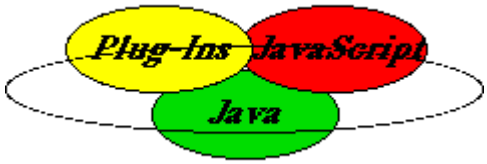

LIVECONNECT

Phillip Denis
Anju Tai

March 1, 2001

Table of Contents

1 Introduction

Java, JavaScript and browser plug-ins are powerful tools that enable developers to create sophisticated web-applications. However, each of these languages is subject to it's own set of limitations.

LiveConnect is a technology developed by Netscape Communications Inc. It allows intercommunication between these languages to create powerful, integrated web-applications. This communication takes place between an applet and a script on the same page or with a plug-in that was loaded by a page. It allows the following:

1. JavaScript can access Java applet methods, classes, packages and variables
2. Java applets can access JavaScript methods and properties
3. Java applets can call plug-in methods
4. Plug-in methods can be called by Java applets

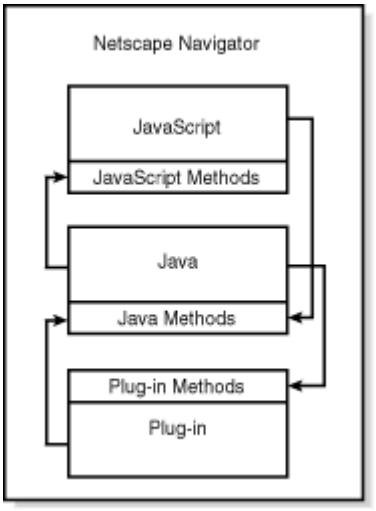

Figure 1 – LiveConnect Overview

Note that LiveConnect does not allow a plug-in to communicate directly with JavaScript, but this is a non-issue because it can be simulated using Java as an intermediary.

LiveConnect was implemented in Netscape's browser product, Netscape Navigator (version 3.0 and later). It was a revolutionary new technology because it allowed channels of communication on the client side without the need for interaction with the

server. After being introduced by Netscape, Microsoft incorporated it's own implementation of the LiveConnect technology in versions of Internet Explorer 4.0 and later.

Both Java and JavaScript have to be enabled in the browser for LiveConnect to work. If either of these is disabled, it will render this technology useless. The "Enable JavaScript" and "Enable Java" checkboxes both have to be checked, as is illustrated in the figure below.

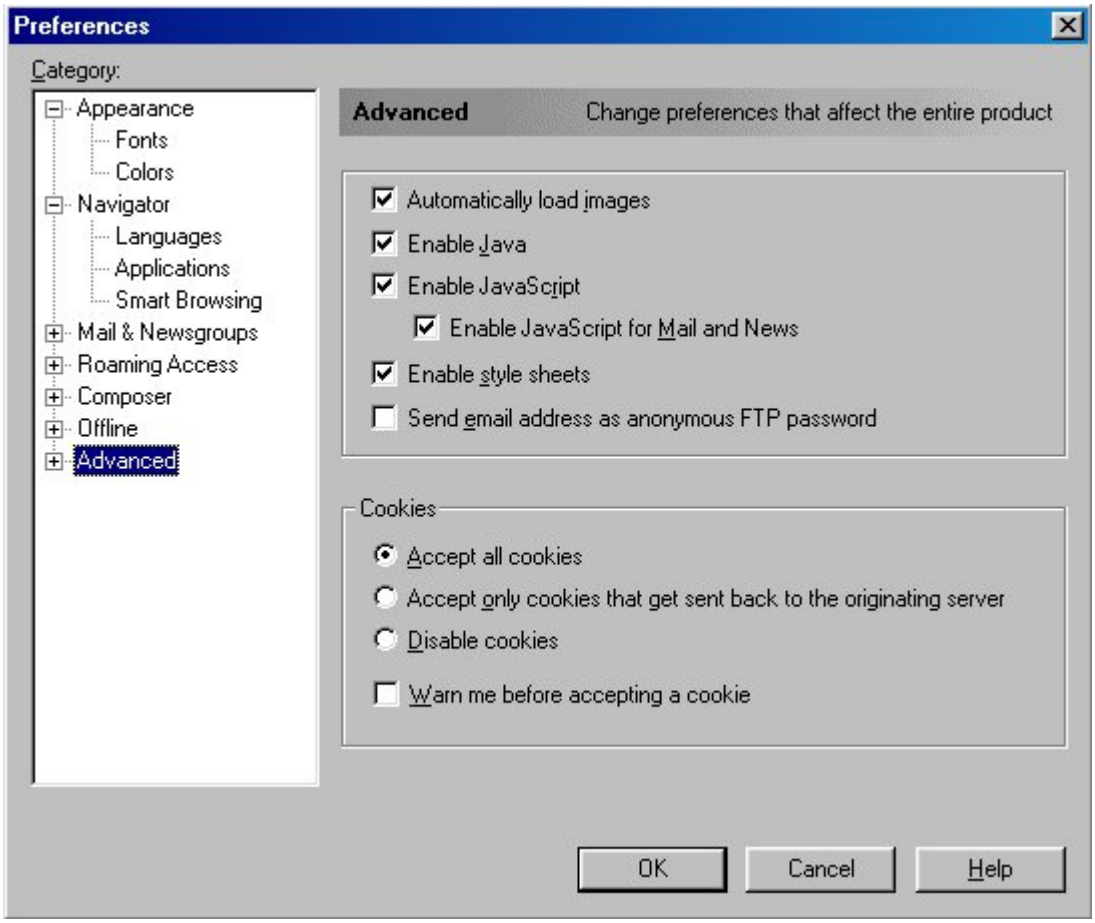

Figure 2 - Netscape Preferences Dialog

2 Setting Up the Development Environment

Before you can use create Java applets and plug-ins that use the LiveConnect technology, you need to set up the environment for proper communication to take place.

For Java to JavaScript communication:

1. Add the java40.jar file to your CLASSPATH. This file is distributed with Netscape browsers, and can be found in the Program\java\classes subdirectory of the Netscape distribution.
2. Import the netscape.javascript package in your Java applet.

   ```
   import netscape.javascript.*;
   ```

3. Give the Java applet permission to access the JavaScript in your HTML file. This is done by using the `MAYSCRIPT` attribute in the `<APPLET>` tag.

   ```
   <APPLET CODE="someapplet.class" WIDTH=... HEIGHT=... MAYSCRIPT>
   ...
   </APPLET>
   ```

If permission is not granted to access JavaScript, an exception will be generated.

For JavaScript to Java communication, the Java methods called by JavaScript code must be public.

For Java communication with plug-ins:

1. Add the java40.jar file to your CLASSPATH. This file is distributed with Netscape browsers, and can be found in the Program\java\classes subdirectory of the Netscape distribution.Java plug-ins are to be compiled with the Plugin class (or a subclass of the Plugin class) to allow JavaScript and Java applets to access the plug-ins. Java code may also need to declare objects of class Plugin.

3 Java to JavaScript Communication

Java to JavaScript communication is very simple. Manipulations of JavaScript objects from Java are done through a package created by Netscape called netscape.javascript. This package contains 2 classes - JSObject and JSException. The JSObject class acts as a wrapper for JavaScript objects. It allows Java applet code to access JavaScript functions, object properties, and data structures. The JSException class is used to throw exceptions when JavaScript errors are encountered.

3.1 JSObject Methods

The JSObject class contains the following methods for communicating with JavaScript. The most commonly used methods are getWindow(), getMember(), setMember(), call() and eval().

JSObject Method	Functionality	JavaScript Equivalent
public static JSObject getWindow(Applet a)	Returns a JSObject for the window containing the given applet passed in as a parameter.	-
public Object call(String methodName, Object args[ ])	Calls a JavaScript method.	this.methodName(args[0], args[1], ...)
public Object eval(String s)	Evaluates a JavaScript method passed in as a parameter.	-
public Object getMember(String name)	Retrieves a named member of a JavaScript object.	this.name
public Object getSlot(int index)	Retrieves an indexed member of a JavaScript object.	this[index]
public void removeMember(String name)	Removes a named member of a JavaScript object.	-
public void setMember(String name, Object value)	Sets a named member of a JavaScript object.	this.name = value
public void setSlot(int index, Object value)	Sets an indexed member of a JavaScript object.	this[index] = value

public String toString()	Converts a JSObject to a String.	-

3.2 Accessing JavaScript Functionality

Once you have set up the environment, you can start accessing JavaScript objects and functions from your Java applet.

1. First, you need to create a reference to the window that contains the JavaScript that you want to access, and of course, the applet itself. This is done through the JSObject static method getWindow().

   For example, in your Java code, you would write

   ```
   JSObject window = JSObject.getWindow(this);
   ```

2. To access JavaScript objects and properties, the getMember() method is very handy. Since the return type of the getMember() method is Object, you need to cast the return Object to the necessary type. Eg. JSObject, int, etc.

   For example,

   ```
   JSObject doc = (JSObject) window.getMember("document");
   JSObject myform = (JSObject) doc.getMember("someFormName");
   JSObject someTextField = (JSObject)
   myform.getMember("someTextFieldName");
   int screenHeight = (int) window.getMember("screen.height");
   ```

3. To set the properties of JavaScript objects, the setMember() is used. Note that the second argument must be a Java Object.

   For example:

   To set the background color of the page,

   ```
   doc.setMember("bgColor", "red");
   ```

   To set the value of a text field,

   ```
   someTextField.setMember("value", "someTextValue");
   ```

4. To call JavaScript methods of interest, either use the call() or eval() method. LiveConnect does not restrict these methods to the JavaScript built-in methods. User-defined functions can be called as well.

   For example, to bring up a message using the JavaScript alert() method,

   - Using the call() method:

     ```
     String[ ] message = {"An alert message."}
     window.call("alert", message);
     ```

   - Using the eval() method:

     ```
     window.eval("alert(\"An alert message.\");");
     ```

3.3 Java to JavaScript Data Type Conversions

Values passed from Java to JavaScript are converted as outlined in the following table.

Java Type	**Converts to --->**	**JavaScript Type**
Byte, char, short, int, long, float, double		number
boolean		boolean
JSObject		original JavaScript object
array		JavaScript pseudo-Array object (behaves just like a JavaScript array object)
object of any other class		JavaScript wrapper (can be used to access methods and fields of the Java object)

4 JavaScript to Java Communication

JavaScript can use LiveConnect technology in the following ways:

1) Call Java methods
2) Control Java applets
3) Control Java plug-ins

4.1 Calling Java Methods

A developer can directly call Java methods in JavaScript code. This is accomplished with regular Java syntax:

```
var today = new java.util.Date();
System.out.println(today);
```

The above example will print the value of the date object (the current date) to the Java console window, just as a Java applet would if it were to execute the same code. Any public method or instance variable can be accessed in JavaScript code.

Accessing packages in JavaScript is also accomplished using regular Java syntax, but the package name should have the prefix "Packages.". This is necessary because Java packages are properties of the Packages object (in JavaScript). The syntax for referencing a Java package is [Package.]<packageName>.<className>.<methodName>. For example, to access the ConnectionPool class's getConnection method of the com.phil.denis.db package in JavaScript, the following code would be used:

```
Packages.com.phil.denis.db.ConnectionPool.getConnection();
```

There are three special packages that do not need the "Packages." prefix. They are java.*, sun.* and netscape.*. These special packages are aliased because they are used so frequently. Both of the following statements are legal in JavaScript:

```
var myVector = new java.util.Vector();
var myVector = new Packages.java.util.Vector();
```

Since there is no JavaScript equivalent for Java's import statement, all calls must be fully qualified. To avoid this, a developer can create his own alias by assigning the class that would be imported (in Java) to a JavaScript variable:

```
var Vector = java.util.Vector;
var myVector = new Vector();
```

4.2 Controlling Java Applets

JavaScript can call Java methods, but it can also access the methods and instance variables of any applet found in the web page containing the JavaScript. Applets can be referred to by accessing the applets array of the document object, or by using the property of the document object named after the applet. The applets array can either be indexed by number or by the name of the applet. If an index is used, the applets are numbered in the order they appear from the top of the page and from left to right.

An example of the HTML needed to put an applet on a web page is:

```
<APPLET CODE=”MyApplet.class” NAME="myApplet" WIDTH=100
HEIGHT=100>
```

The above applet could be referenced in any of the following ways:

By Name	`document.myApplet`
By Index	`document.applets[0]`
By Associative Name	`document.applets[“myApplet”]`

All public variables of the applet are available from JavaScript. All static methods and properties of an applet are accessible as methods and properties of the Applet object (which is contained in the document object).

There are several possibilities for using JavaScript to control Java applets. One could include form elements on a screen to capture some user information and use a JavaScript event handler to call some method of the Java applet based on the value input into the form field. Other uses include having buttons on the HTML page to

start/stop the applet. These buttons would be connected to JavaScript methods that called the applet's start() and stop() methods.

4.3 Controlling Plug-ins

Plug-ins can be referenced as elements of the embeds array (which is a property of the document object). They can be accessed by number or as a variable of the document array, similar to the way that Java applets are accessed. The embeds array can either be indexed by number or with the name of the plug-in. An example of the HTML needed to put a plug-in on a web page is:

```
<EMBED SRC=myAvi.avi NAME="myEmbed" WIDTH=100 HEIGHT=100>
```

All of the following examples of referencing the above plug-in are legal syntactically:

By Name	`document.myEmbed`
By Index	`document.embeds[0]`
By Associative Name	`document.embeds["myEmbed"]`

4.4 JavaScript to Java Data Type Conversion

Values passed to Java from JavaScript are converted as indicated in the following table:

JavaScript Type		**Java Type**
Numbers	**Converts to --->**	Float
Boolean		Boolean
Strings		Strings
Other		JSObject

A consequence of the data conversion that takes place is that JavaScript values appear in Java as objects of the java.lang.Object class. They have to be cast to the appropriate type before being used in Java code.

5 Java to Plug-in Communication

In order for Java to communicate with a plug-in, the plug-in must have support for LiveConnect. The plug-in must have a LiveConnect API that Java can use. For example, the LiveAudio plug-in (a plug-in for playing sounds) would have public play() and stop() methods.

The <APPLET> tag in the web page must have the MAYSCRIPT attribute so that the applet and the plug-in can communicate. The MAYSCRIPT attribute is also necessary for the applet to be able to create a reference to the plug-in from JavaScript.

Java associates plug-ins with the class Plugin from the netscape.plugin package. To reference a plug-in embedded in a web page,

```
Plugin plugin = (Plugin) doc.getMember("SomePluginName");
```

For example, to reference a LiveAudio plug-in,

```
JSObject window = JSObject.getWindow(this);

JSObject doc = (JSObject) window.getMember("document");

SoundPlayer plugin = (SoundPlayer)
doc.getMember("LiveAudioPluginName");
```

Now, you can manipulate the plug-in by simply calling its methods.

```
plugin.play();

plugin.stop();
```

6 Security Considerations

Downloading and executing code from the Internet can introduce security problems. The computer executing the code could open itself to external monitoring, exporting of information, and other potentially dangerous attacks. Some attacks can change the configuration of the host or even remove files, rendering the machine unusable.

Applets that are executed in a browser run in a “sandbox” that introduces security restrictions on the Java code. This is necessary because Applets are usually written by unknown authors and consist of untrusted code. Some examples of the security restrictions are:

- Applets cannot read or write to the local file systems
- Applets cannot perform most networking operations (such as opening socket connections)
- Applets cannot make use of some of the AWT facilities (such as initiating print jobs)

The consequence of opening the communication channels that LiveConnect does is that it introduces doors to security flaws in the browser. The “sandbox” that Java applets run in does not allow potentially dangerous operations to be performed by applet code. However, these concerns are reopened and represent real threats once LiveConnect is introduced. For example, a Java applet can ask JavaScript to open a socket on its behalf. This socket can be used to send the server private information and perform other malicious activities.

6.1 LiveConnect Attacks

Like all other software projects, LiveConnect developers had to meet tight deadlines. This means that there were many implementation flaws in its infancy, some of which are still being discovered and exploited to this day. These flaws enable intruders to perform attacks on a host computer that is executing their code. One such attack, called the Singapore Privacy Bug, allowed an attacker to track user activities and retrieve information from the cookies file. Another attack exploited the implementation flaw of

LiveConnect that goes against the need-to-know security principle. Applets communicate with JavaScript through the JSObject. If an applet overrides it's stop() method, it will continue to run after the web page containing it is replaced by a new page. If the applet then calls a method of the JSObject after the web page that loaded it is no longer showing, it throws a JSException. It is not difficult to hand-edit the JSObject class to catch the JSException that would kill off the applet. Thus, applets can be made that continue to run and call JavaScript functions after the web page that contained it is long forgotten. This bug was fixed in Netscape 4.01a, but many older browsers are still in use. Also, if this type of bug was present in early versions of LiveConnect, it makes one wonder what security holes are yet to be discovered.

Other attacks include tracking or monitoring the user's activities (i.e. the web pages he/she visits) and gathering privileged information such as usernames and passwords. These attacks are much more complicated than the ones listed above and are beyond the scope of this report. The reader need only be aware that these types of attacks are real and that LiveConnect is a mixed blessing at best. It is a technology that allows great interaction between Java, JavaScript and plug-ins, enabling developers to build powerful applications, but with great power comes great responsibility. There are some people out there who would exploit this technology for malicious purposes.

7 Conclusion

There are many web technologies that have the capabilities to accomplish different tasks with ease. For example, JavaScript is simple to use, and it can use the Document Object Model to create amazing web applications. It is also easy to define functions in an HTML page with JavaScript that manipulate the contents and properties of the web page. Java is a powerful platform-independent language that is flexible and easy to use. Plug-ins give browsers a way to interact with users through media content like sounds and video. Netscape's LiveConnect technology transforms the browser into extremely powerful application development platform by combining the strengths of JavaScript, Java, and plug-ins so that they can interact in real time on the client side. Web developers should be wary of the threats that LiveConnect poses, though. If used properly and responsibly, LiveConnect allows web developers to expand their creativity and construct interactive and complex web applications.