-in function.

Example 1: Simple Function

```
/**
 * Add-in functions must be derived from com.f1j.addin.Func
 */
public class IsEven extends com.f1j.addin.Func {
    /**
     * This static initializer creates the one and only instance of
     * isEven and adds it to com.f1j.addin.Func's add-in function list.
     */
    static {
        new IsEven();
    }

    /**
     * Make this private so that only the one instance is created.
     */
    private IsEven() {
        super("IsEven", 1, 1);
    }

    /**
     * Override com.f1j.addin.evaluate(...) to do the work.
     */
    public void evaluate(com.f1j.addin.FuncContext fc) {
        com.f1j.addin.Value value = fc.getArgument(0);
        if (value.checkNumber())
            fc.setReturnValue((((long)value.getNumber()) & 1) == 0);
    }
}
```

Example 2: Concatenation

```
/**
 * Add-in functions must be derived from com.f1j.addin.Func
 */
public class MyConcat1 extends com.f1j.addin.Func {

    /**
     * There are no thread problems with having a static final member
     */
    final static short errInvalidValue = 3;

    /**
     * The static initializer creates the one and only instance of
     * MyConCat1 and adds it to com.f1j.calc.Func's function list.
     */
     static {
        new MyConcat1();
     }

     /**
      * Make this private so that only the one instance is created.
      */
     private MyConcat1() {
        // we can allow 1-30 arguments
        super( "MyConcat1", 1, 30 );
     }

     /**
      * Override the com.f1j.addin.evaluate(...) to do the work.
      */
     public void evaluate( com.f1j.addin.FuncContext fc) {
        int argCount = fc.getArgumentCount();

        /**
         * Since we do not "synchronize" we must get a new instance each
         * time to be thread safe. The reference must be on the stack.
         * In most conditions, this will hurt performance. See MyConcat2
         * for a generally faster way to do things.
         */
        StringBuffer accum = new StringBuffer();

        for ( int ii=0; ii < argCount; ii++ ) {
            com.f1j.addin.Value val = fc.getArgument(ii);
            if ( val.checkText() ) {
                accum.append( val.getText() );
            }
            else {
                fc.setReturnValue( errInvalidValue );
                return;
```

```
            }
        }
        fc.setReturnValue( accum.toString() );
    }
}
```

Example 3: Concatenation and Threading

The following example add-in function, like the example above, concatenates text strings. It also demonstrates a more efficient way to handle threading considerations.

```
/**
 * Add-in functions must be derived from com.f1j.addin.Func
 */
public class MyConcat2 extends com.f1j.addin.Func {

    /**
     * We can have static final members with no thread problems.
     */
    final static short errInvalidValue = 3;

    /**
     * Since we have a member than can change, we must worry about
     * thread safety. This one variable will be shared by all threads.
     * Note: This class should be faster because "new" and memory
     * allocations are slower than "synchronized", especially if
     * multiple instance variables are involved.
     */
    StringBuffer m_accum = new StringBuffer();

    /**
     * The static initializer assures that the one and only
     * instance of MyConCat1 is created and added to com.f1j.calc.Func's
     * list of add-in functions.
     */
     static {
        new MyConcat2();
     }

     /**
      * Make this private so that only the one instance is created.
      */
     private MyConcat2() {
        // we can allow 1-30 arguments
        super( "MyConcat2", 1, 30 );
     }

     /**
      * Override the com.f1j.addin.evaluate(...) to do the work
      *
      * We must use the "synchronized" qualifier to ensure that
```

```
         * a 2nd thread does not attempt to modify "m_accum" while we
         * are using it.
         */
        public synchronized void evaluate( com.f1j.addin.FuncContext fc) {

            m_accum.setLength(0);

            int argCount = fc.getArgumentCount();

            for ( int ii=0; ii < argCount; ii++ ) {
                com.f1j.addin.Value val = fc.getArgument(ii);
                if ( val.checkText() ) {
                    m_accum.append( val.getText() );
                }
                else {
                    fc.setReturnValue( errInvalidValue );
                    return;
                }
            }
            fc.setReturnValue( m_accum.toString() );
        }
}
```

Example 4: Ranges and Explicit Values

A custom version of the SUM function, the final example demonstrates how to handle cell references and ranges in an add-in function.

```
/**
 * Complex add-in to demonstrate additional capabilities.
 * The performance of this function will not match the performance of
 * the function SUM, but should show general capabilities.
 */
 public class MyDoSum extends com.f1j.addin.Func {

    final static short errInvalidValue = 3;

    /**
     * This static initializer guarantees that the instance of this
     * function is created and added to the list of add-in functions.
     */
    static {
        new MyDoSum();
    }
    /**
     * Make this private so that only one instance is created
     */
    private MyDoSum() {
        /* we can allow 1-30 arguments */
        super( "MyDoSum", 1, 30 );
        /* any argument can be a reference (not just resolved to
```

```
     * a"value"
     */
    for ( int ii = 1; ii < 30; ii++ )
        setReferenceArgument(ii);
}

/**
 * Override the com.f1j.addin.evaluate(...) to do the work
 */
public void evaluate( com.f1j.addin.FuncContext fc ) {
    int argCount = fc.getArgumentCount();
    double sum = 0.0;  //initialize the result
    for (int ii=0; ii < argCount; ii++) {
        com.f1j.addin.Value val = fc.getArgument(ii);
        if ( val.isCell()) {
            com.f1j.ss.Sheet sheet = val.getSheet();
            int cellType = Math.abs(sheet.getType(
                                    val.getRow1(),
                                    val.getCol1() ));
            if ( cellType == com.f1j.ss.Book.eTypeNumber ) {
                sum += sheet.getNumber( val.getRow1(),
                                        val.getCol1() );
            }
            else if ( cellType == com.f1j.ss.Book.eTypeError ) {
                fc.setReturnValue( errInvalidValue );
                return;
            }
        }
        else if ( val.isRange() ) {
            com.f1j.ss.Sheet sheet = val.getSheet();
            int row2 = val.getRow2();
            int col2 = val.getCol2();
            for ( int i = val.getRow1(); i <= row2; i++ ) {
                for ( int j = val.getCol1(); j <= col2; j++ ) {
                    int cellType = Math.abs( sheet.getType(i, j));
                   if ( cellType == com.f1j.ss.Book.eTypeNumber ) {
                        sum += sheet.getNumber( i,j );
                    }
                    else
                    if ( cellType == com.f1j.ss.Book.eTypeError) {
                        fc.setReturnValue( errInvalidValue );
                        return;
                    }
                }
            }
        }
        else if ( val.checkNumber() ) {
            sum += val.getNumber();
        }
        else {
```

```
                fc.setReturnValue( errInvalidValue );
                return;
            }
        }
        fc.setReturnValue( sum );
    }
}
```

Tidestone

CHAPTER 8

Performance Tuning and Specifications

This chapter provides several techniques for increasing Formula One for Java's performance. The last section lists Formula One for Java's technical specifications. The following topics are covered.

Performance Tuning Techniques

The following tips will help you make the most efficient use of memory and get the best performance from Formula One for Java.

- **Use only the files required to add the specific Formula One for Java functionality needed in your application.** See “Minimizing File Size” on page 21.
- **Use the com.f1j.swing.JBook.getLock() and com.f1j.swing.JBook.releaseLock() methods. GetLock** locks all views and workbooks for the current group so that no other thread will be allowed to access them until **releaseLock** is called. Lock a book before performing multiple related actions in order to improve performance and ensure consistency. See “Getting and Releasing Locks” below for more detailed information.
- **Build worksheets by rows instead of columns.** Formula One for Java allocates memory by rows. You can save memory by building worksheets a row at a time, rather than a column at a time. For example, fill cells in row 1 before moving to row 2, and so on, rather than filling cells in column A before moving to column B, and so on. See “Allocating Row and Column References” below for more detailed information.
- **Avoid formatting blank cells.** When you format a blank range that does not consist of whole rows or whole columns, Formula One for Java must create empty cells before applying formats.
- **Build ranges from the lower right-hand corner.** When building a table one cell at a time from code, starting in the lower right corner of the area in which you are working is faster and more efficient. This ensures that the row references are allocated simultaneously instead of one at a time. Likewise, each row is allocated once instead of reallocated each time a cell is added.
- **Use values instead of formulas whenever possible.**
- **Avoid adding empty rows and columns for white space.** Adjust the row height or column width to create white space instead of adding empty rows or columns. If you must add empty rows or columns, empty rows are more efficient.
- **Disable repainting when performing a series of operations.** When performing a number of sequential operations on a worksheet, disable repainting with **setRepaint** so the screen does not repaint after each operation. This increases the speed of the operation and avoids unnecessary screen flashing.
- **Use methods to copy and move data.** Use **editCopyRight**, **editCopyDown**, **copyRange**, and **moveRange** to copy and move cells. These methods are much faster than using the clipboard. In addition, these methods update cell references to maintain the integrity of your formulas.

Getting and Releasing Locks

The **com.f1j.swing.JBook.getLock()** method locks all views and workbooks for the current group so that no other thread will be allowed to access them until **com.f1j.swing.JBook.releaseLock()** is called.

Most methods and properties in Formula One for Java call **getLock** and **releaseLock** internally. Calls to **getLock** may be nested, but **releaseLock** must be called once for each **getLock**. Typically **getLock** should be used in the following fashion:

```
... jb.getLock(); try { ... } finally { jb.releaseLock(); } ...
```

A view should be locked before performing multiple related actions. This will ensure consistency and improve performance.

Example Code for Getting and Releasing Locks

The following code performs the same repeated action with and without the **getLock** method and outputs the elapsed time for both versions.

Note All of the examples in this Guide run on web servers that support the Java 2 Platform (available online at java.sun.com/products) and JavaServer Pages (also available on line at java.sun.com/products/jsp).

```
void test(int iters, int rows, int cols) {
    try {
        // Create a jb object
        com.f1j.swing.JBook jb = new com.f1j.swing.JBook();
        // Put the number 1.0 in the top left cell
        jb.setNumber(0, 0, 1.0);
        // Copy the number to the specified number of rows and columns
        jb.setSelection(0, 0, rows - 1, cols - 1);
        jb.editCopyRight();
        jb.editCopyDown();

        // Call the test method without doing a getLock().
        double secondsWithoutLock = test(jb, "Without jb.getLock():",
                                         iters, rows, cols);
        double secondsWithLock = 0.0;

        // Acquire a lock on the workbook.
        jb.getLock();
        try {
            // Call the test method with the workbook already locked.
            secondsWithLock = test(jb, "With jb.getLock():",
                                   iters, rows, cols);
            System.out.println("With lock is " +
                               (secondsWithoutLock / secondsWithLock) +
                               " times faster");
        }
```

```
            catch (Throwable e) {
                System.out.println("Got exception e=" + e);
            }
            finally {
                jb.releaseLock();
            }
        }
        catch (Throwable e) {
            System.out.println("Got exception e=" + e);
        }
    }

    double test(com.f1j.swing.JBook jb, String msg, int iters, int rows, int
            cols) throws com.f1j.F1Exception {
        long   start = System.currentTimeMillis();
        while (start == System.currentTimeMillis())
            ;
        start = System.currentTimeMillis();

        try {
            for (int iter = 0; iter < iters; iter++)
                for (int row = 0; row < rows; row++)
                    for (int col = 0; col < cols; col++)
                        jb.setNumber(row, col,
                                     jb.getNumber(row, col) + 1.0);
        }
        catch (com.f1j.util.F1Exception e) {
        }

        long   elapsed = System.currentTimeMillis() - start;
        double seconds = elapsed / 1000.0;

        System.out.println(msg + " cells=" + (iters * rows * cols) +
                " elapsed time=" + seconds + " seconds");

        return seconds;
    }

    void Run_ActionPerformed(java.awt.event.ActionEvent event)
    {
        /* Call test with 100 iterations and 100 rows by 100 columns
         * which will generate 1,000,000 calls to
         * com.f1j.swing.JBook.getNumber() and 1,000,000 calls to
         * com.f1j.swing.JBook.setNumber(int, int, double).
         */
        test(100, 100, 100);
    }
```

When this program is run under JDK 1.1.5 using a dual processor 200MHZ Pentium Pro, the output is:

```
Without jb.getLock(): cells=1000000 elapsed time=117.094 seconds
With jb.getLock(): cells = 1000000 elapsed time=4.469 seconds
With lock is 26.2 times faster
```

Here are the results from a 400MHZ Pentium II:

```
Without jb.getLock(): cells=1000000 elapsed time=3.125 seconds
With jb.getLock(): cells = 1000000 elapsed time=2.063 seconds
With lock is 1.5 times faster
```

Explanation of Example Code Results

The version that does the **getLock** is 26.2 times faster on the dual processor machine, 1.5 times faster with a single processor.

On the dual processor machine, using the version that does not use **getLock**, Formula One for Java starts doing background thread work as soon as the **jb.setNumber(...)** call is returned. This is because the machine uses the extra processor whenever possible. So the next call to **jb.getNumber** would have to "interrupt" the background thread, which is very time-consuming. When **getLock** is done outside of the loop, the other processor is not allowed to start working and therefore does not have to be interrupted.

On the single processor machine, the time savings is not as impressive but still valuable. **getLock** runs faster if you already have a lock, even if it doesn't have to interrupt a background thread. This accounts for most of the improvement.

Understanding Data Structure & Memory Size

Understanding basic data structure and how Formula One for Java allocates memory helps in understanding how to create more efficient workbook files.

There are three basic parts to a Formula One for Java data structure: the row references, the cell references, and the cells.

- Row References

Formula One for Java's row reference array is a contiguous block of memory containing one 4-byte reference for each row. For example, in a spreadsheet with10 rows, the row reference array contains 10 references of 4 bytes each. As you add rows, this array expands.

- Cell References

Each non-blank row consists of an array of cell references large enough to point to the last cell in a row. The cell reference array is a contiguous block of memory containing one 4-byte reference for each cell. So, if the last cell in a row is located in column H (the eighth column), that row contains eight 4-byte references (for columns A through H).

- Cells

The cells themselves are small data structures containing the cell contents. The structures include a cell's value, its format, its formula, its font, its alignment, and other cell attributes. Cells only exist in memory if they contain formulas or data or are formatted differently from the row or column in which they are located.

Allocating Row and Column References

Where possible, fill the sheet by rows instead of columns. Formula One for Java allocates an array of row references for each row with data. For each allocated row, it then allocates an array of column references. Rows without data have only a row reference, not an array of column references for that row.

Load the sheet from bottom right to top left. Since Formula One for Java allocates row and column references based upon the last row or column containing data, loading the sheet from bottom right to top left increases efficiency by allowing Formula One for Java to allocate the entire array of row and column references before filling the sheet. Once these arrays are allocated, Formula One for Java only has to allocate space (not references) for each cell.

Loading the worksheet the opposite way, from top left to bottom right, means that the arrays of row and column references must grow as the number of rows and columns grows. Each time these arrays need to grow, Formula One for Java must allocate a new array. This leads to increased garbage collection (memory re-allocation) and decreased speed.

Pre-allocate the arrays by putting a value in the lower right corner of the sheet, if you cannot load the sheet from the bottom right. This pre-allocates the row references. In order to pre-allocate all the arrays of column references, load a value into every cell in the column on the right (especially recommended in worksheets with a large number of columns). After loading data into the worksheet, delete the data in these pre-loaded cells. The worksheet then re-evaluates the amount of memory it needs and returns the remaining memory. This will decrease load times considerably.

Technical Specifications

The following table lists the technical specifications for Formula One for Java .

Parameter	Specification
Maximum worksheet size	1,073,741,824 rows by 32,768 columns
Maximum number of worksheets in a workbook	32,768
Column width	0 to 255 characters
Row height	0 to 409 points
Text length	16,383 characters
Formula length	1,024 characters
Number precision	15 digits
Largest positive number	9.99999999999999E307
Largest negative number	-9.99999999999999E307
Smallest positive number	1E-307
Smallest negative number	-1E-307
Maximum number of iterations	32,767
Maximum number of colors	56
Maximum number of available colors	Limited by display card and monitor
Maximum number of fonts per sheet	256
Maximum number of selected ranges	2,048
Maximum number of names per sheet	16,384
Maximum length of name	255
Maximum number of function arguments	30
Maximum length of format string	255
Maximum number of tables	Limited by system resources
Excel file format version	Excel 5.0 and 7.0, 97 and 2000
Number of worksheet functions	325
Maximum number of points in a polygon	8,191
Limits of date range	1/1/100 to 12/31/9999

Tidestone

Glossary

active	A cell(s) or other object selected and in a state to interact with a user.
add-in function	A small Java program that extends the capabilities of Formula One for Java.
API	Application Programming Interface. How a programmer accesses the behavior and state of classes and objects.
applet	A Java program distributed as an attachment in an HTML document and executed in a Java-enabled web browser. Includes a GUI and runs in the JVM on a client machine.
application	Sometimes “stand-alone application.” A program designed to run on its own--not within or as part of another program.
arrays	Data in lists or tables of values.
AWT	Abstract Windowing Toolkit. A set of Java development tools for creating GUI’s.
CAB files	CABinet files. Signed versions of JAR files.
class path	Directions to the address of a .class or JAR file.
component	A generic programming object: part of or a module from a larger program that serves a particular purpose or provides a particular functionality. In Java, JavaBeans, applets and servlets are components.
constructor	An instance method that creates an object with the same name as its class.
database	An organized set(s) of data, usually stored as records.
data consumer	In InfoBus, data consumers access data from the InfoBus to use in analysis or display.
data producer	In InfoBus, data producers retrieve data from a database or another source and make it available to other components.
DBMS	Database management system. Programs that manage (usually large, business) databases.
event	A notification of change that happens within a program.
field	Part of a record in which an item of data is stored. In a database, one column typically contains one field.
Formula One for Java	Formula One for Java Version 7.0

garbage collection	The automatic detection and freeing of memory no longer in use.
GUI	Graphical User Interface. The use of images in concert with the keyboard or mouse to provide the user access to the functions of a program.
HTML	HyperText Markup Language. A file format for hypertext documents on the Internet.
InfoBus	Software developed by that provides interfaces and protocols that enable applets to exchange data--an “Information Bus.”
instance/instantiation	A single iteration of a program object.
J2SDK	Java™ 2 SDK, Standard Edition, v 1.2. Software used to develop Java programs.
JAR files	Java ARchive files. A compressed file format that combines many files into one.
JavaBean	Also Java Bean. A portable, reusable, platform-independent component used within another application or applet.
JDBC	Java Database Connectivity. An API for database access that allows database-independent connectivity.
JDK 1.0.x or 1.1.x	Java Development Kit version 1.0.x or 1.1.x (JDK™ 1.x.x). Software used to develop Java programs.
JFC	Java Foundation Classes. Includes Swing components and other features used to create GUIs.
JRE	Java Runtime Environment. Includes the JVM, core class files and other files. Allows end users to run programs written in Java.
JSP	JavaServer Pages. A reusable-component-based, platform-independent web page and web-based application development environment.
JVM	Java Virtual Machine. The part of the Java Runtime Environment (JRE) that interprets bytecodes. Required to run programs written in Java.
ODBC	Open Database Connectivity
PDF files	Portable Document Format files. Files in native Adobe Acrobat format. Device and resolution independent.
platform	A functioning combination of computer hardware and software operating system.
plug-in	A small software component that adds functionality to other software.
Record	Analogous to a row in a worksheet, an ordered set of fields within a database.
RSA	An key-based encryption system. RSA stands for Rivest, Shamir and Adleman, who developed the system.
run time	The period when a program is loaded in memory and operating.
selected	Cell(s) or object(s) made active.

servlet	A component that is executed by the web server's JVM and provides server-side access to resources to the client. Servlets have no GUI.
SQL	"Structured Query Language." A language used in developing, maintaining, and querying relational databases and systems.
string	A sequence of characters or words, etc.
superclass	A class file containing the methods and other subtypes from which other class files are made.
Swing	Also Swing set. An API that contains components to develop a GUI in Java that do not rely on "native code." Using Swing, the GUI looks the same on all platforms, or can be set to emulate the GUI of its current environment.
threads, threading	Program instructions that run independently of the program that creates them.
URL	Uniform Resource Locator. Descriptor for mode of access (http, ftp, etc.) and location of an Internet resource (*e.g.*, a website).
view	Directions for a custom graphical instantiation of a software program.
workbook	A collection of worksheets within a file.
Workbook Designer	The GUI of Formula One for Java where user operations (entry of data, formulas, calculations, etc.) are carried out.
worksheet	Also known as a spreadsheet. A table of values ordered in rows and columns.

Web Page Link List

URL	Description
www.tidestone.com	Tidestone Technologies Web Site
www.javasoft.com or www.java.sun.com	Sun Microsystems Java Information Web Site
esuite.lotus.com	Lotus ESuite Web Site
www.rsa.com	RSA Data Security Web Site
BelSign	BelSign (Certificate Authority) Web Site
Verisign	VeriSign (Certificate Authority) Web Site
Thawte	Thawte Consulting (Certificate Authority) Web Site
developer.netscape.com	Netscape DevEdge Online
http://foldoc.doc.ic.ac.uk	Free On-Line Dictionary of Computing
http://www.sun.com/glossary	Glossary of Sun Terms
http://java.sun.com/docs/glossary.html	Glossary of Java Terms

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W