Solaris Java Plug-in User's Guide

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 816–0378–10
May 2002

020115@3062

Contents

Preface

This manual is an introduction to and overview of the use of version 1.4 of the Java™ Plug-in product. It is a summary of the Java Plug-in documentation available at http://java.sun.com/j2se/1.4/docs/guide/plugin.

Who Should Use This Book

This book contains information that is useful to applet developers and web site managers who host applets on their Web pages.

How This Book Is Organized

Chapter 1 provides an introduction to the Java Plug-in product.

Chapter 3 discusses tags that must be used in HTML files.

Chapter 4 describes the HTML Converter.

Chapter 5 contains information about setting up web sites that host applets.

Chapter 6 discusses support for HTTP, FTP, and GOPHER protocols.

Chapter 7 describes the Java Plug-in supports the use of cookies, which store data on client platforms.

Chapter 8 contains more information related to caching of applets.

Chapter 9 describes the operation and use of the Java Plug-in Control Panel.

Chapter 10 contains information about Netscape 6 and the Open Java Interface (OJI).

Chapter 11 provides references to the various topics covered by the online FAQ for the Java Plug-in.

Related Documentation

These documents have information about version 1.4 of the Java 2 Platform:

- *Java 2 SDK for Solaris Developer's Guide*
- *Java 2 SDK, Standard Edition v. 1.4 Release Notes* located online at http://java.sun.com/j2se/1.4/relnotes.html.
- *Java 2 SDK, Standard Edition, v. 1.4 Documentation* located online at http://java.sun.com/j2se/1.4/docs/index.html.
- *Java 2 Platform, Standard Edition, v 1.4 API Specification* located online at http://java.sun.com/j2se/1.4/docs/api/index.html.

Accessing Sun Documentation Online

The docs.sun.com^{SM} Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is `http://docs.sun.com`.

Typographic Conventions

The following table describes the typographic changes used in this book.

TABLE P–1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`machine_name%` **`su`** `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type **`rm`** *filename*.
AaBbCc123	Book titles, new words, or terms, or words to be emphasized.	Read Chapter 6 in *User's Guide*. These are called *class* options. You must be *root* to do this.

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	`#`

CHAPTER 1

Overview — What is Java Plug-in?

The Java™ Plug-in component (hereafter "Java Plug-in") extends the functionality of a web browser, allowing applets to be run under Sun's Java 2 runtime environment rather than the Java runtime environment that comes with the web browser. Java Plug-in is part of the Java 2 Runtime Environment (JRE) and is installed with it when the JRE is installed on a computer. It works with both Microsoft Internet Explorer and Netscape™ web browsers.

This functionality can be acheived in two different ways:

1. By using the conventional APPLET tag in a web page.
2. By replacing the APPLET tag with the OBJECT tag for Internet Explorer; by replacing the APPLET tag with the EMBED tag for Netscape 4; by replacing the APPLET tag with the OBJECT tag or EMBED tag for Netscape 6. Note, however, that the OBJECT and EMBED tags must conform to a special format as described in the next chapter.

`OBJECT` and `EMBED` tags in web pages may be manually updated, but to facilitate updating web pages to this new format, an HTML Converter is provided. It is described in the section called Chapter 4.

While the above constitutes the heart of Java Plug-in, there are many other related topics that you may want to understand. For instance, you may want to know how proxy configuration works in Java Plug-in, you may want to know what protocols Java Plug-in supports, or you may want to know about cookie support and caching. Such topics are included in this book.

This book also contains information on Netscape 6.1 and the Open Java Interface (OJI) and an FAQ.

The material in this book is taken from the *Java Plug-in Developer Guide*, which is part of the Java 2 Platform, Standard Edition, v1.4 documentation set. The full *Java Plug-in Developer Guide* is available online at

http://java.sun.com/j2se/1.4/docs/guide/plugin/developer_guide/contents.html. In addition to the information provided in this book, the online *Java Plug-in Developer Guide* contains additional chapters dealing with Deployment Schemes, Security, Debugging Support, and Advanced Topics.

CHAPTER 2

Using the Conventional Applet Tag

Support for the APPLET Tag

Java Plug-in now supports the HTML APPLET tag for launching applets. Users may configure their browsers so that JRE 1.4.0 is the default runtime environment for handling APPLET tags.

For developers, the Java Plug-in enhancements enable you to deploy your applet web pages without the need for an HTML Converter or the use of OBJECT tags, while still being able to ensure that your users will have the latest JRE/Java Plug-in for running your code.

Even though Java Plug-in now supports the APPLET tag, it does not support applets that make use of proprietary technologies. Examples of such technologies are

- CAB files
- Authenticode signing
- Java Moniker

Compatibility with the OBJECT Tag

This release of Java Plug-in supports the use of APPLET tags for launching applets. However, it is also fully backward compatible with previous Java Plug-in releases in its support of the OBJECT tag for launching applets. Developers have the option of using the HTML Converter to set up their applet web pages to use the OBJECT tag as before.

Supported Browsers

Java Plug-in provides support for the APPLET tag on the following web browsers:

- Internet Explorer 4.0 (4.01 recommended), 5.0 (5.01 recommended), 5.5 (Service Pack 2 recommended), 6.0
- Netscape 6.0, 6.1

Updating Old Class Files

In some cases, the new Java Runtime Environment associated with this Java Plug-in release will not run class files that were generated with old compilers. The usual symptom is a `java.lang.ClassFormatError` that the virtual machine throws when it attempts to load such a class file. This failure has nothing specifically to do with the changes in this release. Rather, old bytecode compilers did not strictly adhere to proper class-file format in all situations when generating class files. Recent virtual machines are implemented to be strict in enforcing proper class file format, and this can lead to errors when they attempt to load old, improperly formatted class files. Some typical problems in some older class files are (this list is not exhaustive):

- There are extra bytes at the end of the class file;
- The class file contains method or field names that do not begin with a letter;
- The class attempts to access private members of another class;
- The class file has other format errors, including illegal constant pool indices and illegal UTF-8 strings;
- Some early (third-party) bytecode obfuscators produced class files that violated proper class-file format.

You can avoid this type of problem by recompiling your classes with the Javac bytecode compiler from the current Java 2 SDK. If you choose to use a third-party obfuscator, be sure to use one that produces class files that respect proper class-file format.

CHAPTER 3

Using OBJECT, EMBED and APPLET Tags in Java Plug-in

This chapter includes the following topics:

Introduction

This document explains the tagging structure, involving `OBJECT` and `EMBED` tags, required by Java Plug-in. It is intended for web authors who want to manually insert Java Plug-in tags in their HTML pages.

Note – There is a Java Plug-in HTML Converter, available free-of-charge from Sun Microsystems, that automatically does this for you. It is highly recommended that most web authors use it.

Applets are normally specified in an HTML file as follows:

```
<APPLET code="XYZApp.class" codebase="html/" align="baseline"
          width="200" height="200">
        <PARAM name="model" value="models/HyaluronicAcid.xyz">
          No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!
        </APPLET>
```

And normally the APPLET tag specifies information about the applet, while the <PARAM> tags, located between the <APPLET> and </APPLET> tags, store per-instance applet information.

However, an APPLET is rendered by the browser and there is no easy way to intercept the browser and force it to use Sun's Java Runtime Environment (JRE) to run the applet. To force the browser to do so, however, you may use a special Java Plug-in tagging structure involving the OBJECT or EMBED tag or both, as described below, in place of the usual APPLET tag in your HTML pages. This will cause the browser to launch Java Plug-in, which will then run the applet using Sun's JRE.

For various combinations of browsers and platforms, the following sections tell you exactly what you need to do.

Note – Product version numbers are of the form

`n1.n2.n3_n4n5`

where n1.n2 is the major version number, the n3 is the minor version number (also referred to as the maintenance version number), and n4n5 is the update version number.

The version numbers used in the examples below refer to the 1.4 major release with minor release number of 0 and, at times, hypothetical update number.

At times you will see modified usages such as 1,4,0,nm and 14 (the latter to indicate only the major version number).

Java Plug-in in Internet Explorer on Microsoft Windows Platforms

To use Java Plug-in with Internet Explorer on Microsoft Windows platforms, use the OBJECT tag. The following is an example of mapping an APPLET tag to a Java Plug-in tag:

Original APPLET Tag

```
<APPLET code="XYZApp.class" codebase="html/" align="baseline"
          width="200" height="200">
       <PARAM NAME="model" VALUE="models/HyaluronicAcid.xyz">
          No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!
       </APPLET>
```

Note – The URL given in the *codebase* attribute in this and following examples is for illustrative purposes only. No cab file actually exists at that URL.

New OBJECT Tag

```
<OBJECT classid="clsid:CAFEEFAC-0014-0000-0000-ABCDEFFEDCBA"
width="200" height="200" align="baseline"
codebase="http://java.sun.com/jpi/jinstall-14-win32.cab#Version=1,4,0,mn">
<PARAM NAME="code" VALUE="XYZApp.class">
<PARAM NAME="codebase" VALUE="html/">
<PARAM NAME="type" VALUE="application/x-java-applet;jpi-version=1.4">
<PARAM NAME="model" VALUE="models/HyaluronicAcid.xyz">
<PARAM NAME="scriptable" VALUE="true">
        No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!
       </OBJECT>
```

Note that the OBJECT tag contains similar information to the APPLET tag. It is sufficient to launch Java Plug-in in Internet Explorer. The classid in the OBJECT tag is the class identifier for Java Plug-in itself. This class identifier should be the same in every HTML page. When Internet Explorer renders this class identifier in the OBJECT tag, it will try to load Java Plug-in into the browser.

There are several attributes in the OBJECT tag, such as width, height and align, that are mapped directly from the corresponding attributes in the APPLET tag. These contain formatting information that Internet Explorer will use to position Java Plug-in. Since this information is mapped directly without changes, the position and appearance of the applets using Java Plug-in should be the same as those applets using the APPLET tag.

Not all attributes in the APPLET tag can be mapped to the OBJECT tag attributes. For example, the attributes *code* and *codebase* in the APPLET tag are not mapped into the OBJECT tag attribute. Instead, the attribute *code* is mapped into the PARAM code because, according to the HTML specification, the attribute *code* does not exist in the OBJECT tag. There are other attributes that do not correspond in the OBJECT tag attributes. These attributes, with one exception, should be mapped to PARAM tags.

Note – Duplicate parameter names should never be used with the OBJECT tag.

The one exception is the *codebase* attribute. In the APPLET tag, the *codebase* attribute represents the location from which to download additional class and jar files. However, in the OBJECT tag, the codebase attribute represents the location from which to download Java Plug-in when it is not found on the local machine. Because the codebase attribute has two different meanings in the APPLET and OBJECT tags, you must resolve this conflict by mapping this attribute into a PARAM codebase in the OBJECT tag.

In the above example, the *code* and *codebase* attributes in the APPLET tag are mapped into the OBJECT tag parameters. The PARAM code identifies the applet, and its value should be the same as the code attribute in the APPLET tag. The PARAM codebase identifies the codebase of the applet. Java Plug-in knows where to download the applet because it can read this information from the parameters. The parameter type is

not mapped from the APPLET tag, but it is required in the OBJECT tag. It identifies the type of the Java executable, such as an applet, so Java Plug-in knows how to initialize the Java executable. These three PARAM tags (code, codebase, and type) in the above example are specified by Java Plug-in. They do not exist in the PARAM of the original APPLET tag. Note that the model parameter within the OBJECT tag is identical to the model parameter inside the APPLET tag. Except for these first three parameters specified for Java Plug-in, the remainder of the parameters are the same as those inside the APPLET tag.

A new addition for Java Plug-in 1.3 (and valid in 1.4) was the PARAM *scriptable* tag. This was added to improve performance of applets that do not require the use of JavaScript or VBScript. The value should be *true* if the applet requires scripting support and *false* if it does not. The value is *false* by default.

Please note that PARAM *scriptable* is not the same as the PARAM *Mayscript*. *Mayscript* provides support for communication from Java applets to JavaScript only, while *scriptable* allows communication from JavaScript to Java applets in Internet Explorer only.

The text "No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!" in the APPLET tag is mapped inside the <OBJECT> and </OBJECT>tags. Originally, this text is displayed only if the browser does not have Java support. By mapping it inside the OBJECT tag, this text will displayed if the browser does not support the OBJECT tag.

The APPLET-OBJECT tag attributes mapping is as follows:

Attributes	APPLET tag support	OBJECT tag support	Attribute map in OBJECT tag
ALIGN	X	X	Attribute ALIGN
ALT	X		
ARCHIVE	X		Param archive
CODE	X	X	Param code
CODEBASE	X	X	Param codebase
HEIGHT	X	X	Attribute HEIGHT
HSPACE	X	X	Attribute HSPACE
NAME	X	X	Attribute NAME, Param NAME
OBJECT	X		Param object
TITLE	X	X	Attribute TITLE
VSPACE	X	X	Attribute VSPACE

Attributes	APPLET tag support	OBJECT tag support	Attribute map in OBJECT tag
WIDTH	X	X	Attribute WIDTH
MAYSCRIPT	X	X	Param MAYSCRIPT

Some attributes are special to the `OBJECT` tag. These attributes are:

Attribute/Param	Meaning in OBJECT tag
Attribute classid	It should always have the same value for dynamic version support, i.e. clsid:8AD9C840-044E-11D1-B3E9-00805F499D93. For static version support it will have a unique value for the version, e.g., clsid:CAFEEFAC-0014-0000-0000-ABCDEFFEDCBA. *Note:* The examples in this section use the dynamic version.
Attribute CODEBASE	It should be a full URL pointing to a CAB file somewhere on the network. It should, be default, point to the binary on the Java Software web site.
Param type	If it is a Java applet, the value should be "application/x-java-applet;jpi-version=1.4" or "application/x-java-applet". If it is a JavaBeans component, the value should be "application/x-java-bean;jpi-version=1.4" or "application/x-java-bean".
Param codebase	Specifies the base URL of the applet. This attribute is optional.
Param code	Specifies the name of the Java applet or JavaBeans component. It cannot be used the param `object` inside the same `OBJECT` tag.
Para scriptable	Specifies whether the applet is scriptable from the HTML page using JavaScript or VBScript. The value can be either *true* or *false*. This attribute is new in Java Plug-in 1.4.
Param object	Specifies the name of the serialized Java applet or JavaBeans component. It cannot be used with param `code` inside the same `OBJECT` tag. This attribute is optional.
Param archive	Specifies the name of the Java archive. This attribute is optional.
Param mayscript	Specifies whether the applet is allowed to access `netscape.javascript.JSObject`. The value can be either *true* or *false*. This attribute is optional.

Note that if the original APPLET tag has PARAM *type*, *codebase*, *code*, *object* or *archive*, mapping it to the OBJECT tag will cause a problem because duplicate param names will occur. To avoid this, Java Plug-in also supports another set of param names, as follows:

Original Param Names	New Param Names
code	java_code
codebase	java_codebase
archive	java_archive
object	java_object
type	java_type

You should use these new param names only when necessary. If both the new and original param names exist in the same OBJECT tag, the value associated with the new param name is always used by Java Plug-in to load the applet or JavaBean.

Java Plug-in in Netscape Navigator on Microsoft Windows and Linux Platforms or Solaris™ operating environments

To use Java Plug-in in Netscape Navigator™ 4 browsers on Microsoft Windows and Linux platforms or Solaris™ operating environments, use the EMBED tag. The following example maps an APPLET tag to a Java Plug-in EMBED tag:

Original APPLET tag:

```
<APPLET code="XYZApp.class" codebase="html/" align="baseline"
          width="200" height="200">
<PARAM NAME="model" VALUE="models/HyaluronicAcid.xyz">
          No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!
</APPLET>
```

New EMBED tag:

```
<EMBED type="application/x-java-applet;jpi-version=1.4" width="200"
        height="200" align="baseline" code="XYZApp.class"
        codebase="html/" model="models/HyaluronicAcid.xyz"
        pluginspage="http://java.sun.com/jpi/plugin-install.html">
<NOEMBED>
No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!
</NOEMBED>
</EMBED>
```

Note that the EMBED tag contains similar information to the APPLET tag, and it is sufficient to launch Java Plug-in in Netscape Navigator browsers. The attribute type in the EMBED tag is used for identifying the type of the Java programming language executable, such as an applet or a bean. When a Netscape Navigator browser renders this attribute in the EMBED tag, it will try to load Java Plug-in into the browser. By specifying the type attribute, Java Plug-in will know how to initialize the Java programming language executable.

In the above example, several attributes in the EMBED tag, such as width, height and align, map directly from the corresponding attributes in the APPLET tag. These contain formatting information that a Netscape Navigator browser uses to position Java Plug-in. Since this information is mapped directly without changes, the position and appearance of the applets using Java Plug-in should be the same as those applets using the APPLET tag.

Unlike the OBJECT tag, all information must be stored inside the <EMBED> tag instead of using PARAM. Therefore, all attributes and params in the APPLET tag must be mapped as attribute-value pairs inside the EMBED tag.

In the above example, the *code* and *codebase* attributes in the APPLET tag are mapped into the EMBED tag attributes. Attribute *code* identifies the applet. Its value should be the same as the *code* attribute in the APPLET tag. Attribute *codebase* identifies the codebase of the applet. Java Plug-in knows where to download the applet or JavaBeans component because it can read this information from the attributes. Also notice that the model attribute within the EMBED tag is mapped from the model param inside the APPLET tag.

Like the *codebase* attribute in the OBJECT tag, attribute *pluginspage* in the EMBED tag is used by Netscape Navigator browsers if Java Plug-in is not installed. It should always point to the Java Plug-in Download Page on the Java Software web site.

The text "No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!" in the APPLET tag is mapped inside the <NOEMBED> and </NOEMBED> tags. Originally, this text is displayed only if the browser does not have Java technology support. By mapping it inside the NOEMBED tag, this text will be displayed if the browser does not support the EMBED tag or if the browser fails to start the Java Plug-in.

The APPLET-EMBED tag attributes mapping is as follows:

Attributes	APPLET tag support	EMBED tag support	Attribute map in EMBED tag
ALIGN	X	X	Attribute ALIGN
ALT	X	X	Attribute ALT
ARCHIVE	X		Attribute archive

Attributes	APPLET tag support	EMBED tag support	Attribute map in EMBED tag
CODE	X		Attribute code
CODEBASE	X		Attribute codebase
HEIGHT	X	X	Attribute HEIGHT
HSPACE	X	X	Attribute HSPACE
NAME	X	X	Attribute NAME
OBJECT	X		Attribute object
TITLE	X	X	Attribute TITLE
VSPACE	X	X	Attribute VSPACE
WIDTH	X	X	Attribute WIDTH
MAYSCRIPT	X	X	Attribute MAYSCRIPT

Some attributes are special to the `EMBED` tag. These attributes are:

Attribute	Meaning in EMBED tag
Attribute type	If it is an applet, the value should be "application/x-java-applet;jpi-version=1.4" or "application/x-java-applet". If it is a bean, the value should be "application/x-java-bean;jpi-version=1.4" or "application/x-java-bean".
Attribute codebase	Specifies the base URL of the applet. This attribute is optional.
Attribute code	Specifies the name of the Java applet or JavaBeans component. It cannot be used with param object inside the same EMBED tag.
Attribute object	Specifies the name of the serialized Java applet or JavaBeans component. It cannot be used with param code inside the same EMBED tag. This attribute is optional.
Attribute archive	Specifies the name of the Java archive. This attribute is optional.
Attribute pluginspage	It should be a full URL pointing to an HTML page somewhere on the network.

Attribute	Meaning in EMBED tag
Attribute mayscript	Specifies whether the applet is allowed to access netscape.javascript.JSObject. The value can be either "true" or "false". This attribute is optional.

Similar to the `OBJECT` tag case, if the original `APPLET` tag has `PARAM` *type*, *codebase*, *code*, *object*, or *archive*, mapping it to the `EMBED` tag attribute will cause a problem. To avoid this, Java Plug-in also supports the same new set of attribute names, as follows:

Original Attribute Names	New Attribute Names
code	java_code
codebase	java_codebase
archive	java_archive
object	java_object
type	java_type

You should use these new attribute names only necessary. If both new and original attribute names exist in the same `EMBED` tag, the value associated with the new attribute name is always used by Java Plug-in to load the applet or bean.

Java Plug-in in Internet Explorer and Netscape Navigator Browsers

The `OBJECT` tag in Internet Explorer and the `EMBED` tag in Netscape Navigator browsers allows your HTML page to use Java Plug-in if the HTML page is browsed on Microsoft Windows platforms or Solaris operating environments. However, if the HTML page is on the Internet/intranet, the page is likely to be browsed by both Internet Explorer and Netscape Navigator browsers. You should activate the Java Plug-in if both Netscape Navigator browsers and Internet Explorer will browse the same HTML page. You can do this using the Java Plug-in `OBJECT` tag, as follows:

Original APPLET tag:

```
<APPLET code="XYZApp.class" codebase="html/" align="baseline"
          width="200" height="200">
<PARAM NAME="model" VALUE="models/HyaluronicAcid.xyz">
No Java 2 SDK, Standard Editoin v 1.4 support for APPLET!!
</APPLET>
```

New OBJECT tag:

```
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
width="200" height="200" align="baseline"
codebase="http://java.sun.com/jpi/jinstall-14-win32.cab#Version=1,4,0,mn">
<PARAM NAME="code" VALUE="XYZApp.class">
<PARAM NAME="codebase" VALUE="html/">
<PARAM NAME="type" VALUE="application/x-java-applet;jpi-version=1.4">
<PARAM NAME="model" VALUE="models/HyaluronicAcid.xyz">
<PARAM NAME="scriptable" VALUE="true">
<COMMENT>
<EMBED type="application/x-java-applet;jpi-version=1.4" width="200"
    height="200" align="baseline" code="XYZApp.class"
    codebase="html/" model="models/HyaluronicAcid.xyz"
    pluginspage="http://java.sun.com/jpi/plugin-install.html">
<NOEMBED>
</COMMENT>
        No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!
</NOMEMBED></EMBED>
</OBJECT>
```

Because Internet Explorer understands the `</OBJECT>` tag, it will try to launch Java Plug-in. Notice that the `<COMMENT>` tag is a special HTML tag understood only by Internet Explorer. Internet Explorer ignores test between the `<COMMENT>` and `</COMMENT>` tags. In effect, the above tags actually become:

```
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
width="200" height="200" align="baseline"
codebase="http://java.sun.com/jpi/jinstall-14-win32.cab#Version=1,4,0,mn">
<PARAM NAME="code" VALUE="XYZApp.class">
<PARAM NAME="codebase" VALUE="html/">
<PARAM NAME="model" VALUE="models/HyaluronicAcid.xyz">
<PARAM NAME="type" VALUE="application/x-java-applet;jpi-version=1.4">
<PARAM NAME="scriptable" VALUE="true">
   No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!
</NOEMBED></EMBED>
</OBJECT>
```

This is identical to the OBJECT tag example outlined above. The `</NOEMBED>` and `</EMBED>` and tags are ignored by the `OBJECT` tag because there are no corresponding `<NOEMBED>` and `<EMBED>` tags.

Because Netscape Navigator browsers do not understand the `OBJECT` and `COMMENT` tags, they read the above tags as follows:

```
<EMBED type="application/x-java-applet;jpi-version=1.4"
    width="200" height="200"
    align="baseline" code="XYZApp.class" codebase="html/"
    model="models/HyaluronicAcid.xyz"
    pluginspage="http://java.sun.com/jpi/plugin-install.html">
<NOEMBED>
    No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!
</NOEMBED>
</EMBED>
```

This is identical to the EMBED tag example outlined above. A Netscape Navigator browser ignores tags because they are an HTML extension in an Internet Explorer browser only.

This example illustrates that you can use the combined OBJECT-EMBED tag to activate Java Plug-in in the browser if either Internet Explorer or a Netscape Navigator browser is used. This combined tag is strongly recommended unless your HTML page is browsed by users in a homogeneous environment. The Java Plug-in HTML Converter from Sun Microsystems automatically converts HTML pages into this tag style for you.

Java Plug-in Anywhere

In an Internet/intranet environment, an HTML page is likely to be seen by browsers on different platforms. You should activate Java Plug-in only on the right browser and platform combination. Otherwise, you should use the browser's default JVM. You can achieve this using the following Java Plug-in tag:

Original APPLET tag:

```
<APPLET code="XYZApp.class" codebase="html/" align="baseline"
          width="200" height="200">
<PARAM NAME="model" VALUE="models/HyaluronicAcid.xyz">
          No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!
</APPLET>
```

New style tag:

The following is an example of an equivalent Java Plug-in tag. This example includes comments.

```
<!-- The following code is specified at the beginning of the <BODY> tag. -->
<SCRIPT LANGUAGE="JavaScript"><!--
 var _info = navigator.userAgent; var _ns = false;
      var _ie = (_info.indexOf("MSIE") > 0 && _info.indexOf("Win") > 0
          && _info.indexOf("Windows 3.1") < 0);
  //--></SCRIPT>
<COMMENT><SCRIPT LANGUAGE="JavaScript1.1"><!--
var _ns = (navigator.appName.indexOf("Netscape") >= 0
    && ((_info.indexOf("Win") > 0 && _info.indexOf("Win16") < 0
    && java.lang.System.getProperty("os.version").indexOf("3.5") < 0)
    || _info.indexOf("Sun") > 0));
  //--></SCRIPT></COMMENT>

<!-- The following code is repeated for each APPLET tag -->
<SCRIPT LANGUAGE="JavaScript"><!--
      if (_ie == true) document.writeln('
<OBJECT
classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
width="200" height="200" align="baseline"
codebase="http://java.sun.com/jpi/jinstall-14-win32.cab#Version=1,4,0,mn">
```

```
      <NOEMBED><XMP>');
      else if (_ns == true) document.writeln('
<EMBED
      type="application/x-java-applet;jpi-version=1.4"
      width="200" height="200"
      align="baseline" code="XYZApp.class" codebase="html/"
      model="models/HyaluronicAcid.xyz"
      pluginspage="http://java.sun.com/jpi/plugin-install.html">
      <NOEMBED><XMP>');
//--></SCRIPT>
      <APPLET code="XYZApp.class" codebase="html/" align="baseline"
          width="200" height="200">
</XMP>
<PARAM NAME="java_code" VALUE="XYZApp.class">
<PARAM NAME="java_codebase" VALUE="html/">
<PARAM NAME="java_type" VALUE="application/x-java-applet;jpi-version=1.4">
<PARAM NAME="model" VALUE="models/HyaluronicAcid.xyz">
<PARAM NAME="scriptable" VALUE="true">
      No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!
</APPLET></NOEMBED></EMBED>
</OBJECT>

<!--
      <APPLET code="XYZApp.class" codebase="html/" align="baseline"
          width="200" height="200">
      <PARAM NAME="model" VALUE="models/HyaluronicAcid.xyz">
      No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!
      </APPLET>
-->
```

Although this tag seems complicated compared to the old `APPLET` tag, it is not. Most of the Java Plug-in tag is the same regardless of the applet used. For the majority of cases, a webmaster can copy and paste the Java Plug-in tag.

The first block of the script extracts the browser and platform. You must determine the browser and platform on which the applet is running. You do this by using JavaScript™ to extract first the browser name, then the platform. This is done once per HTML document.

The second block of the script replaces the `APPLET` tag. You must replace each `APPLET` tag with a similar block of code. The script replaces the `APPLET` tag with either an `EMBED` tag or `OBJECT` tag, depending on the browser. You use the `OBJECT` tag for Internet Explorer and the `EMBED` tag for Netscape Navigator browsers. Finally, the original `APPLET` tag is included as a comment at the end. It is always a good idea to keep the original `APPLET` tag in case you want to remove the Java Plug-in invocation.

The first JavaScript establishes the browser and the platform on which the browser is running. You must do this because, currently, Java Plug-in supports only Microsoft Windows platforms and the Solaris operating environment. Note that Windows NT 3.51 is the only Win32 platform that Java Plug-in does not support. Java Plug-in should be invoked only on the supported browser and platform. The script sets the

variable *_ie* to true if the browser is Internet Explorer. It sets the variable *_ns* to true if the browser is a Netscape browser. (Note that all variable names in the JavaScript start with "_". This is done to avoid conflicting with other JavaScript variables in the same page.)

To detect the right browser, the JavaScript evaluates three strings that are within the JavaScript's Navigator object: *userAgent*, *appVersion*, and *appName*. These strings contain information about the browser and the platform. By looking at some examples of the string *userAgent*, you can see how to evaluate *userAgent* and use it to determine the browser. The following are some examples of the *userAgent* string for different platforms as it appears in Internet Explorer. In the table, IE stands for Internet Explorer.

Platform and Browser	**JavaScript's** *Navigator.userAgent* **string**
Windows 2000 w/IE 5.0	"Mozilla/4.0 (compatible; MSIE 5.01; Window NT5.0)
Windows NT 4.0 w/ IE 4.0	"Mozilla/4.0 (compatible; MSIE 4.0; Windows NT)" “
Windows 95 w/IE 4.0	"Mozilla/4.0 (compatible; MSIE 4.0; Windows 95)“
Windows NT 3.51 w/ IE 4.0	"Mozilla/4.0 (compatible; MSIE 4.0; Windows 3.1)“
Windows 3.1 w/IE 4.0	"Mozilla/4.0 (compatible; MSIE 4.0; Windows 3.1)“
Windows NT 4.0 w/Navigator 4.04	"Mozilla/4.04 [en] (WinNT; I)“
Windows NT 3.51 w/Navigator 4.04	"Mozilla/4.04 [en] (WinNT; I)“
Windows 95 w/Navigator 4.03	"Mozilla/4.03 [en] (Win95; I)“
Solaris 2.6 w/Navigator 4.02	"Mozilla/4.02 [en] (X11; l; SunOS 5.6 sun4u)

Note that in each case the substring "MSIE" is always in the *userAgent* string in Internet Explorer. Also, the *userAgent* string in Internet Explorer under Windows 3.1 and Windows 3.51 would contain the substring "Windows 3.1" because Internet Explorer in these platforms is 16-bit. While Internet Explorer 4 is also available on Macintosh and the Solaris operating environment, in these versions the *userAgent* string does not contain the substring "Win". In addition, Internet Explorer on Windows CE does not support JavaScript. This can be summarized as follows:

userAgent **string / Browser**	**contains “MSIE”**	**contains “Win”**	**does not contain “Windows 3.1”**
Windows 3.1 w/ IE 4	X	X	

userAgent **string / Browser**	**contains "MSIE"**	**contains "Win"**	**does not contain "Windows 3.1"**
Windows NT 3.51 w/ IE 4	X	X	
Windows 95 w/IE 4	X	X	X
Windows NT 4.0 w/ IE 4	X	X	X
Windows CE w/ IE			
Mac w/ IE	X		X
UNIX w/ IE	X		X
Other browsers on any platform			X

The above table shows that only Windows 95 and Windows NT 4.0 with Internet Explorer can pass the Java Plug-in browser and platform requirements. However, this logic makes no assumptions about future releases of Internet Explorer or future releases of Microsoft Windows with Internet Explorer. As long as the *userAgent* string contains "MSIE" and "Win", the above code should work in future releases of Internet Explorer on Win32.

The above logic summarizes into the following:

```
<SCRIPT LANGUAGE="JavaScript">
          var _info = navigator.userAgent;
          var _ie = (_info.indexOf("MSIE") > 0 && _info.indexOf("Win") > 0
                  && _info.indexOf("Windows 3.1") < 0);
//--></SCRIPT>
```

It is harder to detect Navigator on the right platform. Using just JavaScript, there is no way to determine if the browser is running on the Windows NT 3.51 or Windows NT 4.0 operating platform. (Refer to the above table and examine the *userAgent* string. Notice that the *userAgent* strings in Windows NT 3.51 and Windows NT 4.0 operating platforms are the same in Netscape Navigator browsers.) It is important to make this distinction because Java Plug-in supports only the Windows NT 4.0 operating platform. To run Java Plug-in on the right platform, you must use LiveConnect in Netscape Navigator browsers to determine the OS version number. This can be summarized as follows:

Testing logic / Browser	appName contains “Netscape”	userAgent contains "Win"	userAgent does not contain "Win16"	os.version does not contain 3.5	userAgent contains "Sun"
Windows 3.1 w/ Netscape 4	X	X		X	
Windows NT 3.51 w/ Netscape 4	X	X	X		
Windows 95 w/ Netscape 4	X	X	X	X	
Windows NT 4.0 w/ NS 4	X	X	X	X	
NS on Solaris	X				X
IE on Solaris					X
Netscape on other platform	X		X	Depends on OS	
IE on other platform		X	X		
Other browsers on any platform					

The above logic translates into the following code:

```
<SCRIPT LANGUAGE="JavaScript">
         var _info = navigator.userAgent; var _ns = false;
//--></SCRIPT>
<COMMENT><SCRIPT LANGUAGE="JavaScript1.1">
    var _ns = (navigator.appName.indexOf("Netscape") >= 0
        && ((_info.indexOf("Win") > 0  && _info.indexOf("Win16") < 0
        && java.lang.System.getProperty("os.version").indexOf("3.5") < 0)
        || _info.indexOf("Sun") > 0));
//--></SCRIPT></COMMENT>
```

Referring to the previous table, note that only Windows 95, Windows NT 4.0, and Solaris operating environments with Netscape Navigator browsers pass all the tests. Because LiveConnect is used to get the OS version number and only Netscape Navigator browsers supports LiveConnect, a JavaScript that accesses LiveConnect will not be understood by Internet Explorer. To prevent this from causing a problem, you block out this piece of the script using the `COMMENT` tag since `COMMENT` is an Internet

Explorer-specific comment tag. The text between the COMMENT tag is ignored by Internet Explorer but not by Netscape Navigator browsers. In addition, you must specify the script language as JavaScript1.1 to block this out if the browser is Netscape Navigator 2.

At this point, the above logic for Internet Explorer and Netscape Navigator browsers summarizes to a script that should look as follows:

```
<!-- The following code is specified at the beginning of the <BODY> tag. -->
<SCRIPT LANGUAGE="JavaScript"><!--
    var _info = navigator.userAgent; var _ns = false;
    var _ie = (_info.indexOf("MSIE") > 0 && _info.indexOf("Win") > 0
                && _info.indexOf("Windows 3.1") < 0);
//--></SCRIPT>
<COMMENT><SCRIPT LANGUAGE="JavaScript1.1"><!--
    var _ns = (navigator.appName.indexOf("Netscape") >= 0
        && ((_info.indexOf("Win") > 0 && _info.indexOf("Win16") < 0
        && java.lang.System.getProperty("os.version").indexOf("3.5") < 0
        || _info.indexOf("Sun") > 0));
//--></SCRIPT></COMMENT>
```

Remember that this block of JavaScript should be put at the top of the <BODY> of the HTML file. It is put at the top so that other JavaScripts can reference the variables *_ie* and *_ns*. This JavaScript is the same in all HTML files, and it is only needed once for each HTML body.

The second block of HTML tags are actually the corresponding OBJECT and EMBED tags that are mapped from the data in the APPLET tag. Note that JavaScript outputs the OBJECT tag when the browser is Internet Explorer running on the Windows 95, Windows 98 or Windows NT 4.0 operating environments. If the browser is the Netscape Navigator 4 browser on Windows 95, Windows 98, Windows NT 4.0, or Solaris operating environments, then JavaScript also outputs the EMBED tag, though with a slightly different syntax. Recall that the mechanism for detecting the browser and the platform has been described in the above section. (Tags <!— and –> are used for comments in HTML.)

Original APPLET tag:

```
<APPLET code="XYZApp.class" codebase="html/" align="baseline"
          width="200" height="200">
<PARAM NAME="model" VALUE="models/HyaluronicAcid.xyz">
          No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!
</APPLET>
```

New style tag:

```
<SCRIPT LANGUAGE="JavaScript"><!--
if (_ie == true) document.writeln('<OBJECT
classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
width="200" height="200" align="baseline"
codebase="http://java.sun.com/jpi/jinstall-14-win32.cab#Version=1,4,0,mn">
<NOEMBED><XMP> ');
    else if (_ns == true) document.writeln('<EMBED
```

```
    type="application/x-java-applet;version=1.4" width="200" height="200"
      align="baseline" code="XYZApp.class" codebase="html/"
      model="models/HyaluronicAcid.xyz"
      pluginspage="http://java.sun.com/jpi/plugin-install.html">
<NOEMBED><XMP>');
//--></SCRIPT>
<APPLET code="XYZApp.class" codebase="html/" align="baseline"
          width="200" height="200"></XMP>
<PARAM NAME="java_code" VALUE="XYZApp.class">
<PARAM NAME="java_codebase" VALUE="html/">
<PARAM NAME="java_type" VALUE="application/x-java-applet;version=1.4">
<PARAM NAME="model" VALUE="models/HyaluronicAcid.xyz">
          LUE="models/HyaluronicAcid.xyz">
<PARAM NAME="scriptable" VALUE="true">
          No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!
</APPLET></NOEMBED></EMBED> </OBJECT>
```

Note that the original `APPLET` tag is also mapped in the new Java Plug-in tag. This is done because Java Plug-in is intended to be used only on supported platforms. Leaving the `APPLET` tag in the script ensures that browsers that do not support Java Plug-in, or browsers that do not support JavaScript can gracefully handle the applet using the default JVM. HotJava Browser, Internet Explorer, and Netscape Navigator browsers on non-Java Plug-in supported platforms, or browsers without JavaScript support, read the above tags as follows:

```
<APPLET code="XYZApp.class" codebase="html/" align="baseline"
          width="200" height="200"></XMP>
<PARAM NAME="java_code" VALUE="XYZApp.class">
<PARAM NAME="java_codebase" VALUE="html/">
<PPARAM NAME="model" VALUE="models/HyaluronicAcid.xyz">
<PARAM NAME="scriptable" VALUE="true">
         No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!
<PARAM NAME="java_type" VALUE="application/x-java-applet;jpi-version=1.4">
</APPLET></NOEMBED></EMBED></OBJECT>
```

These browsers ignore the tags `</XMP>`, `</OBJECT>`, `</EMBED>`, and `</NOEMBED>` as well because there is no corresponding `<XMP>`,`<OBJECT>`, `<EMBED>`, and `<NOEMBED>` tags. Because Java Plug-in is targeted for features in the Java 2 SDK, Standard Edition v 1.4 or future releases, those browsers without full Java 2 SDK 1.4 support and who do not support Java Plug-in will display the message "No Java 2 SDK, Standard Edition v 1.4 support for APPLET".

Unlike the previous examples, the mapped `PARAM` names contain *java_code*, *java_codebase*, and *java_type* instead of *code*, *codebase*, and *type*. This is necessary because specifying code and codebase in the `<PARAM>` inside the `<APPLET>` and `</APPLET>` tag causes problems in some browsers.

Internet Explorer on Windows 95, Windows 98 or Windows NT 4.0 reads the tags as follows:

```
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
width="200" height="200" align="baseline"
codebase="http://java.sun.com/jpi/jinstall-14-win32.cab#Version=1,4,0,mn">
```

```
<NOEMBED><XMP>
<APPLET code="XYZApp.class" codebase="html/" align="baseline"
          width="200" height="200"></XMP>
<PARAM NAME="java_code" VALUE="XYZApp.class">
<PARAM NAME="java_codebase" VALUE="html/">
<PARAM NAME="model" VALUE="models/HyaluronicAcid.xyz">
<PARAM NAME="java_type" VALUE="application/x-java-applet;jpi-version=1.4">
<PARAM NAME="scriptable" VALUE="true">
          No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!
</APPLET></NOEMBED></EMBED>
</OBJECT>
```

Be careful when you use the <XMP> tag. Because Internet Explorer renders the <OBJECT> tag, you must disable the <APPLET> tag. If not disabled, two applets will simultaneously show up in the browser--one applet will be running in Microsoft's JVM, and the other will be running in Sun's JVM using Java Plug-in. The <XMP> tag provides a solution. The <XMP> and </XMP> tags basically transform any HTML tag that occurs between them into a stream of static text. In the above example, the <XMP> and </XMP> tags cause the browser to treat the <APPLET> tag as static text instead of an HTML tag. Because the browser ignores any static text between the <OBJECT> tag and the <PARAM> tag, the above tags actually become:

```
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
width="200" height="200" align="baseline"
codebase="http://java.sun.com/jpi/jinstall-14-win32.cab#Version=1,4,0,mn">
<PARAM NAME="java_code" VALUE="XYZApp.class">
<PARAM NAME="java_codebase" VALUE="html/">
<PARAM NAME="java_type" VALUE="application/x-java-applet;version=1.4">
<PARAM NAME="model" VALUE="models/HyaluronicAcid.xyz">
<PARAM NAME="scriptable" VALUE="true">
   No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!
</OBJECT>
```

This is identical to the OBJECT tag example outlined above. Note that the <OBJECT> tag ignores the <NOEMBED>, </NOEMBED>, and <EMBED> tags.

Netscape Navigator 4 on Windows 95, Windows 98 or Windows NT 4.0 operating environments reads tags as follows:

```
<EMBED type="application/x-java-applet;jpi-version=1.4"
         width="200" height="200"
         align="baseline" code="XYZApp.class" codebase="html/"
         model="models/HyaluronicAcid.xyz"
         pluginspage="http://java.sun.com/jpi/plugin-install.html">
<NOEMBED><XMP>
<APPLET code="XYZApp.class" codebase="html/" align="baseline"
         width="200" height="200"></XMP>
<PARAM NAME="java_code" VALUE="XYZApp.class">
<PARAM NAME="java_codebase" VALUE="html/">
<PARAM NAME="java_type" VALUE="application/x-java-applet;jpi-version=1.4">
<PARAM NAME="model" VALUE="models/HyaluronicAcid.xyz">
<PARAM NAME="scriptable" VALUE="true">
         No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!
</APPLET></NOEMBED></EMBED></OBJECT>
```

Note that the <XMP> tag is used again in the <EMBED> tag to also disable the <APPLET> tag. The <EMBED> tag ignores the <PARAM> and </OBJECT> tags as well. In effect, the above tags actually become:

```
<EMBED type="application/x-java-applet;jpi-version=1.4"
         width="200" height="200"
         align="baseline" code="XYZApp.class" codebase="html/"
         model="models/HyaluronicAcid.xyz"
         pluginspage="http://java.sun.com/jpi/plugin-install.html">
<NOEMBED>
        No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!
</NOEMBED>
</EMBED>
```

This is identical to the EMBED tag example outlined above.

You can use the combined OBJECT-EMBED-JavaScript tag to activate Java Plug-in in the right browser on the right platform. This combined tag is complicated and it should be used only if your HTML page is browsed by users in a heterogeneous environment.

Summary

This document describes the OBJECT tag and EMBED tag styles used by Java Plug-in. It focuses on the conversion from an APPLET tag to the OBJECT and EMBED tags. Currently, HTML 4.0 suggests that the OBJECT tag is the best way to insert Java applets and JavaBeans components into a HTML page. This document will be updated in the near future should there be a need to convert the OBJECT tag to the Java Plug-in tag style. Information disclosed in this document is intended to assist ISVs for writing HTML migration tools and to assist webmasters with Java Plug-in migration. The tag style described in this document is subject to change in the future.

Note that the use of Java Plug-in is not limited to the tag styles described in this document. In fact, webmasters are encouraged to modify the tag style or mix the tag with JavaScript to fit their needs. As long as the described OBJECT tag is used in Internet Explorer and EMBED tag is used in Netscape Navigator browsers, there should be no problems running Java Plug-in. Currently, there are several conversion templates shipped with the Java Plug-in HTML converter. Webmasters may find one template better than others for their needs, and are encouraged to modify these templates themselves if necessary.

CHAPTER 4

Using the HTML Converter to Convert Applet Tags for Java Plug-in

This section includes the following topics:

Note –

1. Backup all files before converting them with this tool.
2. Cancelling a conversion will not rollback the changes.
3. Comments within the `APPLET` tag are ignored.

Introduction

The Java Plug-in HTML Converter is a utility for converting an HTML page (file) containing applets to a format for Java Plug-in. The conversion process is as follows:

First, HTML that is not part of an applet is transferred from the source file to a temporary file. Then, when an `<APPLET>` tag is reached, the converter parses the applet to the first `</APPLET>` tag (not contained in quotes) and merges the applet data with the template. If this completes without error, the original HTML file is moved to the backup folder and the temporary file is then renamed as the original file's name.

The converter effectively converts the files in place. Thus, once the converter runs, files are setup for Java Plug-in.

Running the GUI version of the HTML Converter

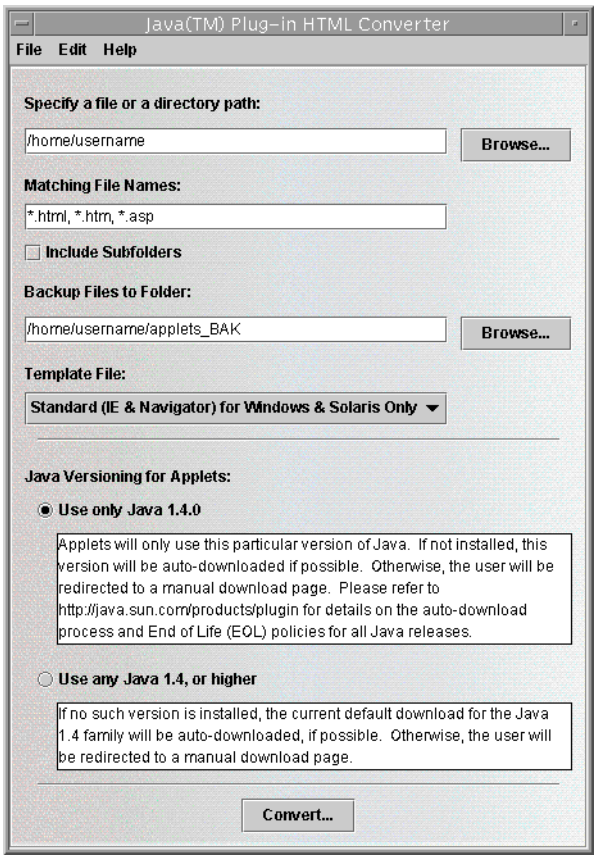
Running the GUI version of the HTML Converter

The HTML converter is located in the `lib/htmlconverter.jar` file inside the Java 2 SDK, Standard Edition, v1.4 installation. In the default package installation, this will be at `/usr/j2se/lib/htmlconverter.jar`. To launch the GUI version of the HTML Converter, do the following.

At the command line, cd to <*HTML Converter Directory*>, then type

```
java -jar htmlconverter.jar -gui
```

The HTML Converter window will appear:



Choosing files within folders to convert:

To convert all files within a folder, you may type in the path to the folder, or choose the browse button to select a folder from a dialog. Once you have chosen a path, you may supply any number of file specifiers in "Matching File Names". Each specifier must be separated by a comma. You may use * as a wildcard. Finally, if you would like all files in nested folders matching the file name to be converted, select the checkbox "Include Subfolders".

Choosing a backup folder:

The default backup folder path is the source path with an "`_BAK`" appended to the name; e.g., if the source path is `/home/user1/html`, then the backup path would be `/home/user1/html_BAK`. The backup path may be changed by typing a path in the field labeled "Backup Files to Folder:", or by browsing for a folder.

Generating a log file:

If you would like a log file to be generated, go to the Advanced Options screen (Edit->Options) and check "Generate Log File". Enter a path in the text field or browse to choose a folder. The log file contains basic information related to the converting process.

Choosing a conversion template:

A default template will be used if none is chosen. This template will produce converted html files that will work with Internet Explorer and Netscape browsers. If you would like to use a different template, you may choose it from the menu on the main screen. If you choose other, you will be allowed to choose a file to be used as the template. If you choose a file, be sure that it is a template.

Choosing a version scheme:

The *specified version* mentioned below is the version of the JRE you use to launch the converter; e.g., 1.4.0. The first two numbers in the version indicate the family; e.g., 1.4.3_02 is in the 1.4 family. For an explanation of product version numbers, see the note in the section called "Using OBJECT, EMBED and APPLET Tags in Java Plug-in."

There are two choices here:

1. You can choose to have conversion tags that will require the specified version of Java Plug-in to run your applets. If the specified version is not installed, then the client will be asked if he/she wants to install and download it.
2. 2.You can choose conversion tags that will allow any installed version of Java Plug-in in the family, equal to or higher than the specified version, to run your applets. If there is no installed version of Java Plug-in, or no installed version in the family equal to or higher than the specified version, then the client will be asked if he/she wants to install and run the latest version of Java Plug-in in the family.

Converting:

Click the "Convert..." button to begin the conversion process. A dialog will show the files being processed, the number of files processed, the number of applets found, and number of errors.

Exit or Convert More Files:

When the conversion is complete, the button in the process dialog will change from "Cancel" to "Done". You may choose "Done" to close the dialog. You can then exit the Java Plug-in HTML Converter program, or select another set of files to convert.

Details about templates:

The template file is the basis behind converting applets. It is simply a text file containing tags that represent parts of the original applet. By adding/removing/moving the tags in a template file, you can alter the output of the converted file.

Supported Tags:

$OriginalApplet$	This tag is substituted with the complete text of the original applet.
$AppletAttributes$	This tag is substituted with all of the applets attributes (code, codebase, width, height, etc.).
$ObjectAttributes$	This tag is substituted with all the attributes required by the object tag.
$EmbedAttributes$	This tag is substituted with all the attributes required by the embed tag.
$AppletParams$	This tag is substituted with all the applet's <param ...> tags.
$ObjectParams$	This tag is substituted with all the <param ...> tags required by the object tag.
$EmbedParams$	This tag is substituted with all the <param ...> tags required by the embed tag in the form name=value
$AlternateHTML$	This tag is substituted with the text in the no-support-for-applets area of the original applet
$CabFileLocation$	This is the URL of the cab file that should be used in each template that targets IE.
$NSFileLocation$	This is the URL of the Netscape plugin to be used in each template that targets Netscape.
$SmartUpdate$	This is the URL of the Netscape SmartUpdate to be used in each template that targets Netscape Navigator 4.0 or later.
$MimeType$	This is the MIME type of the Java object.

Below are four templates that come with the HTML Converter. You can make up others and put them in the `template` folder to use them.

default.tpl — the default template for the converter. The converted page can be used in Internet Explorer and Netscape Navigator browsers on Microsoft Windows platforms to invoke Java Plug-in. This template can also be used with the Netscape browser on the Solaris operating environment.

```
<!-- HTML CONVERTER -->
<OBJECT classid="clsid:E19F9330-3110-11d4-991C-005004D3B3DB"
$ObjectAttributes$ codebase="$CabFileLocation$">
$ObjectParams$
<PARAM NAME="type" VALUE="$MimeType$">
```

```
<PARAM NAME="scriptable" VALUE="false">
$AppletParams$
<COMMENT>
<EMBED type="$MimeType$" $EmbedAttributes$
$EmbedParams$ scriptable=false pluginspage="$NSFileLocation$"><NOEMBED>
</COMMENT>
$AlternateHTML$
</NOEMBED></EMBED>
</OBJECT>

<!--
$ORIGINALAPPLET$
-->
```

ieonly.tpl - the converted page can be used to invoke Java Plug-in in Internet Explorer on Microsoft Windows only.

```
<!-- HTML CONVERTER -->
<OBJECT classid="clsid:E19F9330-3110-11d4-991C-005004D3B3DB"
$ObjectAttributes$ codebase="$CabFileLocation$">
$ObjectParams$
<PARAM NAME="type" VALUE="$MimeType$">
<PARAM NAME="scriptable" VALUE="false">
$AppletParams$
$AlternateHTML$
</OBJECT>

<!--
$ORIGINALAPPLET$
-->
```

nsonly.tpl - the converted page can be used to invoke Java Plug-in in Netscape Navigator browsers on Microsoft Windows platforms and the Solaris operating environment.

```
<!-- HTML CONVERTER -->
<EMBED type="$MimeType$" $EmbedAttributes$
$EmbedParams$ scriptable=false pluginspage="$NSFileLocation$"><NOEMBED>
$AlternateHTML$
</NOEMBED></EMBED>

<!--
$ORIGINALAPPLET$
-->
```

extend.tpl - the converted page can be used in any browser and any platform. If the browser is Internet Explorer or a Netscape Navigator browser on Microsoft Windows, or a Netscape Navigator browser on the Solaris operating environment, Java™ Plug-in will be invoked. Otherwise, the browser's default virtual machine is used.

```
<!-- HTML CONVERTER -->
<SCRIPT LANGUAGE="JavaScript"><!--
var _info = navigator.userAgent; var _ns = false; var _ns6 = false;
var _ie = (_info.indexOf("MSIE") > 0 && _info.indexOf("Win") > 0 &&
_info.indexOf("Windows 3.1") < 0);
```

```
//--></SCRIPT>
<COMMENT><SCRIPT LANGUAGE="JavaScript1.1"><!--
var _ns = (navigator.appName.indexOf("Netscape") >= 0 &&
((_info.indexOf("Win") > 0 && _info.indexOf("Win16") < 0 &&
java.lang.System.getProperty("os.version").indexOf("3.5") < 0) ||
_info.indexOf("Sun") > 0));
var _ns6 = ((_ns == true) && (_info.indexOf("Mozilla/5") >= 0));
//--></SCRIPT></COMMENT>

<SCRIPT LANGUAGE="JavaScript"><!--
if (_ie == true) document.writeln('<OBJECT
classid="clsid:E19F9330-3110-11d4-991C-005004D3B3DB" $ObjectAttributes$
codebase="$CabFileLocation$"><NOEMBED><XMP>');
else if (_ns == true && _ns6 == false) document.writeln('<EMBED
type="$MimeType$" $EmbedAttributes$
$EmbedParams$ scriptable=false
pluginspage="$NSFileLocation$"><NOEMBED><XMP>');
//--></SCRIPT>
<APPLET $AppletAttributes$></XMP>
$ObjectParams$
<PARAM NAME="type" VALUE="$MimeType$">
<PARAM NAME="scriptable" VALUE="false">
$AppletParams$
$AlternateHTML$
</APPLET>
</NOEMBED></EMBED></OBJECT>

<!--
$ORIGINALAPPLET$
-->
```

Running the converter from the command line:

Running the converter from the command line:

Format:

```
java -jar htmlconverter.jar [filespecs]  [-simulate] [-options1 value1
[-option2 value2 [...] ]
```

If only "**`java -jar htmlconverter.jar`**" is specified (no filespecs or options), the GUI version of the converter will be launched. Otherwise, the GUI will be suppressed.

`[filespecs]`: space-delimited list of file specifications which may include wildcard (*), e.g. **.html*, *file*.html*).

`[simulate]`: Set to preview a conversion without actually doing the conversion. Use this option if you are unsuer about a conversion. You will be shown detail information about the conversion had it been done.

Options:	Descriptions
source:	Path to files; e.g., `c:\htmldocs` in Windows, `/home/user1/htmldocs` in Unix. Default is *<userdir>* If the path is relative, it is assumed to be relative to the directory from which the HTMLConverter was launched.
dest	Path to converter file location. Default: *<usrdir>*
backup	Path to the directory where you want backup files to be stored. Default: *<userdir>/<source>_bak* If the path is relative, it is assumed to be relative to the directory from which the HTMLConverter was launched.
f	Force overwritting of backup files.
subdirs:	Sets whether files subdirectories should be processed or not. Default: *false*
template	Name of template file to use. Default: Standard (IE & Navigator) for Windows & Solaris Only. **Note –** Use the default if you are unsure.
log	Path and filename for the log. Default: *<userdir>/convert.log*
progress	Set to display standard out progress during conversion. Default: *true*
latest	Use the latest JRE supporting the MIME type.
gui	Display the graphical user interface for the converter.

CHAPTER 5

Proxy Configuration

This chapter includes the following topics:

Introduction

For enterprise customers it is important to be able to set up secure computing environments within their companies, and proxy configuration is an essential part of doing that. Proxy configuration acts as a security barrier; it ensures that the proxy server monitors all traffic between the Internet and the intranet. This is normally an integral part of security enforcement in corporate firewalls within intranets. Enterprise customers who want to use Java Plug-in to deploy applets on their intranet web pages may also set up proxy support. This support is required for Java Plug-in to work in an intranet environment and can be set up through the Java Plug-in Control Panel.

The Control Panel provides three proxy options:

- "Use browser settings"
- Manual configuration through the Protocol-Address-Port table
- "Automatic proxy configuration URL"

If "Use browser settings" is selected, then proxy information is entered entirely through the browser. For Internet Explorer, go to Tools>Internet Options ... and select the Connections tab and then LAN Settings ... ; for Netscape, go to Edit>Preferences ... and select Advanced under Category and then Proxies. How this works and the three types of connections that can be set up through the browser-Direct, Manual, and Automatic-are described in the following sections.

If you select manual configuration in the Control Panel, then you must enter in the table for each protocol the address and port for the proxy server. Note that you may select to exclude some hosts from requiring proxy servers by listing them in the field labeled "No proxy host".

If you select "Automatic proxy configuration URL", then you must enter the URL for the location of the JavaScript called `FindProxyForURL(URL url)` that returns the proxy server to be used for a URL. Support for this script is the same as described below under Automatic Proxy Configuration.

How Java Plug-in Obtains Proxy Information

Because browsers on different platforms store proxy information differently, there is no generic mechanism to obtain proxy information. Here's how Java Plug-in obtains proxy information for three different browser-platform combinations:

Internet Explorer on Win32: Internet Explorer stores proxy information in the same set of keys in the windows registry. Java Plug-in obtains this information directly.

Netscape Navigator browser on Win32: Navigator 4 stores proxy information in the user preference file on the local machine. Java Plug-in reads and parses the user preference file to obtain the Navigator 4 proxy information. Netscape 6 has an API for obtaining proxy information. `findProxyForURL(URL)` returns proxy configuration information for the URL passed to it.

Netscape Navigator browser on the Solaris operating environment and Linux: Navigator stores proxy information in a file in the local machine. Java Plug-in reads and parses this file to obtain the proxy information. For Netscape 6 the process is the same as described in the previous section.

Java Plug-in obtains proxy information at startup time. If you change the proxy setting after Java Plug-in has started, you may force Java Plug-in to reload the proxy information from the browser through the `p` option in the Java Console.

Direct Connection

Direct connection does not use a proxy. For certain situations, such as when mobile users connect to the company through a modem, direct connection to the intranet environment is required, and proxies should not be used in these cases.

Manual Proxy Configuration

Both Internet Explorer and Netscape Navigator support manual proxy configuration. Users can specify the proxy server and port for each protocol. Users can also specify one proxy server and port for all protocols. To minimize the workload of the proxy server, some sites might bypass the proxy server completely when a machine is connecting to another machine inside the intranet environment. To do this, network administrators and users can specify the proxy server bypass list in the browser's setting.

Internet Explorer: Java Plug-in recognizes and supports the proxy server and port setting associated with the protocol. Internet Explorer supports various syntaxes in the proxy server bypass list, as follows:

- IP address/hostname only
- IP address/hostname with wildcard
- IP address/hostname with protocol

For example, if you specify "**`121.141.23.5;*.eng;http://*.com`**" in the proxy server bypass list, then the browser bypasses the proxy whenever one of the following occurs:

- "**`121.141.23.5`**" is requested
- the URL hostname ends with "**`.eng`**"
- the URL protocol is `http` and the URL hostname ends with "**`.com`**"

Currently, Java Plug-in supports the first two syntaxes in the proxy server bypass list in Internet Explorer. Internet Explorer also supports bypassing the proxy server for local (intranet) addresses without using the proxy server bypass list. Java Plug-in supports this option by bypassing the proxy server if the URL hostname is plain; i.e., the hostname contains no dot (.).

*Netscape Navigator:*Java Plug-in recognizes and supports the proxy server and port setting associated with the protocol. For example, if you specify "**`.eng,.sun.com`**" in the proxy server bypass list in the Netscape Navigator browser, it bypasses the proxy whenever the URL hostname ends with "**`.eng`**" or "**`.sun.com`**". Java Plug-in fully supports this syntax in the proxy server bypass list in Navigator.

For more information about manual proxy configuration in your browser, please consult the user guide for your browser.

Automatic Proxy Configuration

Both Internet Explorer and Netscape Navigator support automatic proxy configuration. The browser's automatic proxy configuration is set to a particular URL that contains a JavaScript file with `.pac` or `.js` extension. This file contains a function called `FindProxyForURL()` that contains the logic to determine which proxy server to use when the browser receives a connection request. This function is written by the system administrator for the particular intranet environment. When the browser starts up, it recognizes the URL of the JavaScript file and downloads the file to the local machine using direct connection. Then whenever it needs to make a new connection, the browser executes the JavaScript function `FindProxyForURL()` in the file to obtain the proxy information to set up the connection.

Internet Explorer: During startup, Java Plug-in downloads the JavaScript file to the local machine using direct connection. Then whenever it needs to make a new connection, it executes the FindProxyForURL function to obtain the proxy information using the JavaScript engine in Internet Explorer.

*Netscape Navigator browser:*During startup, Java Plug-in downloads the JavaScript file to the local machine using direct connection. Then whenever it needs to make a new connection, it executes the `FindProxyForURL()` function to obtain the proxy information by using the JavaScript engine in the Netscape Navigator browser.

There are a number of predefined JavaScript functions which can be called from the JavaScript function `FindProxyForURL()`. Java Plug-in provides its own implementation of these functions to completely emulate the automatic proxy configuration. Here are a few notes regarding this implementation:

- Function `dnsResolve()` always returns an empty string if the host is not an IP address
- Function `isResolvable()` always returns false if the host is not an IP address.
- Function `isInNet()` always returns false if the host is not an IP address.

Note that executing the function `FindProxyForURL()` always returns proxy information as a string. Java Plug-in extracts the setting in the following way:

- If "DIRECT" is in the string, Java Plug-in assumes direct connection.
- If "PROXY" is in the string, it uses the first proxy setting for the connection.
- If "SOCKS" is in the string, it uses the SOCKS v4 for the connection.
- Otherwise, the proxy information in the string is incorrect. In this cases, Java Plug-in assumes direct connection.

For more information about automatic proxy configuration in your browser, consult the user guide for your browser.

CHAPTER 6

Protocol Support

HTTP, FTP and Gopher

Java Plug-in supports HTTP, FTP, and GOPHER protocols, including built-in proxy configuration support.

HTTPS

Introduction

Prior to version 1.4 of the Java 2 Platform, Standard Edition, Java Plug-in supported HTTPS through browser-dependent native APIs. Java™ Secure Socket Extension (JSEE) is a new Java extension in 1.4, providing a Java implementation of SSL and HTTPS for the Java platform. Java Plug-in in 1.4 leverages JSSE to provide HTTPS support, instead of relying on the browser.

This provides the following advantages over using browser-dependent native APIs for support:

- No native code is used, eliminating separate HTTPS support for each browser on each platform and making code more maintainable and portable.
- Implementation of `java.net.HttpsURLConnection` is provided in JSSE, allowing developers to take advantages of all features in HTTPS, including tunneling.
- Support is multi-threaded. Because the implementation is in Java, there is no need to have a mutex to lock up connections and Java performance is enhanced for simultaneous HTTPS connections.

Java Plug-in supports HTTPS through JSSE for Win32, Linux and the Solaris operating environment.

Proxy and Cookie Support

A different proxy configuration may be used for every HTTPS connection. Java Plug-in provides full proxy configuration support in HTTPS. Proxy configuration may be set through user preference in the browser, as well as the Java Plug-in Control Panel. Direct, manual and automatic proxy configuration are supported.

Cookies may be sent/received for every HTTPS connection. Java Plug-in provides full cookie support, automatically retrieving or updating cookies through the browser cookie store.

Error handling support

When accessing an HTTPS server, errors may occur. Java Plug-in has hooked into JSSE to provide the following types of error handling:

- Hostname mismatch: If the HTTPS server host name does not match the name on the server certificate, a warning dialog will appear.
- Untrusted server certificate: If the server certificate can not be verified during the SSL handshaking, a warning dialog will appear.
- Untrusted client certificate: In case client authentication is required by the server and the client certificate cannot be verified, a warning dialog will be appear.
- Server authentication: If the client accesses a protected directory on the HTTPS server, the users will be prompted for a username and password. Note: Only basic and digest authentication are currently supported.

Potential issues with HTTPS through JSSE

Although support of HTTPS through JSSE eliminates many browser-specific problems, there are several issues that developers should be aware of:

- Untrusted server certificate: When SSL handshaking takes place in establishing an HTTPS connection, the server certificate is verified against the root CA store in J2SE. However, J2SE supports fewer root CA certificates than does the browser. As a result, you may have problems with untrusted server certificates.
- Client authentication: An HTTPS server may require client authentication, in which case a local client certificate is sent to the server for authentication. In JSSE, client certificates are stored in a separate file and are independent of the browser. In order for client authentication to work, developers must import client certificates into JSSE through the keytool. For more information, see the JSSE documentation online at http://java.sun.com/j2se/1.4/docs/guide/security/jsse/JSSERefGuide.html.
- Level of error handling: Java Plug-in currently handles the types of error listed in the previous section. However, if there are additional types of error that Java Plug-in doesn't recognize, the Java applet code may break.

- Startup delay: When HTTPS is used, a secure random generator will be created. This process may take several seconds to a minute, depending on the speed of the client machine. In some cases, Java Plug-in may appear to be hung during the creation of the secure random generator. This issue has been addressed by delaying the loading of HTTPS code in Java Plug-in as well as leveraging the native OS secure-seed generator if available on the platform; thus, Java Plug-in startup time should not be affected if HTTPS is not used. However, some users may still see the startup delay, depending on the exact loading sequence of the startup code in Java Plug-in.

Socks

Java Plug-in currently supports SOCKS version 4.

Note – For HTTP/HTTPS, a SOCKS proxy server may be used with a web proxy server to add caching. The behavior, however, may differ from that observed when running a similar configuration in a browser without Java Plug-in.

CHAPTER 7

Cookie Support

This section covers the following topics:

Introduction

Cookies are a way of storing data on the client side. They have been used extensively for personalization of portal sites, user preference tracking, and logging into web sites. For enterprise customers using cookies in their web sites, cookie support in Java Plug-in is essential for making deployment of Java applets or JavaBeans™ components easier.

Cookie support allows an applet or JavaBeans component to pass a cookie back to a web server if that cookie originated from the web server. This provides the server with information about the state of the client. Beginning with version 1.4, Java Plug-in provides bidirectional cookie support. This document specifies how cookie works in different browser environments.

How Java Plug-in Supports Cookies

Java Plug-in supports both Internet Explorer and Netscape Navigator browsers on various Win32 platforms and the Solaris operating environment, and it supports Netscape Navigator browsers on Linux platforms. Java Plug-in provides cookie

support through the browser API. Because browsers on various platforms implement the browser's API differently, cookie support in Java Plug-in varies according to platform. You need to know how each browser supports cookies and how Java Plug-in accesses and updates cookie information.

When a browser makes an HTTP/HTTPS request through a URL connection, it normally checks the cookie cache and policy to determine if a cookie should be sent along with the HTTP/HTTPS request header. If so, the browser will read the cookie from the cache and append the cookie as part of the HTTP/HTTPS request header.

When a browser processes the HTTP/HTTPS respond header through a URL connection, it will check the header to see if any cookies should be set. The browser also checks the cookie policy to determine if the action is allowed. If so, it will extract the cookie from the HTTP/HTTPS respond header and write it into the cookie cache.

When an HTTP/HTTPS request is made using Java Plug-in, Java Plug-in consults the browser to determine if a cookie should be sent along. If so, the HTTP/HTTPS request will contain the cookie as part of the header. Otherwise, the HTTP/HTTPS request will be sent with no cookie attached.

When a cookie needs to be set from the HTTP/HTTPS respond header, Java Plug-in uses the browser API to do so, with the exception of Netscape Navigator 4 browsers. For the Netscape Navigator 4 browser, there is no API allowing Java Plug-in to do so.

There is another limitation for Netscape Navigator 4 browsers. When using Java Plug-in in a Netscape Navigator 4 browser, cookie support works only if the codebase is the same or a subdirectory of the document base. See examples in the table below:

Document Base	Codebase	Will It Work?
http://host.com/my/	http://host.com/my/	Yes
http://host.com/my/	http://host.com/my/page	Yes
http://host.com/my/page	http://host.com/my/	No

Currently, cookie support in Java Plug-in is triggered automatically when an HTTP/HTTPS connection needs to be made.

To ensure that cookie support in Java Plug-in always works as expected, the following is recommended:

- Use Internet Explorer or Netscape 6 browser
- In the case that you must use the Netscape 4 browser, avoid having your web server set cookies in an HTTP/HTTPS connection with an applet.

(The above recommendations apply to an intranet environment, where deployment of browsers and web servers is controllable.)

For more general information about how cookies work, consult the user guide for your browser.

Cookie Policy Support in Java Plug-in

Java Plug-in supports all cookie policies that are supported in both Internet Explorer and Netscape Navigator browsers. Cookie policy can be configured in both browsers (see your browser guide for details). There are various options, including the following:

- Always accept cookies
- Disable all cookie use
- Prompt/Warn before accepting a cookie
- Only accept cookies from a server if they get sent back to that server

When cookie policy is changed in the browser, it will take effect the next time an HTTP/HTTPS connection is made via Java Plug-in.

Java Plug-in does not provide cookie-caching support. Instead, it consults the browser every time an HTTP/HTTPS connection is made. Thus the browser is the only place where cookies are stored. Any change to a cookie in the browser is reflected immediately in Java Plug-in when a new HTTP/HTTPS connection is made.

Cookies and Disk Caching

Java Plug-in provides disk-caching support through the browser's API. It is triggered whenever a `.jar` or `.class` file is downloaded via an HTTP/HTTPS connection. When disk-caching support is triggered, the file is downloaded entirely by the browser, and a cookie will be handled automatically by the browser.

Cookies and Security

Although cookies are sent when an HTTP/HTTPS connection is made in Java Plug-in, applets and beans have no access to this information-even if the code is trusted. Moreover, a cookie is only sent back to the host and domain from which it came.

CHAPTER 8

Applet Caching and Installation in Java Plug-in

This section covers the following topics:

Note – The discussion below applies to JavaBeans components as well as applets.

Caching Option

The version 1.4 release introduces a new form of applet caching, allowing an applet deployer to decide if an applet should be "sticky", i.e., placed in a secondary disk cache which the browser cannot overwrite. The only time a sticky applet gets downloaded after caching is when it is updated on the server. Otherwise the applet is always available for fast loading. Applets providing core business applications should be made sticky to improve startup performance.

Once an applet is cached, it no longer needs to be downloaded when referenced again. Thus performance is improved.

This new feature is activated by including the new *cache_archive*, *cache_version* and *cache_archive_ex* values in the `OBJECT/EMBED` tag that specifies the use of Java Plug-in as shown below:

cache_archive

The *cache_archive* attribute contains a list of the files to be cached:

```
<PARAM NAME="cache_archive" VALUE="a.jar,b.jar,c.jar">
```

Like the archive attribute in the APPLET tag, the list of jar files in the *cache_archive* attribute do not contain the full URL, but are always downloaded from the codebase specified in the EMBED/OBJECT tag.

Note that the list of .jar files in the *cache_archive* attribute and those in the archive attribute may look similar, but they should not contain the same .jar files.

cache_version

The *cache_version* is an optional attribute. If used, it contains a list of file versions to be cached:

```
<PARAM NAME="cache_version" VALUE="1.2.0.1, 2.1.1.2, 1.1.2.7">
```

Each version number is in the form *X.X.X.X*, where *X* is hexadecimal. Each version number corresponds to a respective .jar file in the *cache_archive*.

cache_archive_ex

In order to allow pre-loading of jar files, HTML parameter *cache_archive_ex* can be used, This parameter allows you to specify whether the jar file needs to be pre-loaded; optionally the version of the jar file can also be specified. The value of *cache_archive_ex* has the following format:

```
cache_archive_ex =
"<jar_file_name>;<preload(optional)>;<jar_file_version>,<jar_file_name>;
<preload(optional>;<jar_file_version(optional>,...."
```

The optional tags like *preload* or the *jar_file_version* can appear after the *jar_file_name* in any order seperated by the delimiter ";". Following shows how these tags might be used in an HTML page:

```
<OBJECT .... >
<PARAM NAME="archive" VALUE="a.jar">
<PARAM NAME="chache_archive" VALUE="b.jar, c.jar, d.jar">
<PARAM NAME="chache_version" VALUE="0.0.0.1, 0.0.A.B, 0.A.B.C">
<PARAM NAME="cache_archive_ex" VALUE="applet.jar;preload,
util.jar;preload;0.9.0.abc, tools.jar;0.9.8.7">
```

In the above example, a.jar is specified in archive, whereas b.jar, c.jar and d.jar are specified in cache_archive. The versions are also specified for b.jar, c.jar, and d.jar as 0.0.0.1, 0.0.A.B, and 0.A.B.C, respectively. In *cache_archive_ex*, applet.jar is specified to be pre-loaded. util.jar is also specified to be pre-loaded but along with the version. For tools.jar, only version is specified.

Java Plug-In doesn't compare the versions if they are not specified for all the jar files specified in HTML parameter *cache_archive*. If *cache_archive* is used without *cache_version*, the jar files specified in *cache_archive* are treated no differently than the jar files specified in HTML parameter archive. Similar treatment is given to jar files specified in *cache_archive_ex* when preload and version options are not provided.

Class files and resources will be searched in the following order from the .jar files specified by the HTML parameters

1. cache_archive_ex
2. cache_archive
3. archive

Cache Update Algorithm

By default, without the *cache_version* attribute, applet caching will be updated if:

- The *cache_archive* has not been cached before, or
- The "Last-Modified" value of the *cache_archive* on the web server is newer than the one stored locally in the applet cache, or
- The "Content-Length" of the *cache_archive* on the web server is different than the one stored locally in the applet cache.

However, in some situations, the "Last-Modified" value returned from the web server through HTTP/HTTPS may not reflect the actual version of the applets. For example, if the web server crashes and all the files are restored, the *cache_archive* will have a different modification date on the server. Even if the *cache_archive* has not been updated, it will still force all the Java Plug-in clients to download the *cache_archive* again.

To strongly enforce the version update, it is recommended that the applet deployer use the *cache_version* attribute.

If *cache_version* is used, applet caching will be updated if the *cache_version* for the *cache_archive* in the `EMBED/OBJECT` tag is larger than the one stored locally in the applet cache. Note that the version number is used for triggering an update; there is no actual version number attached to the `.jar` file on the web server. In fact, unless version is used to trigger an update, it is possible the applet on the web server could be updated without the applet in *cache_archive*.

Using *cache_version* eliminates the need to connect to the web server to obtain "Last-Modified" and "Content-Length" of the *cache_archive*. In most cases this will speed up performance.

Security

Although sticky applets are cached locally, they will still conform to the security policy defined by their original codebase and signer.

Known Limitations

- Caching of the JAR files specified in the manifest's *Class-Path* variable using Java Plug-in's cache is currently not supported.
- The path specified in the *cache_archive* must be a relative URL to the applet's codebase. Full URLs are not supported in *cache_archive*.

CHAPTER 9

Using the Java Plug-in Control Panel to Set Plug-in Behavior/Options

This section covers the following topics:

Overview

The Java™ Plug-in Control Panel enables you to change the default settings used by the Java Plug-in at startup. All applets running inside a running instance of Java Plug-in use these settings.

Starting the Java Plug-in Control Panel

You can run the Control Panel by launching the ControlPanel executable file. In the Java 2 SDK, this file is located at

```
<SDK installation directory>/jre/bin/ControlPanel
```

For example if your Java 2 SDK is installed at `/usr/j2se`, launch the Control Panel with this command:

```
/usr/j2se/jre/bin/ControlPanel
```

In a Java 2 Runtime Environment installation, the file is located at

```
<JRE installation directory>/bin/ControlPanel
```

You can also use Netscape to visit the Control Panel applet page, which was installed inside the JRE directory as the file ControlPanel.html. In the Java 2 SDK this file is located at:

```
<SDK installation directory>/jre/ControlPanel.html
```

In the JRE:

```
<JRE installation directory>/ControlPanel.html
```

Saving Options

When you have completed your changes to the Control Panel options, click Apply to save the changes. Click Reset to cancel your changes and reload the Control Panel values from the configuration file.

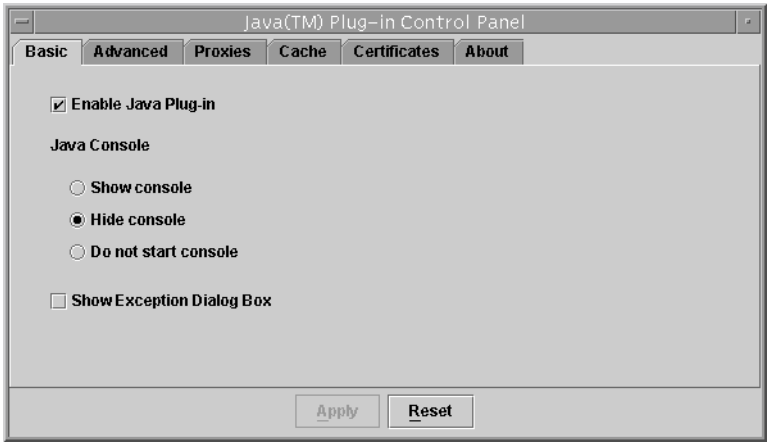
Setting Control Panel Options

There are five panels from which you can set various options within the Java Plug-in Control Panel. These panels are labeled:

- Basic
- Advanced
- Proxies
- Cache
- Certificates

Basic Panel

The basic panel looks like this:

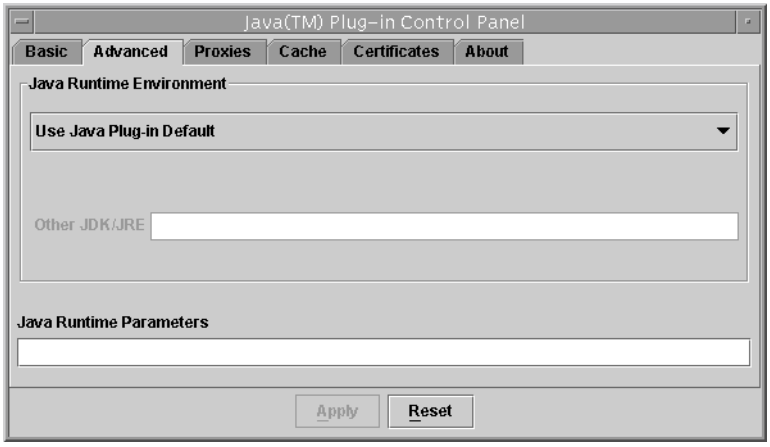


Use the Basic panel to set the following options:

- *Enable Java Plug-in:* Enables Java Plug-in to run applets or JavaBeans components. If this option is left unchecked, Java Plug-in does not run any applets. The default setting is with the Plug-in enabled(checked).
- *Show Console:* Displays the Java Console while running applets. The console displays messages printed by `System.out` and `System.err` objects. It is useful for debugging problems. The default setting is not to show the Java Console (unchecked).
- *Hide console:* The console is running but hidden.
- *Do not start console:* The console is not started.
- *Show Exception Dialog Box :* Shows an Exception Dialog Box when exceptions occur.

Advanced Panel

The Advanced panel looks like this:



Use the Advanced panel to set the following options:

- *Java Runtime Environment:* Enables Java Plug-in to run with any JRE or Java 2 SDK, Standard Edition v 1.3 or 1.4 installed on your machine. Java Plug-in 1.3/1.4 is delivered with a default JRE. However, you can override the default JRE and use an older or newer version. The Control Panel automatically detects all versions of the Java 2 SDK or JRE installed on the machine. In a list box it displays all installed Java 2 SDK or JRE versions which you can use. The first item in the list will always be the Java Plug-in default, and the last item will always say Other. If you choose Other, you must specify the path to the JRE or Java 2 SDK, Standard Edition v 1.3/1.4. Only advanced users should change this option.
- *Java Run Time Parameters:* Overrides the Java Plug-in default startup parameters by specifying custom options. The syntax is the same as used with parameters to the `java` command-line invocation.

  To enable assertion support, the following system property must be specified in the Java Runtime Parameters:

  ```
  -D[ enableassertions | ea ][: <package name>"..." | : <class name>]
  ```

  To disable assertion in the Java Plug-in, specify the following in the Java Runtime Parameters:

  ```
  -D[ disableassertions | da ][:<package name>"..." | : <class name>]
  ```

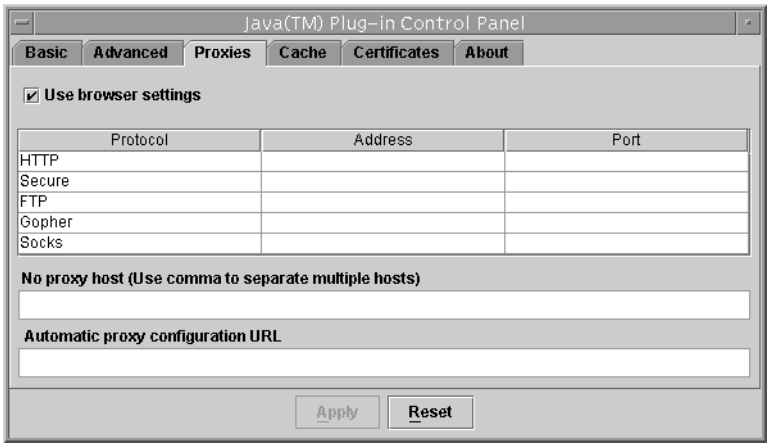
  Assertion is disabled in Java Plug-in code by default. Since the effect of assertion is determined during Java Plug-in startup, changing assertion setting in Java Plug-in Control Panel will require a browser restart in order for the new settings to take effect.

  Because Java code in Java Plug-in also has built-in assertion, it is possible to enable the assertion in Java Plug-in code through

```
-D[ enableassertions | ea ]:sun.plugin
```

Proxies Panel

The Proxies panel looks like this:

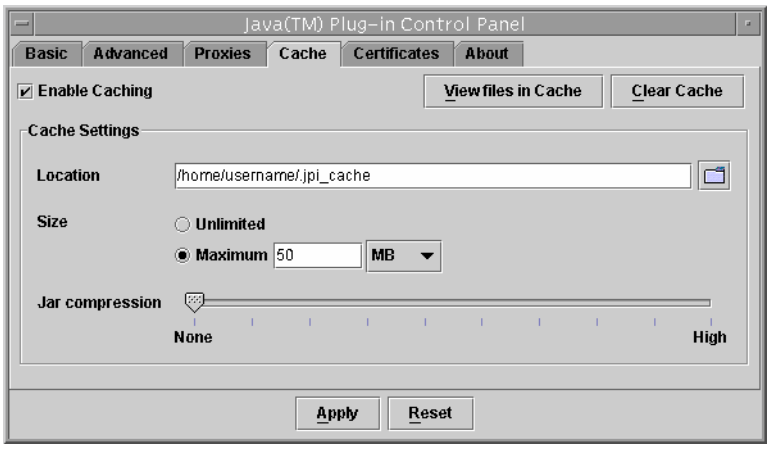


Use the Proxies panel to use the browser default settings or to override the proxy address and port for the different protocols.

- *Use browser settings:* Uses the browser default proxy settings.
- *Proxy information table:* You can override the default settings by unchecking the browser settings check box, then completing the proxy information table beneath the check box. You can enter the proxy address and port for each of the supported protocols: HTTP, Secure (HTTPS), FTP, Gopher, and Socks.
- *No proxy host:* This is a host or list of hosts for which no proxy/proxies are to be used. No proxy host is usually used for an internal host in an intranet environment.
- *Automatic proxy configuration URL:* This is the URL for the JavaScript file (.js or .pac extension) that contains the `FindProxyForURL()` function. `FindProxyForURL()` has the logic to determine the proxy server to use for a connection request.

Cache Panel

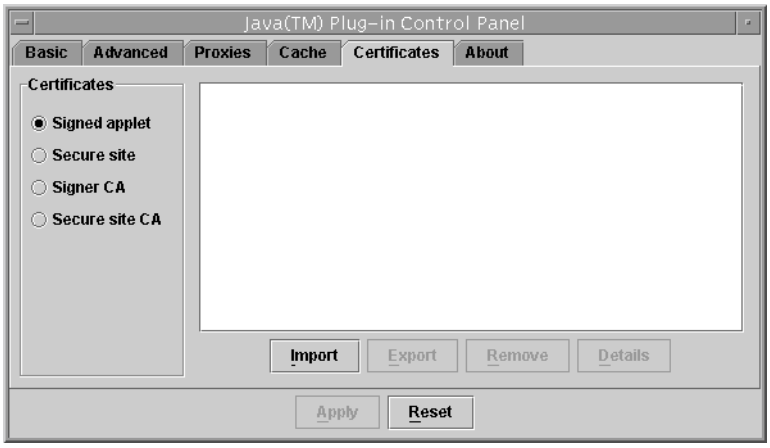
The Cache panel looks like this:



- *Enable JAR Caching:* Check this to enable JAR caching.
- *View JAR Cache:* Press this to view the JAR cache.
- *Clear JAR Cache:* Press this to clear the JAR cache used for storing applet JAR files.
- *Location:* You can use this to specify the location of the JAR cache.
- *Size:* You can check Unlimited to make the JAR cache unlimited in size; or you can set the Maximum size of the JAR cache.
- *Compression:* You can set the compression of the JAR cache files between None and High. While you will save memory by specifying higher compression, performance will be degraded; best performance will be achieved with no compression.

Certificates Panel

The Certificates panel looks like this:



Four types of certificates may be selected:

- *Signed applet:* These are certificates for signed applets.
- *Secure site:* These are certificates for secure sites.
- *Signer CA:* These are certificates of Certificate Authorities (CAs) for signed applets; they are the ones who issue the certificates to the signers of signed applets.
- *Secure site CA:* These are certificates of Certificate Authorities (CAs) for secure sites; they are the ones who issue the certificates for secure sites. For Signed applet and Secure site certificates, there are four options: Import, Export, Remove, and Detail. The user can import, export, remove and view the details of a certificate. For Signer CA and Secure site CA, there is only one option: Detail. The user can only view the details of a certificate.

CHAPTER 10

Netscape 6

This section describes the primary features of the Java runtime in Netscape 6, which is enabled by Java Plug-in.

APPLET, EMBED AND OBJECT Tag Support

Applets are loaded by Java Plug-in if these tags are used. For information on how to use the OBJECT or EMBED tag to launch an applet, see Chapter 3.

Java-JavaScript Bi-directional Communication

JavaScript can access the methods of applets, and applets can access the Document Object Model (DOM) through JavaScript. Thus an HTML author can access applet methods and applet developers can access the DOM.

See online documentation on *Java-to-JavaScript* and *JavaScript-to-Java* at http://java.sun.com/j2se/1.4/docs/guide/plugin/developer_guide/java_js.html and http://java.sun.com/j2se/1.4/docs/guide/plugin/developer_guide/js_java.html respectively. Be sure to read the sections on security.

RSA Signed Applet Verification

RSA signed applet verification is supported.

Display of Java Console

You may display the Java Console through the Netscape 6 browser menu: Tasks->Tools->Java Console.

Enable/Disable the Java Platform

You may enable/disable the Java platform through the Netscape 6 browser menu: Edit->Preferences->Advanced. Note that to take effect, the browser must be restarted.

Applet Lifecycle Change

Whenever a page is visited, the`init()` and `start()` methods of the applet are called; and whenever the page is left, the `stop()` and `destroy()` methods may be called.

Proxy and Cookie Support

Java Plug-in previously handled proxy and cookie support alone. In Netscape 6 this support is moved to the browser.

HTTPS

HTTPS is supported through the Java Secure Socket Extension (JSSE) in the Java 2 Platform, Standard Edition.

Automatic Download

Via its XPInstall mechanism, Netscape 6 will support automatic download of Java Plug-in (JRE) if it is not present.

Backward Compatibility Issue

Although backward compatibility between the Java 2 Platform and the Netscape VM has been the goal, it may not be 100%. Some applets may run as is; other may only need recompilation; others, however, may need to be ported to the Java 2 platform.

CHAPTER 11

Frequently Asked Questions (FAQ)

An FAQ is available for Java Plug-in at
http://java.sun.com/j2se/1.4/docs/guide/plugin/developer_guide/faq/.

The FAQ is comprised of several sections, including a basic FAQ for people unfamiliar with Java Plug-in, a developer FAQ with information for system administrators and developers, a troubleshooting FAQ, and an FAQ about APPLET tag support.