## Java Applets

**(last updated: Feb 6, 2007)**

Select font size: 8pt 9pt 10pt 11pt 12pt 14pt 16pt

### Client-side web programming/Applet execution

Java Applets are used for web-based *client-side* programming in contrast to *server-side* programming effected by Java Servlets and Server Pages. Client-side programming means that the HTML content delivered to the browser contains programmatic content. The most common programming language used is JavaScript (or, more correctly, ECMA script) built-in to the browser. Up to version 1.1.3, Java used to be built into browsers; however this method of delivery was abandoned, in part because of a legal dispute between SUN Microsystems and Microsoft.

The browser can be programmed by any language which can be delivered through a *plugin*. This is the manner by which modern Java Applets are executed. Plugins are widely available for many types of programming systems such as Macromedia's Flash player.

To run a Java Applet in a browser, the Java plugin must be installed. In the Windows framework, all recent lastest Java installations check for installed browsers (such as IE) and query the user whether to install the browser plugin for that version of Java. It is also possible to run Java Applets independent of the browser through the appletviewer tool; however, the default security restrictions are not as rigid for appletviewer execution as they are for browser execution.

#### The Java plugin

It's relatively automatic to install the latest version of the Java plugin in Windows systems. In Linux systems, the plugin installation may not be automatic. The standard JDK plugins are available in the jre/plugin subdirectory of the JDK distribution directory. For example, using the 6, the actual plugin might be this file:

/usr/local/jdk1.6.0/jre/plugin/i386/ns7/libjavaplugin\_oji.so

If, say, you're using Mozilla firefox, this plugin must be "installed" into the plugins subdirectory of the distribution, which may be this directory:

/usr/lib/mozilla-firefox/plugins/

In that case, I recommend changing to the mozilla plugins directory and (as root) making a *symbolic link* to the Java plugin:

# cd MOZILLA-FIREFOX-PLUGINS-DIRECTORY  
# ln -s PATH-TO-PLUGIN/libjavaplugin\_oji.so .

From the command line shell, you can invoke SampleApplet.html executed by firefox by:

$ firefox -url SampleApplet.html

### Applets

The Java applet classes are these

* java.awt.Applet the older AWT class
* javax.swing.JApplet, the Swing class which extends java.awt.Applet

We are interested exclusively in the modern JApplet. Unlike an application, a JApplet is not executed directly by the java interpreter. Instead it is loaded and executed by the java interpreter built-in to the Java plugin. In particular, a JApplet has no main function; instead, it is controlled by standard functions activated by the plugin (or appletviewer), including these, among others:

public void init(); // the JApplet is loaded  
public void start(); // the browser applet is viewed  
public void stop(); // the browser leaves the applet page  
public void destroy(); // the JApplet is unloaded

An applet is activated through HTML-specific tags, often in a HTML file called as *stub*, whose sole purpose is to cause the browser/viewer to load the applet. Here is a sample stub file:

<applet code="MyApplet"  
 width="300" height="100"  
>  
</applet>

In order to run the MyApplet class we would co-locate MyApplet.class along with MyApplet.html in the same directory and then point our browser to MyApplet.html, or, via appletviewer:

appletviewer MyApplet.html

### HtmlConverter

The applet tag is deprecated by HTML standards, and so, even though the applet tag will probably work as is, you are encouraged to convert it to the modern form using the object tag. The Java Development Kit provides a converter operation called HtmlConverter specifically for converting the applet tag usages. From a shell command line, run:

HtmlConverter MyApplet.html

A backup copy of MyApplet.html is made in a separate directory and MyApplet.html is converted to the object form. You can also simply run HtmlConverter without parameters and allow it to choose the target applet stub file.

### Applet Security Restrictions

An applet is downloaded from a server an runs in the browser and so the Java plugin in a browser enforces, by default, stark security restrictions over what the applet can do. This is clearly necessary to protect applets from inadvertently (or maliciously) accessing files and client-side services which it shouldn't do. In particular,

* an applet cannot access client-side files
* an applet cannot execute many of the System functions, such as exit.
* an applet cannot connect to sites other than the one from which it was downloaded

By default, the **only** external access permitted an applet is to make a socket connection back to the server from which the applet was downloaded. Therefore, since our MySQL database server is actually a socket connection to port 3306, we can connect to it so long as

* the web server is also the database server
* the firewall permits the browser connection to the database

One can also circumvent the security restrictions by an elaborate mechanism which involves creating a digital certificate and attaching it to a JAR file containing the applet.

### Accessing Jar files

The archive attribute can be used to access classes in jar files.

<applet  
 code="MyApplet"  
 archive="JarFile1.jar,JarFile2.jar"  
 width="300" height="100"  
>  
Applet Didn't Load  
</applet>

This is particularly useful for accessing our MySQL driver. Indeed, the class MyApplet.class itself may be part of a JAR file and we can avoid any class files with the applet. JAR files can be created using the jar utility provided with JDK.

### Run-time parameters

We cannot use our standard ResourceBundle to deliver run-time parameters since this is effectively file access prohibited by the applet security mechanism. In its place we can deliver parameters through the HTML stub file as follows:

<applet  
 code="MyApplet"  
 width="300" height="100"  
>  
<param name="param\_name1" value="param\_value1" />  
<param name="param\_name2" value="param\_value2" />  
Applet Didn't Load  
</applet>

Within the applet itself, these parameters can be read using the getParameter function as follows:

String value = getParameter("param\_name1");  
...

### Rewriting a Java GUI application as a JApplet

The rewrite of a Java GUI JFrame-based application to a JApplet is fairly simple. The major difficulty is dealing with the security restrictions. Here are the major rewrite changes that need to be done.

1. Replace JFrame by JApplet
2. Eliminate the main function (it doesn't hurt to keep it in, it's just not used)
3. Replace the constructor by public void init()
4. Elminate any calls to super for base-class initialization
5. Remove statements which control the JFrame GUI such as:  
   setDefaultCloseOperation, setSize, setVisible
6. Eliminate any statements which are prohibited by security restrictions, such as System.exit(0)
7. Place any jar files used within the directory holding the stub file and access them in the archive attribute as a comma-separated list. Actually it doesn't have to be *right there*, it's just that the web server may not permit access to the jar file just anywhere in the system.
8. Modify the code so that any runtime parameters are delivered by the getParameter function using parameters defined within the applet's HTML stub file.