

MapleNet 10

Publisher's Guide

© Maplesoft, a division of Waterloo Maple Inc. 2005.

MapleNet and Maplet are trademarks of Waterloo Maple Inc.

© Maplesoft, a division of Waterloo Maple Inc. 2005. All rights reserved.

The electronic version (PDF) of this document may be downloaded and printed for personal use or stored as a copy on a personal machine. Information in this document is subject to change without notice and does not represent a commitment on the part of the vendor. The software described in this document is furnished under a license agreement and may be used or copied only in accordance with the agreement. It is against the law to copy the software on any medium except as specifically allowed in the agreement.

The use of general descriptive names, trade names, trademarks, etc., in this publication, even if the former are not especially identified, is not to be taken as a sign that such names, as understood by the Trade Marks and Merchandise Marks Act, may accordingly be used freely by anyone.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. Maplesoft is independent of Sun Microsystems, Inc.

Internet Explorer is a registered trademark of Microsoft Corporation.

Netscape is a registered trademark of Netscape Communications Corporation.

All other trademarks are property of their respective owners.

Contents

| | |
|--|-----------|
| Introduction | 1 |
| 1.1 Prerequisites | 2 |
| Required | 2 |
| Recommended | 2 |
| 1.2 Installing MapleNet Publisher | 2 |
| 1.3 General Notes | 3 |
| Developing Java-based Applets for MapleNet | 5 |
| In this chapter | 5 |
| 2.1 Developing Java-based Applets | 5 |
| 2.2 Writing Java-based Applets | 6 |
| Passing Requests to MapleNet | 6 |
| Sequence of Events | 8 |
| Maple Notes | 8 |
| 2.3 Compiling Java Code | 9 |
| 2.4 Creating a JAR File that Contains a Manifest | 9 |
| Creating a Manifest | 10 |
| Creating a JAR File | 10 |
| 2.5 Creating an HTML Page | 11 |
| Specifying Parameters | 13 |
| Loading the Web Page | 13 |
| 2.6 Publishing Files to the MapleNet Server | 14 |
| Using MapleNet Publishing Scripts | 14 |
| Example Procedure: How to Create and Publish a JAR File | 14 |
| Using the MapleNet Publisher | 16 |
| 2.7 Accessing Published Content | 18 |
| Developing Maplet Applications for MapleNet | 19 |
| In this chapter | 19 |
| 3.1 Writing a Maplet Application Compliant with MapleNet | 19 |
| Restrictions | 19 |
| 3.2 Creating an HTML Page | 20 |
| Specifying Parameters | 22 |
| Loading the Web Page | 22 |

| | |
|---|-----------|
| 3.3 Publishing Files to the MapleNet Server | 22 |
| Using MapleNet Publishing Scripts | 22 |
| Example Procedure: How to Create and Publish a JAR File | 23 |
| Using the MapleNet Publisher | 25 |
| 3.4 Accessing Published Content | 27 |
| Developing JavaServer Pages for MapleNet | 29 |
| In this chapter | 29 |
| 4.1 Prerequisites | 29 |
| 4.2 Creating MapleNet JSP Content | 29 |
| 4.3 Page Setup | 30 |
| 4.4 Using Maple Tags | 30 |
| Initialization Tag | 31 |
| Statement Tag | 31 |
| Code Tag | 34 |
| Assign Tag | 35 |
| Plot Tag | 35 |
| Release Tag | 35 |
| 4.5 Examples | 36 |
| Integration Example | 36 |
| Plot Example | 38 |
| 4.6 Publishing JSP Content to the MapleNet Server | 39 |
| Using MapleNet Publishing Scripts | 39 |
| Example Procedure: How to Create and Publish a JAR File | 40 |
| Using the MapleNet Publisher | 42 |
| 4.7 Accessing Published JSP Content | 43 |
| Developing Worksheets for MapleNet | 45 |
| In this chapter | 45 |
| 5.1 Writing Worksheets for Use on MapleNet | 45 |
| Tips to Improve Worksheet Layout | 46 |
| Tips to Improve Worksheet Performance | 46 |
| 5.2 Client Considerations | 47 |
| Browser support | 47 |
| Memory Limitations | 47 |
| Delay on First Use | 47 |
| 5.3 Server Considerations | 48 |
| 5.4 Customization of the MapleNet Worksheet Interface | 48 |
| 5.5 Publishing Worksheets to the MapleNet Server | 48 |
| 5.6 Accessing Published Worksheets | 48 |
| Appendix A | 51 |

1 Introduction

The *MapleNet™ Publisher's Guide* provides instructions and examples for the following.

- Installing MapleNet Publisher software to your computer.
See *Installing MapleNet Publisher* on page 2.
- Developing Java™ -based applets that communicate with the MapleNet server, including:
 - Writing a Java-based Applet
 - Compiling Java Code
 - Creating a JAR File that Contains a Manifest
 - Creating an HTML Page
 - Publishing Files to the MapleNet server
 - Accessing Published Content

See *Developing Java-based Applets for MapleNet* on page 5.

- Developing Maplet™ Applications for MapleNet, including:
 - Writing a MapleNet-Compliant Maplet
 - Creating an HTML Page
 - Publishing to the MapleNet server
 - Accessing Published Content

See *Developing Maplet Applications for MapleNet* on page 19.

- Developing JavaServer Pages™ for MapleNet, including:
 - Prerequisites
 - Creating MapleNet JSP Content Overview
 - Page Setup
 - Using Maple Tags
 - Examples
 - Publishing JSP Content to the MapleNet server
 - Accessing Published JSP Content

See *Developing JavaServer Pages for MapleNet* on page 29.

- Developing Worksheets for MapleNet, including:
 - Writing Worksheets
 - Client Considerations

- Server Considerations
- Customization of the MapleNet Worksheet Interface
- Publishing Worksheets to the MapleNet Server
- Accessing Published Worksheets

See *Developing Worksheets for MapleNet* on page 45.

1.1 Prerequisites

Required

- Java 2 Software Development Kit (J2SDK), 1.4.2
- Java 2 Runtime Environment (J2RE), 1.4.2
- Maple 10

Note: If you do not have SDK and J2RE installed, visit Sun Microsystems, Inc. at <http://java.sun.com/j2se/Web> page to download and install the latest version of Java 1.4.2. Java versions 1.4.2_06 and 1.4.2_07 have known issues and are not supported.

Recommended

- Java development environment for developing Java applets
- Web page development environment for creating HTML pages

1.2 Installing MapleNet Publisher

The publisher component is located in the Publisher directory on your CD. Copy this directory to your network or to another CD for distribution.

This component is intended to be installed on computers where MapleNet content is to be authored. It is already installed on the MapleNet server.

To install the publisher component of the MapleNet software:

- Double-click the `WindowsPubInstall.exe` file (Windows) or run the `<UNIXplatform>PubInstall.bin` file (UNIX) and follow the on-screen instructions.

When finished, the directory structure will be as follows.

| Directory | Description |
|-----------|---|
| bin | Location of batch files required for bundling and publishing content |
| doc | Location of Javadoc files describing the MapleNet classes, sample Java and HTML files, and the <code>publisherguide.pdf</code> file |
| jar | Location of Java code files |

Note: To access the script files from *any* directory, the `bin` directory must be added to the `path` environment variable.

1.3 General Notes

- The PearlServer prefixes commands to Maple with `printlevel:=0` to suppress extraneous output. When creating procedures, keep in mind that the default `printlevel` is set to 0.
- **To all MapleNet users, version 2.5 and earlier:**

If you package your applets in your own jar files, then you must update your manifest extension version information as shown in *Chapter 2.4: Creating a JAR File that Contains a Manifest* on page 9, so that your applets will access the newer version of `maplenetclient.jar`.

Also note the change in the archive attribute of HTML files in *Chapter 2.5: Creating an HTML Page* on page 11. You will need to update your published HTML pages.

- **To MapleNet users, version 1.3 and earlier:**

Due to a restructuring of Java classes throughout all Maplesoft products, the MapleNet APIs for the MapleNet client code have been moved to a different package. The package `com.maplesoft.client.*` is now renamed to `com.maplesoft.maplenet.client.*`

Therefore, any imports in your current applets must be changed to point at **`com.maplesoft.maplenet.client.*`** and recompiled.

2 Developing Java-based Applets for MapleNet

In this chapter

- Developing Java-based applets that communicate with the MapleNet Server
- Compiling applet code
- Creating a JAR file that contains a manifest
- Creating an HTML page that references applets
- Publishing to the MapleNet server
- Accessing published applets

2.1 Developing Java-based Applets

To create content for publication on the MapleNet server, complete the following tasks.

1. Write an applet that communicates with the MapleNet server.
See *Writing Java-based Applets* on page 6.
2. Compile the Java code.
See *Compiling Java Code* on page 9.
3. Create a JAR file that contains a manifest.
See *Creating a JAR File that Contains a Manifest* on page 9
4. Create an HTML page that references the applet.
See *Creating an HTML Page* on page 11.

Important Note to Previous MapleNet Users: Prior to MapleNet, Release 2.0, your applets used the Java package `com.maplesoft.client.*`, for example, class `com.maplesoft.client.MapleStatement`. Due to the repackaging of Java classes within all Maplesoft products, the classes in `com.maplesoft.client.*` are now found in the package `com.maplesoft.maplenet.client.*` The imports in your existing applets must be changed and recompiled to work in MapleNet, Release 2.5 and higher.

2.2 Writing Java-based Applets

The following *Sample Integration Applet* (Figure 1) requires a user to enter a **UserID**, **Password**, and expression. By clicking **Calculate**, a request is sent to the MapleNet server to integrate the equation. The result is displayed in the **Result** textbox. This applet illustrates how requests are passed to the MapleNet server. The input and output are text-based Standard Maple syntax.

Figure 1 Sample Integration Applet

Passing Requests to MapleNet The *Sample Integration Applet Code* (Listing 1) provides the section of code that performs the connection with MapleNet. In this example, the computer name is derived from the Web page that contains the applet during the initialization phase of the applet, which is stored in the *host* string variable. The complete code for this example is in the source file *integration.java*, which is available in the *doc/PublisherGuide* folder created during installation.

Listing 1 Sample Integration Applet Code

```
//Routine to take the input expression and
//send it to the MapleNet Server
//using the MapleStatement class
void do_calc (ActionEvent e) {
1 //Erase current answer
   TF_Result.setText( " ");
```

```

2 //Retrieve the expression
String expression = TF_Expression.getText();
//If there is an expression, process it
if ( expression.length () > 0 ) {
3     //Need user and password
String user = TF_Username.getText () ;
String pswd = TF_Password.getText () ;
4     //Must encapsulate the 'execute()' method in a try/catch
//block
try {

5         //Create the host connection. Will use the default MapleNet port
HostInfo hostinfo = new HostInfo(host);
//Create the user login object
UserInfo userinfo = new UserInfo( user, pswd );
6         //Create the MapleStatement that will connect to the server
MapleStatement stmt = new MapleStatement(hostinfo, userinfo );
7         //Encapsulate the expression into maple integration form
String question = "int( "+ expression + ", x)";
8         //Get MapleNet server to process the question
String answer = stmt.execute( question );
//Verify that an answer was retrieved
if (answer==null || answer.length() <= 0 ) {
            //No answer - check the XML returned from the server
            answer = "Error: " +stmt.getXMLString();
        }
9         //Display the result
TF_Result.setText( answer );
    } catch (Exception ex ) {
        //Catch all exceptions here
        //MapleStatement.execute will throw
        // MapleStatementException - usually syntax error in question
        // MapleException - server connection or login errors
        System.out.println("do_calc exception = " + ex.toString() );
    }

    //Put keyboard focus back to expression box
    TF_Expression.setSelectionStart (0);
    TF_Expression.setSelectionEnd( equation.length() );
    TF_Expression.requestFocus();
} else {
    // No expression specified
    Toolkit.getDefaultToolkit() .beep();
}
}
}

```

Sequence of Events When the user clicks the **Calculate** button on the *Sample Integration Applet* (Figure 1), the `do_calc()` routine is invoked with the following sequence of events.

1. The **Result** textbox is set to blank.
2. The expression is extracted from the **Enter** text box.
3. The username and password are obtained from the appropriate text boxes.
4. A **try/catch** block is used to catch any exceptions that are raised when communicating with the MapleNet server.
5. The **HostInfo** and **UserInfo** objects are created. These objects contain connection information.
6. A **MapleStatement** object is created using the **HostInfo** and **UserInfo** objects. Note that this step creates only the object. The actual connection to the server is not made until the **MapleStatement** `execute` method is used.
7. The actual request for the server is built using the Maple syntax for integration of the expression from event 2.
8. The question is passed to the server using the **MapleStatement** `execute` method. The result is stored in the **String variable answer**. Note that any error from the server is raised as an exception. If an exception occurs, it is handled in the `catch()` portion of code.
9. The answer is displayed in the **Result** text box.

Maple Notes

- To pass *multiple questions* to the server, you can build a *string array* containing the questions and pass them to the `execute` method. This method returns a string array of resulting answers corresponding to each question.
- Expression sequences cannot be rendered in MathML. Therefore, the output for any Maple command you are using that returns an expression sequence (for example, `solve`) must be converted to a list if you want to render it in MathML. If you do not convert it, only the first element of the expression sequence will be displayed.

2.3 Compiling Java Code

Compiling the applet code with a Java compiler produces an `integration.class` file. This class file contains the executable code for the applet. To use the executable code, create an HTML page that references the class as an applet.

The MapleNet JAR files must be included in the classpath argument of the compiler. For a complete list of necessary JAR files, see *Appendix A* on page 51. Listing 2 illustrates how to compile a Java class named `integration.java`.

Listing 2 Compiling `integration.java`

On Windows:

```
javac -classpath maplenetclient-client.jar;maplenetclient-freehep.jar;maplenetclient-index.jar;maplenetclient-maplenet.jar;maplenetclient-maplets.jar;maplenetclient-mathdoc.jar;maplenetclient-mathdoc2.jar;maplenetclient-misc.jar;maplenetclient-openviz2.jar;maplenetclient-plot.jar;maplenetclient-url.jar;maplenetclient-util.jar;maplenetclient-utilcodepage.jar;maplenetclient-webeq3.jar integration.java
```

On other platforms (Linux, UNIX and Macintosh):

```
javac -classpath maplenetclient-client.jar:maplenetclient-freehep.jar:maplenetclient-index.jar:maplenetclient-maplenet.jar:maplenetclient-maplets.jar:maplenetclient-mathdoc.jar:maplenetclient-mathdoc2.jar:maplenetclient-misc.jar:maplenetclient-openviz2.jar:maplenetclient-plot.jar:maplenetclient-url.jar:maplenetclient-util.jar:maplenetclient-utilcodepage.jar:maplenetclient-webeq3.jar integration.java
```

Notes:

1. The previous commands assume that `integration.java` file and the jar files are in the same directory. If not, classpath must be changed appropriately.
2. Path separator character is a semi-colon (;) in Windows and a colon (:) on other platforms.

Important note to previous MapleNet users: In earlier versions, all MapleNet client code was packaged in a single JAR file (`maplenetclient.jar`). In MapleNet 10, the code is separated into several JAR files, to improve download efficiency. To be compatible with MapleNet 10, your existing code must be recompiled with these new JAR files.

2.4 Creating a JAR File that Contains a Manifest

The Java Extension Mechanism is used to distribute the `maplenetclient.jar` file that is stored on the Web server. If you reference the `maplenetclient.jar` file in the manifest of the JAR file that contains the compiled code, `maplenetclient.jar` will be installed automatically when required.

To implement this, you need to first create a manifest, then create a JAR file that includes both this manifest and the compiled class files for your applet, and then reference this JAR file in the `.html` file.

Creating a Manifest

To create the manifest, you need to know the URL path to the `maplenetclient.jar` in your site's Web root directory, for example, <http://mysite.com/maplenet/jar/maplenetclient.jar>. If you do not know this information, ask your system administrator.

Create a text file, for example, `mymanifest.txt`, that contains the following lines.

```
Extension-List: maplenetclient
maplenetclient-Extension-Name: maplenetclient
maplenetclient-Specification-Vendor: Waterloo Maple Inc.
maplenetclient-Specification-Version: 10
maplenetclient-Implementation-Vendor: Maplesoft
maplenetclient-Implementation-Vendor-Id: com.maplesoft.maplenet
maplenetclient-Implementation-Version: 10
maplenetclient-Implementation-URL: http://mysite.com/maplenet/jar/maplenetclient.jar
```

where the URL specified in the last line is the path to the `maplenetclient.jar` in your site's Web root directory.

Note: If your Web server, for example, Tomcat or Apache, is running on a port other than 80, such as 8080, you must change the URL to include the port number. For example, <http://mysite.com:8080/maplenet/jar/maplenetclient.jar>

Important note to previous MapleNet users: Note the change in version number, Implementation-Vendor and Implementation-Vendor-ID in the manifest on page 10.

When this text file is created and the class files for the applet are compiled, you can create the JAR file that will be referenced in the `.html` file.

Creating a JAR File

Use the following command to create a JAR file, for example, `myapplet.jar`, that contains the manifest and the required class files. This example assumes the manifest file and the class files are both in the current directory.

```
jar -cmf mymanifest.txt myapplet.jar *.class
```

where `mymanifest.txt` is the manifest file created in *Creating a Manifest* on page 10, `myapplet.jar` is the name of the JAR file being created, and `*.class` is used to include all class files in the present directory

For some classes in the API to be executed (**PlotComponent**, **mathmlEquation** and **mathmlEditEquation**), the application jar file must be signed for appropriate permissions on the client machine. For complete instructions on how to sign a jar file, refer to

<http://java.sun.com/docs/books/tutorial/security1.2/toolsign/index.html>.

Once this JAR file is created (and signed, if necessary), specify it in the `.html` file, as shown in the following section.

2.5 Creating an HTML Page

Before you can publish to the MapleNet server, you must create an HTML page that references the compiled Java code (the `.class` file) as an applet.

The following *Integration.html* sample code (Listing 3 on page 11) illustrates how to build a page for Microsoft® Internet Explorer® and Netscape®.

Note: Java applets that require a Java version greater than 1.1 must use the Java Plugin. Use the `<object>` tag in Internet Explorer and the `<embed>` tag in Netscape. For a complete discussion of setting up Web pages for simultaneous Internet Explorer and Netscape support, refer to

<http://java.sun.com/products/plugin/1.3/docs/intranet.html>

Listing 3 Integration.html

```
<html>
<head>
<title>
Integration Example
</title>
</head>
<body>
<center>
<H2>Integration Applet will appear below in a Java-enabled browser</H2>
<object classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93" width="400" height="300"
align="middle" >
  <param name="type"      value="application/x-java-applet;version=1.4" />
  <param name="CODEBASE"  value = "./" />
  <param name="CODE"      value = "integration.class" />
  <param name="NAME"      value = "IntegrationApplet" />
</object>
</center>
</body>
</html>
```

```
<param name="ARCHIVE" value = "myapplet.jar,/maplenet/jar/maplenetclient-
index.jar"1 />

<param name="expression" value="3*x^2 + 2*x +1" />

<comment>
  <embed type="application/x-java-applet;version=1.4"
    width="400"
    height="300"
    align="middle"
    code="integration.class"
    codebase="."

    archive= "myapplet.jar,/maplenet/jar/maplenetclient-index.jar"1
    expression="3*x^2 + 2*x +1">
  </embed> </comment> Error Java 1.4 not supported </noembed>
</embed>
</object>
</center>
</body>
</html>
```

If you choose to use an applet tag in your Web site, the parameters must be specified as shown in Listing 4.

Listing 4 Applet Parameter Syntax

```
<applet
width="400"
height="300"
align="middle"
code=" integration.class "
codebase="."
archive=" myapplet.jar, /maplenet/jar/maplenetclient-index.jar" >
<param name=" expression " value="3*x^2 + 2*x +1"/>
</applet>
```

1. Relative path to **maplenetclient-index.jar** must be specified. Ensure that all **maplenetclient-*.jar** files listed in Appendix A are in the same directory as **maplenetclient-index.jar**.

Specifying Parameters

In the `.html` file, specify the parameters listed in Table 1.

Table 1 HTML File Parameters

| Parameter | Description |
|-----------------------|--|
| <code>height</code> | Specifies the size of the applet on the Web page. |
| <code>width</code> | Specifies the size of the applet on the Web page. |
| <code>codebase</code> | Specifies the directory from which to load the class file. |
| <code>code</code> | Specifies the class of the applet. In the example, <code>integration.class</code> is specified. |
| <code>archive</code> | Specifies <code>maplenetclient-index.jar</code> and additional JAR files that are required, for example, <code>myapplet.jar</code> . |

For this example, `integration.java`, you can add *optional* input parameters in the Java source code, as listed in Table 2.

Table 2 Java Source Code File Parameters

| Parameter | Description |
|-------------------------|---|
| <code>host</code> | Specifies the name of the MapleNet server. The default is the name of the server for the Web page. |
| <code>expression</code> | Specifies the initial value for the Enter Expression text box. The applet does not perform the integration until the Calculate button is clicked. If not specified, the field is blank. |

Loading the Web Page

When the Web page loads, it runs the applet specified by the `code` parameter. This code accesses the MapleNet server using the classes indexed in the `maplenetclient-index.jar` file. This file and all other files listed in *Appendix A* on page 51 must be accessible to the Web browser.

Once the applet is compiled and the Web page is created, publish the associated files (`.html`, `.java`, and `.jar`) to the MapleNet server.

2.6 Publishing Files to the MapleNet Server

The Web page, source code, and compiled code files can be published to the Web server directly or using one of the following methods.

- *Using MapleNet Publishing Scripts* on page 14 (command line)
- *Using the MapleNet Publisher* on page 16 (graphical interface)

Using MapleNet Publishing Scripts

MapleNet publishing scripts bundle and send files to the server to be installed in a location accessible by the client.

To publish files to the server:

1. Place the files required for publishing in the same subdirectory.
2. Run the `CreateJar` (`createjar` in UNIX) script to wrap the contents of the subdirectory into a JAR file. A JAR file is a Java version of a zip file. As such, it can be viewed with a zip program such as WinZip® 7.0.
3. Run the `PublishJar` (`publishjar` in UNIX) script to publish the contents of the JAR file on the server.

For an example procedure, see *Example Procedure: How to Create and Publish a JAR File* on page 14.

Example Procedure: How to Create and Publish a JAR File

The following example files are used to illustrate the procedure for moving files to the server.

| Directory or File | Description |
|--------------------------------------|--|
| \Projects\Tutorial | Main directory |
| \Projects\Tutorial\integration.html | Web page |
| \Projects\Tutorial\integration.java | Source code for the applet to be contained in the page |
| \Projects\Tutorial\integration.class | Compiled Java code |

1. Create a JAR file using the example files.

Windows

Run the `CreateJar.bat` file from the `Projects` directory.

```
cd Projects
CreateJar tutor1.jar Tutorial
```

where `tutor1.jar` is the JAR file to be created and `Tutorial` is the subdirectory to be searched for source files.

The `CreateJar` command searches the directory (`Tutorial`) and all its subdirectories for source files and then creates the `tutor1.jar` file in the `Projects` directory.

UNIX

Run the `createjar` script from the `Projects` directory.

```
cd Projects
createjar tutor1.jar Tutorial
```

where `tutor1.jar` is the JAR file to be created and `Tutorial` is the subdirectory to be searched for source files.

The `createjar` command searches the directory (`Tutorial`) and all its subdirectories for source files and then creates the `tutor1.jar` file in the `Projects` directory.

2. Verify the contents of the `tutor1.jar` file using the `jar` tool from the J2SDK. Enter the command:

```
jar -tf tutor1.jar
```

All directories/files contained in the jar file are displayed. The contents will resemble the following.

```
/Projects>jar -tf tutor1.jar
META-INF/
META-INF/MANIFEST.MF
Tutorial/
Tutorial/integration$1.class
Tutorial/integration.class
Tutorial/integration.html
Tutorial/integration.java
/Projects>
```

If the contents are not similar to the above listing, there was a problem with the `createjar` step.

3. Publish the JAR file on the server. The command parameters are described in Table 3.

Windows

Run the `PublishJar.bat` file.

```
PublishJar user password jarfile destination
```

UNIX

Run the `publishjar` command.

```
publishjar user password jarfile destination
```

For example,

```
publishjar admin spiff tutor1.jar //Servername/Projects/tutor1.jar
```

The server recreates all the files and directories contained in the JAR file. For example, the files are extracted to `\Projects\Tutorial` on the server.

Table 3 Command Parameters

| Parameter | Description |
|-------------|--|
| user | User name already defined on the server to which you are publishing. |
| password | Password for the user. |
| jarfile | Name of the JAR file. |
| destination | Name of the remote server and directory where the JAR file is sent and unwrapped. The directory will be placed in the Web server's root directory. Note: The publisher must have write permission to this directory. |

Note: You can publish to the server only when the MapleNet server and the Web server are running. For more information, refer to the *MapleNet Installation Guide*.

Using the MapleNet Publisher

The **MapleNet Publisher** allows you to select the source content, either an archived file or folder, and to specify a target folder.

You must know the following information.

- The URL of the MapleNet server and the port on which the server listens.
- The MapleNet user account and password that has authority to publish content to the Web server.
- The location of the files to be archived or the archive, if previously created using another method.
- The target folder from which the archive is to be extracted.

To launch the MapleNet Publisher:

In the `bin` directory, double-click the `maplenetpublisher.bat` file (Windows) or run the `maplenetpublisher` file (UNIX). The **MapleNet Publisher** dialog opens.

- If you have an archive, follow the instructions in *To publish an archive*.
- If you must create an archive, follow the instructions in *To create an archive* and then *To publish an archive*.

To create an archive:

1. In the **MapleNet Publisher** dialog, click **Create Archive**. The **Jar Create Utility/Package Directory** dialog opens.
2. Enter a directory and click **Next**.
(Optional) Click **Browse** to find and select a directory.
3. In the **Jar Create Utility/Create Archive** dialog, enter the archive filename, and click **Finish**. A message is displayed indicating that the archive filename extension has been changed to ".jar".

To publish an archive:

1. In the **MapleNet Publisher** dialog, click **Publish Archive**. The **MapleNet Publisher/Choose Archive** dialog opens.
2. Enter (or **Browse** for) an archive filename. The contents of the selected archive (.jar file) are listed in the **Archive Contents** section of the dialog. To review the code of a particular HTML file in the **Archive Contents**, select an .html file and click **Explore**. The **Zip File Browser** dialog opens, displaying the HTML code. To return to the previous dialog, close the **Zip File Browser**.
3. Once you have selected an archive, click **Next**. The **MapleNet Publisher/MapleNet Server** dialog opens.
4. Enter the **Name** of the MapleNet server, for example, `mysite.com`. The **Port** number field is automatically populated with the default 14444.
5. Enter the **User Name** and **Password**. The default **Admin** password is **spiff**.
6. Click **Next**. The **MapleNet Publisher/Deploy** dialog opens. Enter the path and destination jar filename, for example, enter `/Projects/tutor1.jar`. The Publisher places the `tutor1.jar` on the server in the `/Projects` directory and then extracts all the files from `tutor1.jar` into this directory.
7. Click **Finish**. The **MapleNet Publisher** closes.

2.7 Accessing Published Content

Demonstrating with the example files of the previous section, a user can access the published Web pages using the following URL.

<http://Servername/maplenet/Projects/Tutorial/integration.html>

The path is based on the `destination` parameter specified in the `PublishJar` (`publishjar` in UNIX) command. (See Table 3.)

3 Developing Maplet Applications for MapleNet

In this chapter

- Writing a Maplet Application Compliant with MapleNet
- Creating an HTML Page (that references a Maplet application)
- Publishing files to the MapleNet Server
- Accessing Published Content

3.1 Writing a Maplet Application Compliant with MapleNet

When writing a Maplet application for MapleNet, follow these guidelines.

- Ensure that the last line in the file is the `Maplets:-Display` command.
- Save the file as a `.maplet` file.
- It is recommended that you use the `waitforresult='false'` setting in your `Evaluate` statements.
- Expression sequences cannot be rendered in MathML. Therefore, the output for any Maple command you are using that returns an expression sequence (for example, `solve`) must be converted to a list if you want to render it in MathML. If you do not convert it, only the first element of the expression sequence will be displayed.

Restrictions

When using Maplet applications in MapleNet, consider the following restrictions.

- You cannot open files.
- You cannot open a Maplet application from another Maplet application.

3.2 Creating an HTML Page

Before you can publish to the MapleNet server, you must create an HTML page that references the saved `.maplet` file.

The following sample code (Listing 5) illustrates how to build a page for Internet Explorer and Netscape.

Note: Java applets that require a Java version greater than 1.1 must use the Java Plug-in. Use the `<object>` tag in Internet Explorer and the `<embed>` tag in Netscape. For a complete discussion of setting up Web pages for simultaneous Internet Explorer and Netscape support, refer to <http://java.sun.com/products/plugin/1.3/docs/intranet.html>

Listing 5 Maplet Application HTML Code

```
<head>
<title>
Maplet Application Test Page
</title>
</head>
<body>
This Web page contains an applet that will call the Maplet application specified below.
The Maplet application will appear in a separate window. <br>
<center>
  <object classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93" width="400"
height="300" align="middle" >
    <param name="CODEBASE" value = "." />
    <param name="CODE" value = "com.maplesoft.maplenet1.maplets.MapletLoader.class"
/>
    <param name="ARCHIVE" value = "/maplenet/jar/mapletloader.jar,
/maplenet/jar/maplenetclient-index.jar2" />
    <param name="NAME" value = "plot_02_x" />
    <param name="mapleFile" value="mytest.maplet" />
    <param name="user" value="client" />
    <param name="password" value="demopass" />
  <comment>
```

-
1. Prior to MapleNet, Release 2.0, Maplet applications were loaded via the class *com.maplesoft.client.maplets.MapletLoader*. This class is now deprecated and must be replaced by *com.maplesoft.maplenet.maplets.MapletLoader*
 2. Ensure that all `maplenetclient-*.jar` files (see the complete list of JAR files in Appendix A) are in the same directory as `maplenetclient-index.jar` file.

```

    <embed type="application/x-java-applet;version=1.4"
          width="400"
          height="300"
          align="middle"
          code="com.maplesoft.maplenet.maplets.MapletLoader.class"
          codebase="."
index.jar1"
          archive="/maplenet/jar/mapletloader.jar,/maplenet/jar/maplenetclient-
          mapleFile="mytest.maplet"
          user="client"
          password="demopass">
    <noembed> </comment> Error Java 1.4 not supported </noembed>
  </embed>
</object>
</center>
</body>
</html>

```

If you choose to use an applet tag in your Web site, the parameters must be specified as shown in Listing 6.

Listing 6 Applet Parameter Syntax

```

<applet
width="400"

height="300"
align="middle"
code=" com.maplesoft.maplenet.maplets.MapleLoader.class"
codebase="."
archive=" /maplenet/jar/mapletloader.jar, /maplenet/jar/maplenetclient-index.jar1" >
<param name=" mapleFile" value="mytest.maplet"/>
<param name="user" value="client"/>
<param name="password" value="demopass"/>
</applet>

```

1. Ensure that all maplenetclient-*.jar files (see the complete list of JAR files in Appendix A) are in the same directory as maplenetclient-index.jar file.

Specifying Parameters

In the `.html` file, specify the parameters listed in Table 4.

Table 4 HTML File Parameters

| Parameter | Description |
|------------------------|--|
| <code>height</code> | Specifies the size of the Maplet application on the Web page. |
| <code>width</code> | Specifies the size of the Maplet application on the Web page. |
| <code>codebase</code> | Specifies the directory from which to load the class file. |
| <code>code</code> | Specifies the required class file. For Maplet applications, <code>MapletLoader</code> class is always called. |
| <code>archive</code> | Specifies JAR files that are required, for example, <code>mapletloader.jar</code> and <code>maplenetclient-index.jar</code> Note: Ensure the path to jar files is correct for your system. |
| <code>mapleFile</code> | Specifies the Maplet application file. |

Loading the Web Page

When the Web page loads, it runs the `MapletLoader` applet, which in turn starts the Maplet application specified by the `mapleFile` parameter. This code accesses the MapleNet server using the classes contained in the JAR files (listed in *Appendix A* on page 51), indexed by the `maplenetclient-index.jar` file. These files must be accessible to the Web browser.

Once the Maplet application and the Web page are created, publish them to the MapleNet server.

3.3 Publishing Files to the MapleNet Server

The Web page and Maplet application file can be published to the Web server directly or using one of the following methods.

- *Using MapleNet Publishing Scripts* on page 22 (command line)
- *Using the MapleNet Publisher* on page 25 (graphical interface)

Using MapleNet Publishing Scripts

MapleNet publishing scripts bundle and send files to the server to be installed in a location accessible by the client.

To publish files to the server:

1. Place the files required for publishing in the same subdirectory.
2. Run the `CreateJar` (`createjar` in UNIX) script to wrap the contents of the subdirectory into a JAR file. A JAR file is a Java version of a zip file. As such, it can be viewed with a zip program such as WinZip 7.0.
3. Run the `PublishJar` (`publishjar` in UNIX) script to publish the contents of the JAR file on the server.

For an example procedure, see *Example Procedure: How to Create and Publish a JAR File*.

Example Procedure: How to Create and Publish a JAR File

The following example files are used to illustrate the procedure for moving files to the server.

| Directory or File | Description |
|---------------------------------------|--|
| \Projects\Tutorial | Main directory |
| \Projects\Tutorial\integration.html | Web page |
| \Projects\Tutorial\integration.maplet | Source code for the Maplet application to be contained in the page |

1. Create a JAR file using the example files.

Windows

Run the `CreateJar.bat` file from the `Projects` directory.

```
cd Projects
CreateJar tutor1.jar Tutorial
```

where `tutor1.jar` is the JAR file to be created and `Tutorial` is the subdirectory to be searched for source files.

The `CreateJar` command searches the directory (`Tutorial`) and all its subdirectories for source files and then creates the `tutor1.jar` file in the `Projects` directory.

UNIX

Run the `createjar` script from the `Projects` directory.

```
cd Projects
createjar tutor1.jar Tutorial
```

where `tutor1.jar` is the .jar file to be created and `Tutorial` is the subdirectory to be searched for source files.

The `createjar` command searches the directory (Tutorial) and all its subdirectories for source files and then creates the `tutor1.jar` file in the Projects directory.

2. Verify the contents of the `tutor1.jar` file using the `jar` tool from the J2SDK. Enter the command:

```
jar -tf tutor1.jar
```

All directories/files contained in the jar file are displayed. The contents will resemble the following.

```
/Projects>jar -tf tutor1.jar
META-INF/
META-INF/MANIFEST.MF
Tutorial/
Tutorial/integration.html
Tutorial/integration.maplet
/Projects>
```

If the contents are not similar to the above listing, there was a problem with the `createjar` step.

3. Publish the JAR file on the server. The command parameters are described in Table 5.

Windows

Run the PublishJar.bat file.

```
PublishJar user password jarfile destination
```

UNIX

Run the publishjar command.

```
publishjar user password jarfile destination
```

For example,

```
publishjar admin spiff tutor1.jar //Servername/Projects/tutor1.jar
```

The server recreates all the files and directories contained in the JAR file. For example, the files are extracted to \Projects\Tutorial on the server.

Table 5 Command Parameters

| Parameter | Description |
|-------------|---|
| user | User name already defined on the server to which you are publishing |
| password | Password for the user |
| jarfile | Name of the JAR file |
| destination | Name of the remote server and directory where the JAR file is sent and unwrapped. The directory will be placed in the Web server's root directory. Note: The server must have write permission to this directory. |

Note: You can publish to the server only when the MapleNet server and the Web server are running. For more information, refer to the *MapleNet Installation Guide*.

Using the MapleNet Publisher

The **MapleNet Publisher** allows you to select the source content, either an archived file or folder, and to specify a target folder.

You must know the following information.

- The URL of the MapleNet server and the port on which the server listens.

- The MapleNet user account and password that has authority to publish content to the Web server.
- The location of the files to be archived, or the archive if previously created using another method.
- The target folder from which the archive is to be extracted.

To launch the MapleNet Publisher:

In the `bin` directory, double-click the `maplenetpublisher.bat` file (Windows) or run the `maplenetpublisher` file (UNIX). The **MapleNet Publisher** dialog opens.

- If you have an archive, follow the instructions in *To publish an archive*.
- If you must create an archive, follow the instructions in *To create an archive* and then *To publish an archive*.

To create an archive:

1. In the **MapleNet Publisher** dialog, click **Create Archive**. The **Jar Create Utility/Package Directory** dialog opens.
2. Enter a directory and click **Next**.
(Optional) Click **Browse** to find and select a directory.
3. In the **Jar Create Utility/Create Archive** dialog, enter the archive filename, and click **Finish**. A message is displayed indicating that the archive filename extension has been changed to “.jar”.

To publish an archive:

1. In the **MapleNet Publisher** dialog, click **Publish Archive**. The **MapleNet Publisher/Choose Archive** dialog opens.
2. Enter (or **Browse** for) an archive filename. The contents of the selected archive (.jar file) are listed in the **Archive Contents** section of the dialog.
To review the code of a particular HTML file in the **Archive Contents**, select an .html file and click **Explore**. The **Zip File Browser** dialog opens, displaying the HTML code. To return to the previous dialog, close the **Zip File Browser**.
3. Once you have selected an archive, click **Next**. The **MapleNet Publisher/MapleNet Server** dialog opens.
4. Enter the **Name** of the MapleNet server, for example, `mysite.com`. The **Port** number field is automatically populated with the default 14444.

5. Enter the **User Name** and **Password**. The default user name is **admin** and password is **spiff**.
6. Click **Next**. The **MapleNet Publisher/Deploy** dialog opens. Enter the path and destination jar filename, for example, enter `/Projects/tutor1.jar`. The Publisher places the `tutor1.jar` on the server in the `/Projects` directory and then extracts all the files from `tutor1.jar` into this directory.
7. Click **Finish**. The **MapleNet Publisher** closes.

3.4 Accessing Published Content

Demonstrating with the example files of the previous section, a user can access the published Web pages using the following URL.

`http://Servername/maplenet/Projects/Tutorial/integration.html`

The path is based on the `destination` parameter specified in the `PublishJar` (`publishjar` in UNIX) command. (See Table 5.)

4 Developing JavaServer Pages for MapleNet

MapleNet JavaServer Pages (JSP[™]) allow you to use HTML pages to display Maple calculations. Maple expressions can be inserted in Web page sources and rendered within Web browsers.

In this chapter

- Prerequisites
- Creating MapleNet JSP Content
- Page Setup
- Using Maple Tags
- Examples
- Publishing JSP Content to the MapleNet Server
- Accessing Published JSP Content

4.1 Prerequisites

The server must support the use of JavaServer Pages. For example, a compatible server is Tomcat 4.1.

4.2 Creating MapleNet JSP Content

To display Maple content on the Web you must add special tags to Web page sources. The tags follow normal HTML tag conventions in that they are either of the single tag form

```
<tag ... tag parameters .../>
```

or matched opening and closing tags of the form

```
< tag ... tag parameters ...>
```

```
... data ...
```

```
</tag>
```

For example, in static HTML pages most of the page text is enclosed in `html` and `body` tags, as in the following.

```
<html>
```

```
<body>
  This is a simple page.
</body>
</html>
```

To begin creating MapleNet JSP content, follow the instructions in the *Page Setup* and *Using Maple Tags* sections of this chapter.

4.3 Page Setup

The server must recognize Maple specific tags. All tags in the page are processed in the order encountered on the source pages.

1. Add a `taglib` directive before the `<html>` tag at the top of the document. This allows the server to recognize Maple specific tags.

```
<%@ taglib prefix="maple" uri="/maplemath.tld" %>
```

where

- `<%@` identifies this as a JSP directive
- `taglib` specifies the name of the directive
- `prefix="maple"` states that `maple` is the tag name, for example `<maple:init ... >`
- `uri` specifies that the server use the uniform resource identifier (uri) `"/maplemath.tld"` to locate the Tag Library Descriptor

Technical Note: In the default install, the `WEB-INF/web.xml` file instructs the JSP engine to map the `/maplenet.tld` uri to the `WEB-INF/tlds/maplenet.tld` file)

2. Add a page tag following the `taglib` directive. This allows the server to maintain user data between pages requested by the user.

```
<%@ page session="true" %>
```

where

- `<%@` identifies this as a JSP directive
- `page` specifies the name of the directive
- `session="true"` indicates the server will store data for the entire session

4.4 Using Maple Tags

This section describes the `init`, `statement`, `code`, `assign`, `plot`, and `release` tags.

Initialization Tag

Add an `init` (initialization) tag to indicate how the Maple kernel is to be used. Use the following syntax.

```
<maple:init scope="[tag|page|request|session]"
debug="[false|true]" />
```

where

- `maple:init` specifies the name of the tag. This tag must occur before any other maple tag. If not, any statement or code tags encountered before the `init` tag are executed as if the `scope` was set to `page`.
- `scope` indicates how long the Maple kernel is kept active.

`tag` specifies that the kernel is used only for the expression(s) within the current `<maple:xxx>` tag. Each Maple tag has a new kernel, and no data is stored between Maple tags.

`page` specifies that the same kernel is used for all Maple tags encountered on this page. Once the page is sent to the user's browser, the kernel is released. Therefore multiple visits by a user to the same page result in a new kernel on each visit. This frees up the kernel for another user, but no data is stored for the same user on subsequent visits.

`request` specifies that the same kernel is used for all Maple tags in a single request to the server. A browser request may traverse one or more source pages.

`session` specifies that the same kernel is used for all Maple tags processed by the server for a particular user in a single browser session. Browser sessions are established by the `<%@ page ... >` directive.

If the tag `<%@ page session="true" %>` is encountered in a server, then the server uses the same session (not Maple session) for this user and remembers all the users server settings between visits as long as the browser is active and has not timed out. (Timeout levels are set by the server administrator).

If the page directive `session` is true, and the `<maple:init ...>` `scope` is set to `session`, then a handle to the Maple kernel is stored in the session data so that the Maple tags can share the same Maple kernel.

- `debug`, when true, specifies that all `<maple:xxx>` tags add debugging information to the output HTML.

Statement Tag

The `statement` tag executes one or more Maple expressions, with the result of the expression(s) being inserted directly into the resultant HTML page. The form of the tag is as follows.

```
<maple:statement lastline="[false|true]" breakline="[true|false]"
type="[text|plot]" width="value" height="value">
    ... Maple expression ...
    ... Maple expression ...
</maple:statement>
```

If the result is Maple output, it is inserted as text. If the result is a plot, the plot is converted to an image, and an HTML `` tag is inserted in the page.

Important: Each line in the statement block is executed separately. Therefore, all the code for control structures, such as `if/else` and `try/catch` must be on the same line. For example:

```
if (x>0) then s := +1; else s:=-1; end if
```

- A page can have more than one statement section. If the `init` tag sets the scope to `session`, all statement sections use the same kernel.
- If one or more of the expressions are expected to create a static plot, set the `type` parameter to `'plot'`. The kernel then generates a plot using the numeric `width` and `height` parameters. For interactive plots, see *Plot Tag* on page 35.
- The parameters `lastline` and `breakline` are optional. When `lastline` is `true`, all Maple expressions are evaluated within the tag but only the output generated by the last expression is inserted into the Web page content. If `breakline` is `true`, an HTML break tag, `
`, is inserted after the output of each expression, forcing the browser to display each expression on a new line.

When using both parameters, consider the following inserted into Web page content.

```
<maple:statement breakline=... lastline=... >
1+1; 2+2; 3+3;
7;
8;
9;
</maple:statement>
```

Table 6, "Output in Browser," on page 33 illustrates how the above input is displayed in a Web browser based on parameter usage.

Table 6: Output in Browser

| | | breakline | |
|----------|-------|----------------------|----------|
| | | true | false |
| lastline | true | 9 | 9 |
| | false | 2 4 6 7 8 9 | 2 4 6789 |

The following illustrates how content is processed and displayed using the examples given in *Table 6*.

- If `lastline` is `true` and `breakline` is `true`, all lines are processed but only the result of the last line is displayed. Therefore all four lines of data

```
1+1; 2+2; 3+3;
7;
8;
9;
```

are sent to Maple, but only the last result is displayed and an HTML `
` string is appended after the line.

```
9<br>
```

- If `lastline` is `true` and `breakline` is `false`, all lines are processed but only the result of the last line is displayed. Therefore all four lines of data

```
1+1; 2+2; 3+3;
7;
8;
9;
```

are sent to Maple, but only the last result is displayed.

```
9
```

- If `lastline` is `false` and `breakline` is `true`, all lines are processed and displayed. Therefore all 4 lines of data

```
1+1; 2+2; 3+3;
7;
8;
9;
```

are sent to Maple, resulting in MapleNet producing

```
2 4 6<br>
7<br>
```

```
8<br>
9<br>
```

The browser displays this as

```
2 4 6
7
8
9
```

- If `lastline` is `false`, and `breakline` is `false`, all lines are processed and displayed. Therefore all four lines of data

```
1+1; 2+2; 3+3;
7;
8;
9;
```

are sent to Maple, resulting in MapleNet producing

```
2 4 6
7
8
9
```

Note that browsers typically ignore white space and so the result is displayed as the following.

```
2 4 6789
```

Code Tag

The code tag is similar to the `<maple:statement ...>` tag, but no data is generated to the output page. This can be used to define common procedures and data for later use by expressions in the `<maple:statement ...>` sections. The form of the tag is as follows.

```
<maple:code>
  ... Maple expression ....
  ... Maple expression ....
</maple:code>
```

This tag has no parameters. Each expression in the tag body is evaluated by the Maple kernel.

Note: Unlike the statement block, all expressions are evaluated together. As such, control structures (`if/else`, `try/catch`) can span several lines.

Assign Tag

The `assign` tag allows parameters from HTML forms to be passed as variables to Maple. The form of the tag is as follows.

```
<maple:assign param="param name" variable="Maple variable"
default="def value" />
```

where

- `param name` (required) specifies the identifier of an HTML form parameter that has an associated value
- `Maple variable` (required) specifies the name of a variable in Maple. It is assigned the value of the HTML parameter
- `default` (optional) specifies the value to assign to a Maple variable if the parameter is not set

Although the `<maple:xxx>` tags appear in the source, they do not appear in the user browser because they are converted by the Maple Tag Library into content that is consistent with Web pages. The content is the output from the Maple kernel.

Plot Tag

The `plot` tag creates an interactive (dynamic image) plot. The form of the tag is as follows.

```
<maple:plot width="value" height="value">
Maple_plot_equation</maple:plot>
```

where

- `maple:plot` specifies a interactive plot
- `width` and `height` specify the numeric parameters to generate the plot
- `Maple_plot_equation` specifies the plot equation, for example,

```
plot3d(sin(x^0.5*y^0.5), x=-Pi..Pi,y=-2*Pi..2*Pi,axes=BOXED)
```

For static plots, see *Statement Tag* on page 31.

Release Tag

The `release` tag terminates the JSP connection to Maple, allowing other JSP sessions to access the Maple server. The form of the tag is as follows.

```
<maple: release />
```

This tag will immediately release Maple, regardless of the `scope` setting in the `init` tag. This is the preferred method of freeing Maple for other user sessions.

4.5 Examples

To understand more about MapleNet JavaServer Pages, review the following `integration.jsp` and `MaplePlotTest.jsp` example code listings.

Integration Example

The following example is similar to the applet example (Listing 3 on page 11) for performing simple integration. However, it uses only Web page tags and no client-side Java.

Listing 7 Integration.jsp

```
<%@ taglib prefix="maple" uri="/maplemath.tld" %>
<%@ page session="true" %>
<maple:init scope="session" debug="false" />
<maple:assign param="expr001" variable="eqn" default="sin(x)" />
<maple:code>
  doInt := proc()
    global eqn;
    int(eqn, x);
  end proc;
</maple:code>

<html>
  <head>
    <title>
      Sample Integration
    </title>
  </head>
  <body>
    <center>
      <h1>Sample Integration</h1>
      <form method="post" action="Integration.jsp" >
        <table width="50%" >
          <tr>
            <td width="35%">Enter Expression : </td>
```

```

    <td width="65%">
      <input type="text" size="40" name="expr001"
        value="<maple:statement>eqn</maple:statement>" >
    </td>
  </tr>
  <tr>
    <td width="35%">Result : </td>
    <td width="65%">
      <input type="text" size="40" name="result001"
        value="<maple:statement>doInt()</maple:statement>" >
    </td>
  </tr>
</table>
  <input type="submit" name="Calculate" value="Submit">
</form>
</center>
  <maple:statement type="plot" height="200" width="400">
    plot(eqn, x=0..10)</maple:statement>
</body>
<maple: release />
</html>

```

The following discussion refers to Listing 7 on page 36.

- The first tag, `<%@ taglib ...%>`, establishes that `maple` is the tag prefix for Maple calculations.
- The `<%@ page ...%>` tag sets the server to remember this user for a session.
- The `<maple:init ...>` tag specifies that the same kernel is used for this session and that debugging is off.

Note: If an error arises on the server side while processing of the Maple tags, the current kernel is released and any state information is lost.

- The `<maple:assign ...>` tag takes the form parameter, `expr001`, and assigns it to the Maple variable, `eqn`. The first time this page is loaded, the form has not yet been processed and `expr001` is undefined. Therefore the default value, `sin(x)`, is assigned to the variable, `eqn`.
- Following this is a `<maple:code ...>` block. The server passes the included Maple code directly to the Maple kernel. This block defines a simple procedure, `doInt()`, that integrates the expression assigned to `eqn`. This procedure is referenced later in the Web page.

- The remainder of the example is an HTML document that has two input fields inside a `<form ...>` tag. The action of the form is to invoke the same Web page, **Integration.jsp**. The `expr001` input field is filled using a `<maple:statement ...>` tag to output the value of `eqn`. The `result001` field is filled using a `<maple:statement ...>` to evaluate the `doInt()` procedure that is defined by the code block.
- The `release` tag terminates the JSP connection to the Maple server.

When the **Calculate** button is clicked, the form is submitted to the server to be processed again by **Integration.jsp**. This time the `<maple:assign ...>` tag takes the user input from the `expr001` field and assigns it to `eqn`. The new value of `eqn` is integrated and placed in the output page in the `result001` field.

Plot Example

The following code provides an example of static and interactive plots.

Listing 8 MaplePlotTest.jsp

```
<%@ taglib prefix="maple" uri="/maplemath.tld" %>
<%@ page session="true" %>
<maple:init scope="page" debug="true" />
<maple:assign param="eqn001" variable="plt_eqn" default="plot3d(sin(x^0.5*y^0.5), x=-
Pi..Pi, y=-2*Pi..2*Pi,axes=BOXED)" />
<html>
<head>
<title>
MaplePlotTest
</title>
</head>
<body bgcolor="#ffffc0">
<h1>
Test of Maple Plot Tag
</h1>
Time is <%= new java.util.Date().toString() %>
<br>
<center>
<b> Static Image plot</b>
<br>
<maple:statement type="plot" width="250" height="200"
>plot(cos(4*x),x=0..10)</maple:statement>
</center>
```

```

<br>
<center>
<b>Dynamic Image plot</b>
<br>
<maple:plot width="500" height="300" >plt_eqn</maple:plot>
</center>
<br>
<center>
<form method="post">
  <br>
  <input type="text" size="50" name="eqn001" value="plot3d(sin(x^0.5*y^0.5), x=-
Pi..Pi, y=-2*Pi..2*Pi,axes=BOXED)" >
  <br>
  <input type="submit" name="Submit" value="Submit">
  <input type="reset" value="Reset">
</form>
</center>
<maple:release/>
</body>
</html>

```

A static plot is displayed above an interactive plot. The user can submit new values for the interactive plot in an input field.

4.6 Publishing JSP Content to the MapleNet Server

The `.jsp` application file can be published to the Web server directly or using one of the following methods.

- *Using MapleNet Publishing Scripts* on page 39 (command line)
- *Using the MapleNet Publisher* on page 42 (graphical interface)

Using MapleNet Publishing Scripts

MapleNet publishing scripts bundle and send files to the server to be installed in a location accessible by the client.

To publish files to the server:

1. Place the files required for publishing in the same subdirectory.

2. Run the `CreateJar` (`createjar` in UNIX) script to wrap the contents of the subdirectory into a JAR file. A JAR file is a Java version of a zip file. As such, it can be viewed with a zip program such as WinZip 7.0.
3. Run the `PublishJar` (`publishjar` in UNIX) script to publish the contents of the JAR file on the server.

For an example procedure, see *Example Procedure: How to Create and Publish a JAR File*.

Example Procedure: How to Create and Publish a JAR File

The following example files are used to illustrate the procedure for moving files to the server.

| Directory or File | Description |
|---|----------------|
| <code>\Projects\Tutorial</code> | Main directory |
| <code>\Projects\Tutorial\integration.jsp</code> | Web page |

1. Create a JAR file using the example files.

Windows

Run the `CreateJar.bat` file from the `Projects` directory.

```
cd Projects
CreateJar tutor1.jar Tutorial
```

where `tutor1.jar` is the JAR file to be created and `Tutorial` is the subdirectory to be searched for source files.

The `CreateJar` command searches the directory (`Tutorial`) and all its subdirectories for source files and then creates the `tutor1.jar` file in the `Projects` directory.

UNIX

Run the `createjar` script from the `Projects` directory.

```
cd Projects
createjar tutor1.jar Tutorial
```

where `tutor1.jar` is the jar file to be created and `Tutorial` is the subdirectory to be searched for source files.

The `createjar` command searches the directory (`Tutorial`) and all its subdirectories for source files and then creates the `tutor1.jar` file in the `Projects` directory.

2. Verify the contents of the `tutor1.jar` file using the `jar` tool from the SDK.

Enter the command:

```
jar -tf tutor1.jar
```

All directories/files contained in the jar file are displayed. The contents will resemble the following.

```
/Projects>jar -tf tutor1.jar
META-INF/
META-INF/MANIFEST.MF
Tutorial/
Tutorial/integration.jsp
/Projects>
```

If the contents are not similar to the above listing, there was a problem with the createjar step.

3. Publish the JAR file on the server. The command parameters are described in Table 7.

Windows

Run the PublishJar.bat file.

```
PublishJar user password jarfile destination
```

UNIX

Run the publishjar command.

```
publishjar user password jarfile destination
```

For example,

```
publishjar admin spiff tutor1.jar //Servername/Projects/tutor1.jar
```

The server recreates all the files and directories contained in the JAR file. For example, the files are extracted to \Projects\Tutorial on the server.

Table 7 Command Parameters

| Parameter | Description |
|-------------|--|
| user | User name already defined on the server to which you are publishing. |
| password | Password for the user. |
| jarfile | Name of the JAR file. |
| destination | Name of the remote server and directory where the JAR file is sent and unwrapped. The directory will be placed in the Web Server's root directory. Note: The publisher must have write permission to this directory. |

Note: You can publish to the server only when the MapleNet Server and the Web Server are running. For more information, refer to the *MapleNet Installation Guide*.

Using the MapleNet Publisher

The **MapleNet Publisher** allows you to select the source content, either an archived file or folder, and to specify a target folder.

You must know the following information.

- The URL of the MapleNet server and the port on which the server listens.
- The MapleNet user account and password that has authority to publish content to the Web server.
- The location of the files to be archived, or the archive if previously created using another method.
- The target folder from which the archive is to be extracted.

To launch the MapleNet Publisher:

In the `bin` directory, double-click the `maplenetpublisher.bat` file (Windows) or run the `maplenetpublisher` file (UNIX). The **MapleNet Publisher** dialog opens.

- If you have an archive, follow the instructions in *To publish an archive*.
- If you must create an archive, follow the instructions in *To create an archive* and then *To publish an archive*.

To create an archive:

1. In the **MapleNet Publisher** dialog, click **Create Archive**. The **Jar Create Utility/Package Directory** dialog opens.
2. Enter a directory and click **Next**.
(Optional) Click **Browse** to find and select a directory.
3. In the **Jar Create Utility/Create Archive** dialog, enter the archive filename, and click **Finish**. A message is displayed indicating that the archive filename extension has been changed to “jar”.

To publish an archive:

1. In the **MapleNet Publisher** dialog, click **Publish Archive**. The **MapleNet Publisher/Choose Archive** dialog opens.
2. Enter (or **Browse** for) an archive filename. The contents of the selected archive (`.jar` file) are listed in the **Archive Contents** section of the dialog.
To review the code of a particular HTML file in the **Archive Contents**, select an `.html` file and click **Explore**. The **Zip File Browser** dialog opens, displaying the HTML code. To return to the previous dialog, close the **Zip File Browser**.

3. Once you have selected an archive, click **Next**. The **MapleNet Publisher/MapleNet Server** dialog opens.
4. Enter the **Name** of the MapleNet server, for example, `mysite.com`. The **Port** number field is automatically populated with the default **14444**.
5. Enter the **User Name** and **Password**. The default **Admin** password is **spiff**.
6. Click **Next**. The **MapleNet Publisher/Deploy** dialog opens. Enter the path and destination jar filename, for example, enter `/Projects/tutor1.jar`. The Publisher places the `tutor1.jar` on the server in the `/Projects` directory and then extracts all the files from `tutor1.jar` into this directory.
7. Click **Finish**. The **MapleNet Publisher** closes.

4.7 Accessing Published JSP Content

Demonstrating with the example files of the previous section, a user can access the published Web pages using the following URL.

`http://Servername/maplenet/Projects/Tutorial/integration.jsp`

The path is based on the `destination` parameter specified in the `PublishJar` (`publishjar` in UNIX) command. (See Table 7.)

5 Developing Worksheets for MapleNet

In this chapter

- Writing worksheets for MapleNet
- Client and server considerations
- Publishing worksheets to the MapleNet server
- Accessing published worksheets

5.1 Writing Worksheets for Use on MapleNet

MapleNet can display both standard and classic worksheets in a Web browser, even if the worksheets have not been specifically designed for it. Content is static in any worksheet displayed by MapleNet, with the following exceptions.

- Embedded components are fully functional, including plot components, buttons, toggle buttons, check boxes, math containers, combo boxes, list boxes, labels, and sliders
- Content that can be modified using embedded components behaves accordingly
- Standard plot output can be manipulated normally
- Sections can be collapsed and expanded

When displayed by MapleNet, the embedded components provide means for a user to add new input to a worksheet.

The limitations imposed on worksheets displayed by MapleNet are as follows.

- Maplets are disabled
- System calls and file I/O are disabled, by default

Worksheets that contain these disabled features can still be viewed on MapleNet, but the generated page may contain error messages indicating these operations are disabled. As a result, these worksheets may not function as expected.

When a worksheet is loaded by MapleNet, all autoexecutable commands are evaluated before that worksheet is displayed to the user. Therefore, the state of a worksheet when first opened by a user is the same as the state of the worksheet when opened and autoexecuted in Maple. As a result, the logic of a worksheet can be fully tested before being deployed to MapleNet.

Worksheets displayed on MapleNet are rendered with a fixed width. This width is specific to each worksheet, and is set in Maple by changing the math formatting width. This option is found on the **General** tab of the **Tools>Options** dialog.

All relatively sized tables in the worksheet are scaled relative to this limit, and any content that is too large is scaled to fit. That is, changing the math formatting width has the same effect on worksheet layout as resizing the window in Maple. The math formatting width is converted to approximately 100 pixels per inch.

Tips to Improve Worksheet Layout

Refrain from using tables unnecessarily. If a table provides no useful formatting, it would be best to remove it. Placing content such as images, plots, or sketches inside table cells imposes size restrictions on these elements. When the table cells are scaled to the math formatting width, these restrictions may be undesirable and unnecessary.

Maple sketch regions are cropped, not resized, when placed inside table cells. MapleNet, however, will scale the sketch to fit the cell. For sketches to be displayed the same in both instances, it is important to resize sketch regions in Maple to fit their table cells. To resize a sketch in Maple, simply drag the sketch border.

Try to avoid scaling images. The scaling of images may introduce undesirable artifacts. It is best to generate images of the correct size, and display them without scaling.

Tips to Improve Worksheet Performance

In a worksheet viewed on MapleNet, when an action is performed on an embedded component, a request is submitted to the server to execute the associated Maple code. However, if this action code is empty, no request is generated. This prevents unnecessary requests to the server. To take advantage of this optimization, if the action code of an embedded component does no real work, the code should be removed. Note that the default action code on a new component is not empty, even though its execution has no effect on the state of the worksheet.

Use collapsed sections to help improve the loading time of worksheets. If it is not necessary to have all the components visible when a worksheet is loaded, place any extraneous components in collapsed sections, to be expanded and accessed by the user as needed.

5.2 Client Considerations

Browser support

For a worksheet to have full functionality when viewed on MapleNet, the client's browser must support the following.

- Java 1.4.2 (or better)
- JavaScript
- LiveConnect (communication between JavaScript and Java applets)

There are free, downloadable browsers available for recent versions of Macintosh, Windows, and UNIX/Linux operating systems.

For browsers that do not support LiveConnect, or have JavaScript disabled, MapleNet will render a version of the worksheet that has limited functionality.

In browsers without LiveConnect:

- Embedded component sliders and math containers can be used for output only

In browsers without JavaScript:

- Embedded component sliders and math containers can be used for output only
- Actions on components, except buttons and toggle buttons, will not trigger requests to be submitted to the server

Memory Limitations

Plots, plot components, slider components, and math container components are all displayed as Java applets in the client Web browser. As a result, the amount of memory available to these elements is limited by the client Java plug-in settings. This limit is configurable, with a default value of approximately 100MB. This is more than enough memory for the applets themselves, but it does impose restrictions on the amount of data the applets can display.

If you are creating large worksheets, you may need to inform your users to increase the default limit.

Delay on First Use

As previously stated, plots, plot components, slider components, and math container components are displayed as Java applets. The first time a user

accesses a worksheet containing these elements, there will be a delay as their browser automatically downloads the applets. Once these applets have been downloaded, they are cached and do not need to be downloaded for any other worksheets at the site. This cache is persistent between browser sessions.

5.3 Server Considerations

Each worksheet opened by each user requires a server-side representation of the worksheet in memory and an instance of the Maple kernel, just as it does in Maple. The memory and processing requirements of this representation are approximately the same as they are in Maple. These resources remain allocated to the worksheet until it is closed by the user (using the **Close** button in the worksheet interface) or until the user's session expires due to inactivity.

It is possible to increase the amount of memory available to MapleNet. For more details, refer to the *Administration Guide* or consult your site administrator.

5.4 Customization of the MapleNet Worksheet Interface

When worksheets are viewed on MapleNet, an interface to the Web application is included in the header and footer of the page. It is possible to modify the look of this interface, but this customization requires a reconfiguration of the MapleNet server. For more details, please refer to the *Administration Guide*.

5.5 Publishing Worksheets to the MapleNet Server

Publishing a worksheet on a MapleNet server requires only the copying of the worksheet file into the appropriate directory on the server. Consult your site administrator to find out where your worksheets must be placed.

To publish files on the server, follow the instructions in *Publishing JSP Content to the MapleNet Server* on page 39 of this guide. Alternatively, worksheets can be published using any tool, such as FTP, SSH, or WebDAV, to transfer the files to the correct directory on the server.

5.6 Accessing Published Worksheets

Consult your site administrator for the exact URL to access your worksheets. Assuming the MapleNet server is **myserver.com** and that your worksheet

myworksheet.mw is located in the directory **worksheet** of the MapleNet Web application, it can be accessed by pointing your browser to **<http://myserver.com/maplenet/worksheet/myworksheet.mw>**.

Appendix A

The following JAR files contain MapleNet code for the client side. All applications must be compiled with these files on the classpath. These files must also be made accessible to users in your Web root when publishing MapleNet content.

- `maplenetclient.jar`
- `maplenetclient-client.jar`
- `maplenetclient-freehep.jar`
- `maplenetclient-index.jar`
- `maplenerclient-maplenet.jar`
- `maplenetclient-maplet.jar`
- `maplentclient-mathdoc.jar`
- `maplentclient-mathdoc2.jar`
- `maplenetclient-misc.jar`
- `maplentclient-openwiz2.jar`
- `maplentclient-plot.jar`
- `maplentclient-util.jar`
- `maplentclient-utilcodepage.jar`
- `maplentclient-url.jar`
- `maplentclient-webeq3.jar`

The JAR files are indexed for efficient download. Only the `maplenetclient-index.jar` must be specified in the applet archive parameter in the `.html` file. The other JAR files listed here will be downloaded as needed as long as they reside in the same directory as `maplenetclient-index.jar` in your Web root.