MICRO FOCUS

# Reflection for the Web

## Reference Guide

**November 2017**

Legal Notice

# Contents

# Reflection for the Web - Reference Guide

This Reference Guide includes introductory and advanced topics for use with Reflection for the Web.

Beginning in version 12.3, Reflection for the Web is a standalone client that requires the Host Access Management and Security Server and its Administrative Server. (In previous versions, the Advanced topics were available from the Administrative WebStation.)

In this guide:



## How Reflection for the Web Works

Using Reflection for the Web and Management and Security Server, you can configure secure web-based terminal emulations sessions that connect to host applications located inside or outside the firewall.

Applets are downloaded to each user's workstation as needed and are cached locally for faster performance. Sessions are centrally managed and secured using Host Access Management and Security Server.

Briefly, here's how Reflection for the Web works:

1. An administrator installs Reflection for the Web on a web server and either installs or uses an existing installation of Management and Security Server
2. The administrator uses the Administrative Console (in Management and Security Server) to create, configure, and secure terminal emulation sessions. Optional security settings can be configured on a per-session basis.
3. A user clicks a link to start a terminal session
4. The Reflection for the Web emulation applet is downloaded to the user's workstation and is cached locally.
5. The user connects to and communicates with the host system using the downloaded emulation applet.
6. When the session is closed (**Save/Exit**), settings are sent to the Management and Security Server.

# 1 Configuring Reflection for the Web Sessions

The configuration of a web-based terminal emulation session involves both the Reflection for the Web applet and the host session.

When you create a session, you determine its configuration settings using applet defaults, applet parameters, host system defaults, and a configuration file. (The applet parameters and configuration file are optional.) Users can then create their own user preference files to change some elements of the default configuration. The degree to which users can save their own configurations is itself part of the session configuration.

## Configuring a Reflection for the Web Session

The easiest way to create a terminal session is by using the Manage Sessions panel in the Management and Security Server's Administrative Console. The default choices for a given host session type are provided.

Some configuration settings can be controlled only by Reflection for the Web applet parameters:

- Applet name (nameattribute)
- Shortcut menu access (shortcutMenu  parameter)
- New window (frame parameter)
- Window title (title parameter)

These applet settings remain in effect each time a new configuration file is opened or saved.

In other cases, the same configuration element can be set using applet parameters, a configuration file, or user preferences. The load order of the information determines the final configuration.

### Load Order of Configuration Information

Reflection for the Web loads configuration information from four sources when it starts a terminal session. These sources are listed in the order that they are loaded when Reflection runs. -items later in the list (such as user preferences and applet parameters) are loaded later and can override earlier items:

1. **Applet and host system defaults**--Default configuration values, usually embedded in the program code and inaccessible to users.
2. **Configuration files**--Component configurations, such as keyboard mapping, saved to a file by a system administrator. Within the configuration file, a system administrator can control how much customization users are able to save with user preferences.

3. **User preferences**--Customized session settings saved locally by each user. The range of configuration that the user can save is determined by the system administrator in the configuration file. For more information, go to Configuring User Preferences.
4. **Applet parameters**--Applet-based administrator settings. Parameters are loaded last and therefore have the highest priority. For a list of the parameters used in Reflection applets, see Applet Attributes and Parameters.

Using the load order you can generate different configurations for different groups in your enterprise. Create a session with a basic configuration file. Import the configuration file into new sessions; then, use applet parameters to override the basic configuration with settings customized for a specific user group.

# Using Configuration Files

Sessions launched from the **Manage Sessions** panel always have an administrator profile, regardless of the actual session's profile. Opening a session as an Administrator allows you to set choices on the Administration menu. You can also set up a session with an Administrator profile and provide access to the Administration menu.

Once you have a configuration file, you can import it into other sessions. The imported configuration file is saved with a new name associated with the new session.

---

**NOTE:** Configuration file settings for web-based sessions can be overridden by user preferences or applet parameters, and some session settings can be set by applet parameters only. See Load Order of Configuration Information.

---

# Configuring User Preferences

User preferences are session settings selected by each user and stored locally on the user's machine. Based on the Reflection configuration load order, these values override the configuration file created by the system administrator.

The administrator uses the Set User Preference Rules dialog box to determine which settings the user is allowed to change and save. After the administrator sets the preference rules and saves them in a configuration file, users can save their preferences by clicking Save Preferences on the File menu in Reflection. Any allowed preferences that the user saves are automatically loaded the next time the user starts the same Reflection session. If the administrator does not set any preference rules, the Save Preferences option on the File menu appears dimmed and a user preferences file cannot be saved.

### Blocking user preferences

By default, existing user preferences are always loaded when a user starts a Reflection session. To prevent existing preferences from loading, use the loadUserPrefs parameter in the terminal session applet tag

## Naming conventions for user preference files

Reflection saves and names the user preferences file by combining the applet name with the .pref extension. (The applet is named using the name attribute in the applet tag.) For example, if an applet is named `AccountingSession`, the preference file will be named `AccountingSession.pref`. If the applet is not named, users will not be able to save preferences for the session.

## Storage locations of user preference files

Preference files are stored on the user's computer under the RWEB_PREFS folder. To find the home folder, click About Reflection on the Help menu of a terminal session, then click the System Information tab.

## Enabling and saving user preferences

After launching a new session:

1. Use the User Interface Profiler option on the Administration menu to set up which menus, dialog boxes, and toolbars should be available to users.
2. In the terminal window, click Set User Preference Rules on the Administration menu
3. In the Set User Preference Rules dialog box, select the components that will allow user changes to be saved when the user exits Reflection
4. Click OK to close the dialog box.
5. Set any other options in the terminal session that you want to include.
6. When you're done configuring the session, choose Save and Exit from the File menu. Click Save/Exit. The new session appears in the session list.

When a user runs a session, he or she can save the settings for the components that you permitted when you created the configuration file. The user has the following options when user preferences are enabled:

- To save preferences in the Reflection terminal session, click the Exit command on the File menu. All preferences allowed by the administrator in the configuration file are saved on the local computer.
- To clear the preferences saved in the terminal session, click the Reset Preferences command on the File menu. The session settings return to the defaults in the configuration file.
- To create additional copies of the session with different sets of preferences, click Duplicate from the Links List. The same preferences can be modified in each duplicated session.

If an administrator has not enabled user preferences for the terminal session, the Reset Preferences command on the Reflection session File menu is dimmed and preferences are not saved upon exiting.

# Recording macros

The Reflection for the Web macro recorder makes it easy to automate repetitive tasks. You can record macros in the Session Manager, and you can record them in end user sessions if it is enabled in the session profile.

---

**NOTE:** Automated sign-on macros, such as single sign-on macros, allow users to log on to hosts without entering their usernames and passwords. This type of macro recording is initiated from the Session Setup dialog box within the emulator and has a different set of options and requirements than those described below.

---

**To create a macro in a green screen terminal session**

1. Choose Macro > Start Recording.

2. Perform the actions you want to record, and then choose Macro > Stop Recording.
3. Name the macro in the Save Macro dialog box and click Save.

To run the macro, choose Macro > Playback, click on the macro name, and click Play.

The Save Macro dialog box also allows you to set options for the macro. You can enter a description that will appear in the Playback dialog box; record the initial cursor position for the macro; and set the macro to run at startup. A table shows the individual steps in the macro, and for each step you can choose whether to automatically use the response you gave when you recorded the macro or to prompt for the response when the macro is run.

Macros recorded in the Manage Sessions panel are saved on the server. All macros, including startup macros, are available to the end user in the Play Macro dialog box if this capability is not restricted by the session profile. Even if user access to macros is restricted, however, a macro designated as a startup macro still plays back when the session starts.

Macros recorded by end users are stored in a user-specific directory on the local machine, but they can be exported and used in other sessions of the same type.

# 2 API and Scripting

The Reflection for the Web emulation applets include two application programming interfaces (APIs) that let you automate routine tasks.

- The JavaScript API, or JSAPI for short, is intended for scripting Reflection sessions using high-level scripting languages such as JavaScript and Microsoft's JScript or VBScript. The JavaScript API is intended primarily to help you automate host logon tasks and provide access to setup dialog boxes in the emulator applets, and the methods and properties provided are designed specifically with these tasks in mind.
- The Reflection for the Web **Emulator Class Library**, or **ECL** API, is designed primarily for Java programmers and includes more advanced features such as event listeners, access to user interface components, and screen recognition.

  The ECL development kit is included on your product download in an `api` folder. The documentation in the toolkit provides more detail about using the API to automate Reflection tasks from standalone Java applets and applications, as well as through "attachment classes," custom classes that you attach to a running terminal session.

**Related Topics**

- Using ECL

## Using ECL

Even if you are not a Java programmer, you can still take advantage of many ECL features from JavaScript. A built-in ECL module is provided with Reflection that integrates many of the ECL features with the LiveConnect features of JavaScript. When you use this module, called JSEventNotifier, you can write JavaScript code that registers listeners to receive event notification from the Reflection ECL; in many ways, this makes your JavaScript code act similar to a Java applet.

The JSAPI is the API documented in this section. API examples are shown primarily using JavaScript and VBScript; other scripting languages are typically similar in structure.

Because the API's focus is on logon tasks, it is recommended that you set all of your other configuration options using the Reflection terminal session menus, and then save those settings to a configuration file. See Configuring Reflection for the Web Sessions for more information about creating and using configuration files.

Then, in the web pages that you use to launch Reflection terminal sessions, script only those tasks needed to perform the logon.

- What you need to use the API
- How to use the API
- Using the API from JavaScript and VBScript
- Accessing the ECL API from JavaScript
- Timing issues when scripting Reflection
- Browser dependencies when using the API

- API Reference
- API Examples

# What you need to use the API

If you plan to write scripts using JavaScript, JScript, or VBScript, you need only a text editor to write your scripts and a web browser to run them. Most web browsers support JavaScript to Java communications. VBScript requires Microsoft Internet Explorer for Windows.

If you plan to write Java applets using the ECL API, you need a Java development environment, such as JetBrains Inc.'s IntelliJ IDEA, the NetBeans IDE or Eclipse. The development environment you use must also support Java version 1.6 or higher. The ECL documentation has more details about setting up your development environment.

You can learn more about language syntax and how to program in JavaScript, VBScript, and Java from the many online resources and publications available.

# How to use the API

Regardless of the scripting language you use when accessing the Reflection for the Web API, the basics are the same.

## Incorporate the code into a session

There are two methods for incorporating JavaScript into your sessions:

- Static page
- Frameset - With this option you benefit from dynamic session generation.

### Static page

1. Create a session in the Administrative Console, Manage Sessions panel. Launch and save the session.
2. Open the session from the list of sessions.
3. Under Advanced, click More and scroll down to View Session HTML. Copy the HTML and save it to the `deploy` folder, giving it a name with a .html extension. You are automatically given a name for the applet that you can use in your code.
4. In the applet HTML, locate `param name=configuration` and view the name of the session's config file.
5. Copy the configuration file from `deploy/dyncfgs` to `deploy`.
6. Add code to the HTML page.

Your user’s access the HTML page and deploy the session using the URL you provide them.

### Frameset

1. Define and save the session in the Administrative Console, Manage Sessions panel.
2. Write your code and save it in a page in the `session` folder.
3. Create an HTML frameset and save it in the `deploy` folder.
4. Set the source of one frame to the full URL for the session. (This is the same URL you would use to directly access the session.)

5. Set the source of the second frame to the code page.
6. To deploy the session, give your users the URL to the HTML frameset.

## Referencing the session applet

Manipulate the Reflection session through its API methods and properties by referencing the named session applet from your JavaScript code.

For example, a JavaScript line that displays the About box for the applet above would look like this:

```
var api = document.IBM3287.getAPI("JSAPI","IBM3287");
   api.showDialog( "aboutBoxDialog" );
```

You could also perform the same task in JavaScript using the document's `applets[]` property, which is an array of the applets in the document (use this technique if the applet's name contains spaces or punctuation characters):

```
document.applets["IBM3270Applet"].getAPI("JSAPI","IBM3270").showDialog(
"aboutBoxDialog" );
```

If you don't name an applet, you can reference it in JavaScript by its index in the `applets[]` array, like this:

```
   // If the Reflection applet is the only one on the page, its index
   // in the applets[] array is 0.
   document.applets[0].getAPI("JSAPI","IBM3270").showDialog( "aboutBoxDialog" );
```

---

**TIP:** Because the Reflection for the Web API is intended primarily for scripting logon tasks, it is recommended that you set all of your other Reflection session configuration options using the Manage Sessions panel. Then, in the web pages that you use to launch the Reflection session, script only those tasks needed to perform the logon.

---

# Using the API from JavaScript and VBScript

To use the API from scripting languages such as JavaScript and VBScript, you need only a text editor to write your script files, and a web browser to test and run them. In addition, the web browser must support the scripting language that you use, and scripting must be enabled in the browser. If scripting is either not available or not enabled, you can display an appropriate message using the HTML tags `<noscript>` and `</noscript>`.

To use API methods from a scripting language, name the Reflection session applet on the web page, and then reference the API from your script, as described in How to use the API.

- If the Reflection session applet and the script code are both in the same HTML file, you reference the applet's name in the current document. Many of the API examples show how this is done, including a simple example that uses JavaScript buttons to open Reflection dialog boxes.
- If the Reflection session applet and the script code are in different HTML files--for example, the applet is in one frame of a frameset file and the script is in a different frame of the frameset--you must supply the full frame name of the applet's document from your script code. The API examples include a web page that shows how to use JavaScript in a different HTML file than Reflection.

## Scripting tips

When using the API from a scripting language, keep in mind the following:

- Variables are typically untyped in the scripting language, but have specific data types in Reflection. If a variable cannot be converted to the appropriate type for the API method, a scripting error will occur.
- You may find it convenient to store a reference to the Reflection API in a script variable, especially when working with a frameset (because the location of the applet can result in a long reference name). For example, instead of prefixing every API method with something like `parent.rightframe.document.IBM3270Applet.getAPI("JSAPI","IBM3270")`, you might create a variable called `mAPI`, and set it to the name of the Reflection applet like this:

  ```
   // In JavaScript
       var mAPI =
  parent.rightframe.document.IBM3270Applet.getAPI("JSAPI","IBM3270");
  ```

  or

  ```
  ' In VBScript
       Set mAPI =
  parent.rightframe.document.IBM3270Applet.getAPI("JSAPI","IBM3270")
  ```

  If you use this technique, keep in mind the scope of the variable and where in your HTML file the script is located. If you attempt to assign the variable before the applet or API exists, you may get a script error saying that the object has no properties, that the variable is not an object, or you may get a null return value for the API. Also notice the use of the Set statement in the VBScript example; the Set statement is required in VBScript to create a reference to an object.
- The way you specify non-printing characters, such as escape sequences, control characters, etc., when waiting for strings or transmitting data to the host (for example, using transmitString), depends on your scripting language. If you're using JavaScript, you can generally use octal or hexadecimal notation; for example, the octal value \015 or the hex value \x0D for a carriage return. If you're using VBScript, you should use either a VBScript constant (such as vbCR for the carriage return character) or the Chr function to specify the character's value.
- Most API methods do not generate specific errors (that is, they don't throw Java exceptions). However, many methods return `True` or `False` to indicate the success or failure of the method. You should check the return value of these methods before deciding how your script should proceed.
- Depending on the scripting language you use, you may or may not be able to trap errors that occur in your script code. In JavaScript, you can trap errors with the `Window.onerror` event handler, to catch situations in which a JavaScript error occurs. In VBScript, you can trap errors in individual procedures with the `On Error Resume Next` statement.

  You might want to trap errors in cases where you're not certain that an API method is available; if a method is not available and you try to call it, a trappable error occurs. This is different than the situation described above, in which the method is available but fails to produce the desired result.

# Accessing the ECL API from JavaScript

Even if you're not a Java programmer, you can still take advantage of many of the powerful features of the Reflection Emulator Class Library (ECL).

**JSEventNotifier**, a built-in module provided with Reflection for the Web, integrates many features of the Reflection ECL API with the LiveConnect features of JavaScript. When you use the JSEventNotifier module, you can write JavaScript code that registers listeners to receive event notification from the Reflection ECL; in many ways, this makes your JavaScript code act similar to a Java applet.

The JSEventNotifier module works as an ECL "attachment" class, a type of custom Java module that attaches itself to an existing Reflection session after the session has been initialized. Attachment classes usually fall into two categories:

- Macro-like attachments, which are invoked on demand to perform a single task.
- Monitor-like attachments, which are loaded during the session but don't perform any action until some event occurs or some screen appears, at which time they perform their task.

These types are not exclusive; an ECL attachment class may act like a macro initially, then wait and watch for a particular screen in order to perform another task.

Because the JSEventNotifier module uses LiveConnect, it can send messages to your waiting JavaScript; that means you don't have to write JavaScript code that continually polls the terminal session to determine if new data has arrived or if the connection state has changed.

## Getting started with the JSEventNotifier module

If you already have installed the Reflection for the Web files on your web server, you already have the JSEventNotifier installed and ready to use. But first, you must modify the APPLET tag that controls your Reflection session, and you must write a bit of JavaScript code.

The following sections describe how to make these changes so the JSEventNotifier module gets loaded automatically when the Reflection session starts.

### Modifying the Applet tag

To enable the JSEventNotifier module, add the following parameter to the APPLET tag of the web page that launches your Reflection terminal session:

```
<param name="onStartupJavaClass"
value="com.wrq.eNetwork.ECL.modules.JSEventNotifier">
```

In addition, add the `MAYSCRIPT` attribute to the `APPLET` tag. The `MAYSCRIPT` attribute allows a Java applet (Reflection, in this case) to communicate with JavaScript, in addition to the JavaScript-to-Java communication that's always available.

Depending on how your session is generated, you might add these values using the Applet Parameters section of the Manage Sessions panel in the Administrative Console, or by manually editing the HTML page that launches your session.

An applet tag for an IBM 3270 session might look something like this:

```
<APPLET name="JSSession" CODE="com.wrq.rweb.Launcher" codebase="./ex"
      width=800 height=600 archive="Launcher.jar" MAYSCRIPT>
   <param name="launcher.sessions" value="IBM3270">
   <param name="prefsName" value="myIBMsession">
   <param name="hostURL" value="tn3270://myhost.mycorp.com">
   <param name="onStartupJavaClass"
value="com.wrq.eNetwork.ECL.modules.JSEventNotifier">
</APPLET>
```

**NOTE:** If you use the JSEventNotifier module to receive initial notification, you can omit the parameter `preloadJSAPI` from the Reflection applet tag. If you do include both parameters, you should also include a JavaScript function called `jsapiInitialized` in your script. Some browsers, such as Firefox, will not call the `ECLInitComplete` function described next if the `jsapiInitialized` function is not also present but is called because the `preloadJSAPI` parameter is set.

## Accessing the ECL API

When an HTML page that contains an `APPLET` tag like the one above is loaded by the web browser, the Reflection session launches. When the session has completed its initialization, it loads the module specified by the `onStartupJavaClass` parameter, in this case, JSEventNotifier.

After the JSEventNotifier module is loaded, it invokes a JavaScript function on your web page called `ECLInitComplete`, and it passes a reference to the JSEventNotifier class itself. (If you prefer to call the JavaScript function that receives the initial notification something different, specify the function name using the `initCallback` parameter in the Reflection `APPLET` tag.)

Once your JavaScript function receives a reference to the JSEventNotifier object, it can obtain the JavaScript API object by calling the `getJSAPI()` method. It can also use other methods available though the JSEventNotifier module for registering event listeners and setting up screen recognition systems.

The power in using the JSEventNotifier module is that when the `ECLInitComplete` function in your JavaScript gets invoked, you're assured that the Reflection terminal session is fully initialized and ready for scripting. This means you do not have to poll your Reflection session using the `getInitialized()` method to determine when Reflection is ready for scripting, making your scripts much more robust and efficient.

**NOTE:** If you want your script to receive initialization notification but do not need to use the ECL API functions available via the JSEventNotifier module, you can use the lighter-weight notification mechanism provided by the `preloadJSAPI` parameter. See Using Notification to Determine When Reflection Is Initialized for more information.

## Example

The following example shows how to write a simple JavaScript that receives initialization notification, then sets up a listener to receive connection state changes.

```
<script language="JavaScript">
<!--
    var mJSNotifier = null;
    var mJSAPI = null;

    /*
     *  This function receives the initial notification when the
     *  Reflection session is initialized and the JSEventNotifier
     *  module is done loading. It stores references to the
     *  notifier module itself and the JSAPI object,
     *  sets up the connection listener, then starts the host
     *  communications.
     */
    function ECLInitComplete(jsNotifier)
    {
        mJSNotifier = jsNotifier;
        mJSAPI = mJSNotifier.getJSAPI();
        registerConnectionListener();
        mJSAPI.connect();
    }

    /*
     *  Adds a listener to receive connection state changes. The
     *  parameter "connectionCallback" specifies the name of the
     *  JavaScript function that should be invoked when the
     *  connection state listener is invoked.
     */
    function registerConnectionListener()
    {
        mJSNotifier.addConnectionCallback("connectionCallback");
    }

    /*
     *  Callback function for connection state changes. This is the
     *  function defined above in the registerConnectionListener()
     *  function. This function will be invoked by the JSEventNotifier
     *  module when the connection state changes. The "state" parameter
     *  will contain the current state of the connection, either "true"
     *  if connected, or "false" if disconnected. The parameter is a boolean,
   *  but Firefox seems to see it as a string, while Internet Explorer sees
 *  it as a boolean.
     */
    function connectionCallback(state)
    {
    if ( state == true || state == "true" )        // session became connected
      {
        alert( "Session is connected!" );
      }
        else if ( state == false || state == "false" )  // session became
disconnected
    {
          alert( "Session is disconnected!" );
    }
    }
//-->
</script>
```

The JSEventNotifier module is part of the ECL API and its methods are detailed in the ECL documentation that's part of the ECL development kit.

**TIP:** The JSEventNotifier module also provides methods for accessing many of the ECL API objects, including ECLSession, ECLPS, ECLOIA, and others. Although you can access these objects from browser-based scripting languages, they are not fully compatible with these languages due to browser security restrictions, and invoking methods on these objects may result in script failures (as indicated by exceptions in the Java console). You should restrict your use of JSEventNotifier methods to those methods provided by the JSEventNotifier class itself, and to the methods in the JSAPI object you obtain.

# Timing issues when scripting Reflection

When working with the API, there are situations where timing and synchronization can become an issue between your script and the Reflection terminal session applet. For example:

- You want to perform a task automatically after a Reflection session starts, perhaps to open a dialog box for the user. In this case, you need to make sure that the Reflection session is initialized before you open the dialog box. See Determining if Reflection Is Initialized for more information.
- You want to get information from a Reflection session dialog box before proceeding with your script. See Synchronizing Dialog Box Input for more information.
- You want to automate a host logon procedure by transmitting a user name and password after the appropriate host prompts have been received. In this case, you need to wait for the appropriate host prompts before transmitting the responses. See Waiting for Strings and Transmitting Data for more information.

# Browser dependencies when using the API

The two most popular Microsoft Windows-based web browsers--Microsoft Internet Explorer and Firefox--both support JavaScript (along with Java). JavaScript to Java communication is also supported in many web browsers on other platforms, including Mozilla-based browsers (various platforms) and Apple Computer's Safari browser (version 1.2 or later) on the Macintosh. Microsoft Internet Explorer for Windows also provides VBScript, a scripting language with a syntax and structure similar to Visual Basic and Visual Basic for Applications, but not similar to JavaScript. If you write scripts using VBScript, they can run only in Internet Explorer for Windows.

If you expect to have multiple instances of Reflection emulation applets running at the same time and you need to access them using the API, it is important that each applet instance have a unique name (which you can specify using the Reflection `prefsName` parameter in the `<applet>` tag), and that your script use that unique name.

If you're using static web pages, one way to generate a unique name for each applet is to generate the applet tag and the scripts that reference the applet dynamically using JavaScript, and generate a unique applet name each time the web page is loaded. You could do this by getting the current system time when the page is loaded, and using that as the applet name.

## Using the Java plug-in with the API

If you're running Reflection using the Java Plug-in, you can access API methods via JavaScript (and VBScript in Internet Explorer) as you normally do, but you must also do the following:

Add the parameter preloadJSAPI to the Reflection session and set the value to "true".

For more information about using the Java Plug-in with Reflection, see (http://www.oracle.com/technetwork/java/javase/downloads/index.html).

# API Reference

This is a list of Reflection for the Web API methods, properties, and dialog box constants. Properties are shown with their type (such as "String property") and read/write status; properties are manipulated using the "property accessor methods" listed below. Dialog box constants are used with the `showDialog()` method listed below. The list is arranged alphabetically.

- ftpLogin (String, String, String, boolean)
- ftpOptions
- ftpProtocol
- ftpPwd()
- ftpReceiveFile (String, String, Int)
- ftpReceiveFiles (String, int)
- ftpSendFile (String, string, boolean, int)
- ftpSendFiles (String, int)
- ftpUI
- getAPI()
- getBoolean(String)
- getCoordinateBase()
- getCursorColumn()
- getCursorPosition()
- getCursorRow()
- getDisplayText (int, int,i nt, int, boolean, string)
- getFieldText (int, int, int)
- getHostName()
- getHostStatusText (int, int)
- getHostURL()
- getInitialized()
- getInteger(String)
- getPort()
- getPrintToFileFolder()
- getPrintToFileName()
- getString(String)
- ignoreUserTyping
- importKeymap (InputStream)
- indReceiveFile (String, string, boolean, boolean)
- indSendFile (String, string, boolean, boolean)
- isConnected()
- keyMapConfigure
- keyPaletteOptions
- loadJavaClass()
- mouseMapConfigure
- playbackMacro (macroName)
- printDBCSScale
- printerConfigure
- printToFile
- requestDisplayFocus()

- resetUserPreferences()
- saveUserPreferences()
- screenPrint()
- sessionConfigure
- setBoolean (String, boolean)
- setCoordinateBase()
- setCursorPosition (int,int)
- setHostURL (String)
- setInteger (String, int)
- setPrintToFileFolder (String)
- setPrintToFileName (String)
- setString (String, string)
- sftpAllowUnknownHost
- showDialog (String)
- showHotspots
- sshAllowUnknownHost
- streamIsEncrypted
- terminalConfigure
- terminalModel
- transmitString (String)
- transportTerminalKey
- transportType
- waitForCursorEntered (int, int, long)
- waitForCursorLeft (int, int, long)
- waitForDisplayString (String, long)
- waitForDisplayStrings (Vector, long)
- waitForHostPrompt (String, long)
- waitForIncomingData (long)
- waitForKeyboardLock (long, long)
- waitForKeyboardUnlock (long, long)
- waitForString (String, long)
- waitForDisplayString (String, boolean)
- waitWhileDisplayString (String, int, int, long, boolean)
- xfr400ReceiveFile (String, string, boolean)
- xfr400SendFile (String, string, boolean)

## aboutBoxDialog

Dialog box constant. When used with the showDialog method, opens the About Reflection dialog box.

```
public void showDialog( "aboutBoxDialog")
```

## apiExit

Method. Closes the Reflection for the Web applet without invoking the Java class specified by an `onExitJavaClass` applet parameter.

```
public void apiExit()
```

## apvuReceiveFile()

Method. Transfers a file from an IBM mainframe to the desktop computer, using the APVUFILE transfer protocol. This method is valid only when using a double-byte enabled version of Reflection for the Web.

```
public int apvuReceiveFile( String localFile,
String hostFile,
int transferType,
boolean showStatus )
```

## apvuSendFile()

Method. Transfers a file from the desktop computer to an IBM mainframe, using the APVUFILE transfer protocol. This method is valid only when using a double-byte enabled version of Reflection for the Web.

```
public int apvuSendFile( String localFile,             String hostFile,
int transferType,
boolean showStatus )
```

## autoPrintLineFeed

Property. This property determines whether Reflection automatically wraps to a new line if the host attempts to print a character beyond the edge of the host-defined page. Read/write.

```
Setter: public void setBoolean( "autoPrintLineFeed", boolean enable )

Getter: public boolean getBoolean( "autoPrintLineFeed" )
```

## buttonPaletteConfigure

Dialog box constant. When used with the showDialog method, opens the Button Palette Setup dialog box.

## cancelPrintJob()

Method. Cancels the current print job by sending a Cancel Print message to the host. The printer continues to print until all data already sent to the printer is done printing.

```
public void cancelPrintJob()
```

## colorConfigure

Dialog box constant. When used with the showDialog method, opens the Color Setup dialog box.

```
public void showDialog( "colorConfigure" )
```

## connect()

Method. Establishes a connection to the host computer specified in the applet parameter hostURL. Via the API, you can use the getHostURL method to specify the host URL.

```
public void connect()
```

## declans

Property. Specifies the text of an answerback message that gets sent to the host in response to an ENQ character. This property is linked to the Answerback message box in the Advanced Terminal Setup dialog box. Read/write. The name of this property derives from the mnemonic for the VT terminal's "Load Answerback Message" control sequence.

```
Setter: public void setString( "declans", String message)
Getter: public String getString( "declans" )
```

## deviceName

Property. Indicates the device name (also known as the LU name) that the session is currently connected to. This property is linked to the Device name box in the Session Setup dialog box. Read/write.

```
Setter: public void setString("deviceName", String inDevice )
Getter: public String getString("deviceName")
```

## disconnect()

Method. Disconnects the current session.

```
public void disconnect()
```

## display (string, boolean)

Method. Displays a string of text in the terminal window at the current cursor position, without sending the text to the host computer (compare this with the `transmitString` method). This method can be handy for displaying informational messages in the terminal window.

```
public void display (String inString, boolean interpret)
```

## emulateFormFeed

Property. When this property is True (the default), a form feed in the data stream closes the current page and starts a new one. When this property is False, Reflection determines how many line feeds are required to reach the bottom of the page (as defined by the host) and then sends that many line feeds. Read/write.

```
Setter: public void setBoolean( "emulateFormFeed", boolean enable)
Getter: public boolean getBoolean( "emulateFormFeed" )
```

## enableHotspots

Property. Indicates whether hotspots are enabled or not. This property is linked to the Enable hotspots check box in the Hotspot Setup dialog box. Read/write.

```
Setter: public void setBoolean("enableHotspots", boolean enable)
Getter: public boolean getBoolean("enableHotspots")
```

## exit()

Method. Closes the Reflection for the Web applet.

```
public void exit()
```

## exportKeymap (OutputStream)

Method. Saves the terminal session's current keymap, independent of any preference or configuration information. Keymap files can be imported using the importKeymap API. This method is for use only from Java.

```
public boolean exportKeymap(OutputStream outStream)
```

## fileTransferConfigure

Dialog box constant. When used with the showDialog method, opens the File Transfer Setup dialog box.

```
public void showDialog( "fileTransferConfigure")
```

## findField (int, int, int)

Method. Finds a field with the specified attributes. You can then get the location, length, and contents of the found field using other API methods.

```
public boolean findField(int inRow,
                         int inCol,
                         int direction,
                         int inAttribute )
```

## findText (String, int, int, boolean)

Method. Finds a text string in the terminal window. For this method to return True, the text must already be present on the display. If your goal is to find text that's either already on the display or will appear on the display from datacomm, use the waitForDisplayString method instead.

```
public boolean findText(String inString,
                          int inRow,
                          int inCol,
                          boolean ignoreCase)
```

## fitHostPageX

Property. Reflection can mediate between the page width, as specified by the host, and the printer to assure that the printout will fit correctly on the page. Reflection does this by scaling. If the host does not define a page size, or if Override host page format is selected, the page width is derived from the Columns setting on the Page tab in the Print Setup dialog box.

```
Setter: public void setBoolean("fitHostPageX", boolean scaleWidth)
Getter: public boolean getBoolean("fitHostPageX")
```

## fitHostPageY

Property. Reflection can mediate between the page height, as specified by the host, and the printer to assure that the printout will fit correctly on the page. Reflection does this by scaling. If the host does not define a page size, or if Override host page format is selected, the page height is derived from the Rows setting on the Page tab in the Print Setup dialog box.

```
Setter: public void setBoolean("fitHostPageY", boolean scaleHeight)
Getter: public boolean getBoolean("fitHostPageY")
```

## foundFieldEndColumn

Property. Indicates the last column number of a field located using the `findField` method. Read only.

```
public int getInteger( "foundFieldEndColumn")
```

## foundFieldEndRow

Property. Indicates the last row number of a field located using the findField method. Read only.

```
public int getInteger("foundFieldEndRow")
```

## foundFieldLength

Property. Indicates the length of a field located using the findField method. Read only.

```
public int getInteger("foundFieldLength")
```

## foundFieldStartColumn

Property. Gets the first column number of a field located using the `findField` method. Read only.

```
public int getInteger("foundFieldStartColumn")
```

## foundFieldStartRow

Property. Gets the first row number of a field located using the `findField` method. Read only.

```
public int getInteger("foundFieldStartRow")
```

## foundTextStartColumn

Property. Gets the starting column number in which the text string located using the `findText` method was found. Read only.

```
public int getInteger("foundTextStartColumn")
```

## foundTextStartRow

Property. Gets the starting row number in which the text string located using the findText method was found. Read only.

```
public int getInteger("foundTextStartRow")
```

## ftpCd (String)

Method. Changes directories on the FTP server. This method is valid only after an FTP login is successful. If the FTP session is disconnected, then reconnected, the server directory must be set again. This method is useful when transferring multiple files using the `ftpSendFiles` and `ftpReceiveFiles` methods, which transfer files to and from the current local and server working directories.

```
public void ftpCd(String dir)
```

## ftpConfigure

Dialog box constant. When used with the showDialog method, opens the FTP Connection Setup dialog box for configuring connections to an FTP server.

```
public void showDialog("ftpConfigure")
```

## ftpDefaultFolders

Dialog box constant. When used with the `showDialog` method, opens the FTP Select Default Directory dialog box for configuring the default working directories.

```
public void showDialog("ftpDefaultFolders")
```

## ftpDisconnect ()

Method. Disconnects the current FTP session.

```
public boolean ftpDisconnect()
```

## ftpGetLastServerResponse()

Method. Returns the text message from the FTP server received as a result from the last user command.

```
public String ftpGetLastServerResponse()
```

## ftpLCd (String)

Method. Changes the current working directory on the local machine. This method is valid only after an FTP login is successful. If the FTP session is disconnected, then reconnected, the local directory must be set again. This method is useful when transferring multiple files using the `ftpSendFiles` and `ftpReceiveFiles` methods, which transfer multiple files to and from the current local and server working directories.

```
public void ftpLCd(String dir)
```

## ftpLogin (String, String, String, boolean)

Method. Logs into an FTP or SFTP server in preparation for transferring files. This method supports both FTP and SFTP transfers, as well as FTP transfers through the Reflection security proxy server.

```
public boolean ftpLogin( String HostName,
                         String userName,
                         String passWord,
                         String acct,
                         boolean usePASV )
```

## ftpOptions

Dialog box constant. When used with the showDialog method, opens the FTP Options dialog box window for configuring FTP options.

```
public void showDialog("ftpOptions")
```

## ftpProtocol

Property. Specifies the protocol to use for FTP file transfers. This property is linked to the Connection type drop-down menu in the Connection Setup dialog of the FTP client. Typically, you do not need to set this property unless you want to perform SFTP file transfers; the default value is for FTP transfers. Read/write.

```
Setter: public void setString("ftpProtocol", String protocol)
Getter: public String getString("ftpProtocol")
```

## ftpPwd()

Method. Returns the current working directory on the FTP server. This method is useful when you are sending and receiving multiple files.

```
public String ftpPwd()
```

## ftpReceiveFile (String, String, Int)

Method. Receives an FTP file.

```
public boolean ftpReceiveFile( String remoteFile,
                               String localFile,
                               int transferMethod )
```

## ftpReceiveFiles (String, int)

Method. Receives multiple files using FTP.

```
public boolean ftpReceiveFiles( String remoteFiles,
                                int transferMethod )
```

## ftpSendFile (String, string, boolean, int)

Method. Sends a file using FTP.

```
public boolean ftpSendFile( String remoteFile,
                            String localFile,
                            boolean mkDirectories,
                            int transferMethod )
```

## ftpSendFiles (String, int)

Method. Sends multiple files using FTP.

```
public boolean ftpSendFiles(String localFiles, int transferMethod)
```

## ftpUI

Dialog box constant. When used with the `showDialog` method, opens the FTP main window for performing file transfers between the desktop computer and an FTP server.

```
public void showDialog("ftpUI")
```

## getAPI()

Method. Gets a reference to the API object on which all other API methods are invoked.

## getBoolean(String)

Property accessor method. Gets the value of a boolean property. This method is available in all session types, but the valid properties vary according to emulation.

```
public boolean getBoolean( String propName )
```

`propName` The name of the property whose value you want to get from the list of properties below.

Properties that can use this method:

- autoPrintLineFeed
- emulateFormFeed
- enableHotspots
- fitHostPageX
- fitHostPageY
- showHotspots
- streamIsEncrypted

## getCoordinateBase()

Method. Gets the current screen coordinate numbering base, either 0-based or 1-based coordinates.

```
public int getCoordinateBase()
```

## getCursorColumn()

Method. Gets the terminal window display column in which the host cursor is currently located.

```
public int getCursorColumn()
```

## getCursorPosition()

Method. Gets the terminal window display row and column in which the host cursor is currently located.

```
public Dimension getCursorPosition()
```

## getCursorRow()

Method. Gets the terminal window display row in which the host cursor is currently located.

```
public int getCursorRow()
```

## getDisplayText (int, int,i nt, int, boolean, string)

Method. Gets text from the display. Form 1 of `getDisplayText` lets you specify a starting row and column, an ending row and column, and whether the text selection should be linear or rectangular (see the Notes section below for examples of these types of selections); the text retrieved is formatted with the line ending characters you specify separating each line of display text.

Form 2 of `getDisplayText` lets you specify a starting row and column and the number of characters to retrieve. If the number of characters remaining on the display line is less than the number of characters you request, only the characters remaining on the line are returned.

```
Form 1: public String getDisplayText( int inStartRow,
                                      int inStartCol,
                                      int inEndRow,
                                      int inEndCol,
                                      boolean isRect,
                                      String inLineEnd )

Form 2: public String getDisplayText( int inStartRow,
                                      int inStartCol,
                                      int numCharacters )
```

## getFieldText (int, int, int)

Method. Gets the contents of a host field from the display. You specify the row and column of the field whose text you want returned and the number of characters to retrieve. To use this method, first use the `findField` method to find a field on the display with specific attributes. After the desired field is

located, use the `foundFieldStartRow, foundFieldEndRow, foundFieldStartColumn, foundFieldEndColumn`, and `foundFieldLength` properties to determine the coordinates and length of the found field, and pass the appropriate values to the `getFieldText` method.

```
public String getFieldText(int inRow, int inCol, int inChars)
```

## getHostName()

Method. Gets the host name for the current session. This property is linked to the Host name or IP address box in the Session Setup dialog box.

```
public String getHostName()
```

## getHostStatusText (int, int)

Method. Gets text the host application has displayed on the status line. Returns a value only if the Status Line control in the Options Panel for the Terminal Setup dialog box is set to Host Writeable.

```
public String getHostStatusText(int inStartCol, int numCharacters)
```

## getHostURL()

Method. Gets the host URL for the current session, including the transport type, host name, and port. This lets you get all of the current host connection information with a single method. To get the individual values that comprise a host URL, use the `getHostName` and `getPort` methods, and the `transportType` property.

```
public String getHostURL()
```

## getInitialized()

Method. Indicates whether the Reflection terminal session applet has been fully initialized. You should not attempt to make other API calls until you're sure that the session is initialized, which you can determine with this method. Read only.

```
public int getInitialized()
```

## getInteger(String)

Property accessor method. Gets the value of an integer property. This method is available in all session types, but the valid properties vary according to emulation.

```
public int getInteger( String propName)
```

Properties that can use this method:

- foundFieldEndColumn
- foundFieldEndRow
- foundFieldLength
- foundFieldStartColumn
- foundFieldStartRow

Returns the value of the specified property.

## getPort()

Method. Gets the protocol port for the current session. This property is linked to the Port box in the Session Setup dialog box.

```
public int getPort()
```

## getPrintToFileFolder()

Method. Gets the name of the folder to which printer output is directed when printing to a file. Use the method `getPrintToFileName` to get the name of the file that is to receive the print output. This method is relevant only when the `printToFile` property is set to True.

```
public String getPrintToFileFolder()
```

## getPrintToFileName()

Method. Gets the name of the file in which printer output is saved when printing to a file. Use the method `getPrintToFileFolder` to get the name of the folder in which the output file is stored. This method is relevant only when the `printToFile` property is set to True.

```
public String getPrintToFileName()
```

## getString(String)

Property accessor method. Gets the value of a String property. This method is available in all session types, but the valid properties vary according to emulation.

```
public String getString( String propName)
```

Properties that can use this method:

- declans (load answerback message)
- deviceName
- printDBCSScale
- terminalModel
- transportType

Returns the string value of the specified property.

## ignoreUserTyping

Property. Determines whether keystrokes typed by the user into the terminal display should be processed as usual or should be ignored. During lengthy scripting operations, it may be desirable to have user keystrokes ignored so they do not interfere with the operation. After the scripting operation is complete, user typing can be re-enabled. When user typing is ignored, only keystrokes entered in the terminal display are ignored; accelerator keys used to trigger menu commands are still processed. Read/write.

```
Setter: public void setBoolean( "ignoreUserTyping", boolean ignore)
Getter: public boolean getBoolean("ignoreUserTyping")
```

## importKeymap (InputStream)

Method. Opens a keymap and applies it to the current session. A keymap file can be created with the exportKeymap API. This method is for use only from Java; see the Notes section below for more information.

```
public boolean importKeymap( InputStream inStream)
```

## indReceiveFile (String, string, boolean, boolean)

Method. Transfers a file from an IBM mainframe to the desktop computer, using the IND$FILE transfer protocol. Currently, only structured field IND$FILE transfers are supported.

```
public int indReceiveFile(String localFile, String hostFile, boolean isASCII,
boolean showStatus)
```

## indSendFile (String, string, boolean, boolean)

Method. Transfers a file from the desktop computer to an IBM mainframe, using the IND$FILE transfer protocol. Currently, only structured field IND$FILE transfers are supported.

```
public int indSendFile(String localFile, String hostFile, boolean isASCII, boolean
showStatus)
```

## isConnected()

Method. Indicates whether the session currently has a host connection.

```
public boolean isConnected()
```

## keyMapConfigure

Dialog box constant. When used with the showDialog method, opens the Keyboard Setup dialog box.

```
public void showDialog( "keyMapConfigure")
```

## keyPaletteOptions

Dialog box constant. This constant was removed starting with Reflection for the Web version 4.1, because the Terminal Keyboard Options dialog box was removed.

## loadJavaClass()

Method. Loads the specified Java class to run as an "attachment class." Java attachment classes are a feature of the Reflection Emulator Class Library (ECL) that allow Java code to attach to the currently running terminal session and perform automated tasks. This method is equivalent to the Load Java Class command in the Macro menu.

```
public void loadJavaClass( String className)
```

## mouseMapConfigure

Dialog box constant. When used with the showDialog method, opens the Mouse Setup dialog box.

```
public void showDialog( "mouseMapConfigure")
```

## playbackMacro (macroName)

Method. Plays back a previously recorded macro.

```
public boolean playbackMacro(String macroName)
```

## printDBCSScale

Property. By default, Reflection uses enough space for two single-byte columns to print one column of double-byte data. For example, if you've set up your page for 80 columns, you'll be able to print 40 columns of double-byte data. This parameter gives you the option of printing two double-byte columns in the same amount of space as three single-byte columns. So if you set up your page for 80 columns, you'll be able to print about 53 columns of double-byte data. Read/write.

```
Setter: public void setString("printDBCSScale", String ratio)
Getter: public String getString("printDBCSScale")
```

## printerConfigure

Dialog box constant. When used with the showDialog method, opens the Print Setup dialog box.

```
public void showDialog( "printerConfigure")
```

## printToFile

Property. Indicates whether printer output is to be directed to a printer or to a text file. This property is linked to the **Send output to printer** and **Send output to file** options in the Print Setup dialog box. By default, Send output to printer is selected and this property is False, and printer output is directed to a printer.

```
Setter: public void setBoolean("printToFile", boolean enable)
Getter: public boolean getBoolean("printToFile")
```

## requestDisplayFocus()

Method. Requests that the Reflection terminal window should receive the input focus. If you include buttons or other form elements on your web page to perform actions via the API, you can use this method to return input focus to the Reflection session so the user can type at the terminal cursor; otherwise, the input focus may remain on the form element, and a mouse click in the terminal window is required to give focus back to the Reflection session.

```
public void requestDisplayFocus()
```

## resetUserPreferences()

Method. Resets the session settings in the user preference file to the defaults established by the administrator. The user preference file contains the user's personal Reflection terminal session customizations and is stored on each user's computer. The `resetUserPreferences` method resets the preference file for the session by replacing the file with one containing no preference information. After the preference file is reset, the Reflection session must be restarted for the default session settings to take effect.

The specific settings that can be stored in the user preference file depend on the options that the administrator has selected for user preferences; see Configuring Reflection: User Preferences for more information about user preferences.

Preference files are stored on the user's computer in the reflectionweb sub folder under the user home folder. The user home varies according to operating system, browser, and virtual machine. To find the home folder for a given operating system/browser/virtual machine combination, look for USER_HOME in the Java console.

The name of the user preference file that gets reset is based on the value of the applet's prefsName parameter, and if that doesn't exist, on the name attribute in the <applet> tag of the web page, with a ".pref" extension added. For example, if the name attribute for the applet is IBM3270Applet, the name of the preferences file will be IBM3270Applet.pref. Avoid naming an applet using characters that are invalid for the desktop platform's file naming convention; for example, Windows operating systems do not allow file names to contain these characters: \ / : * ? " < > |.

```
public void resetUserPreferences() throws IOException
```

## saveUserPreferences()

Method. Saves the current session settings to the user preference file, a file contains the user's personal Reflection terminal session customizations and is stored on each user's computer.

The settings that can be stored depend on the options that the administrator has selected in Reflection's Set User Preference Rules dialog box (available from the Administration menu)--if no preference rules are configured, an empty user preference file is saved.

Preference files are stored on the user's computer in the reflectionweb sub folder under the user home folder. The user home varies according to operating system, browser, and virtual machine. To find the home folder for a given operating system/browser/virtual machine combination, look for USER_HOME in the Java console.

The name of the user preference file is based on the value of the applet's `prefsName` parameter, and if that doesn't exist, on the name attribute in the <applet> tag of the web page, with a ".pref" extension added. For example, if the name attribute for the applet is IBM3270Applet, the name of the preferences file will be IBM3270Applet.pref. Avoid naming an applet using characters that are invalid for the desktop platform's file naming convention; for example, Windows operating systems do not allow file names to contain these characters: \ / : * ? " < > |.

If the <applet> tag does not define a name attribute, a user preference file will not be saved.

To disable the loading of the user preference file, set the loadUserPrefs parameter in the <applet> tag of the Reflection web page to false.

```
public void saveUserPreferences() throws IOException
```

## screenPrint()

Method. Prints the contents of the terminal window display, using the options selected in the Print Setup dialog box.

```
public void screenPrint()
```

## sessionConfigure

Dialog box constant. When used with the showDialog method, opens the Session Setup dialog box.

```
public void showDialog( "sessionConfigure")
```

## setBoolean (String, boolean)

Property accessor method. Sets the value of a boolean property. This method is available in all session types, but the valid properties vary according to session type.

```
public void setBoolean( String propName, boolean value)
```

Value - The value to set the property to--either True or False.

Properties that can use this method:

- autoPrintLineFeed
- emulateFormFeed
- enableHotspots
- fitHostPageX
- fitHostPageY
- ignoreUserTyping
- showHotspots

## setCoordinateBase()

Method. Specifies whether methods that perform operations using screen coordinates should use 0-based numbering or 1-based numbering.

```
public void setCoordinateBase( int base)
```

## setCursorPosition (int,int)

Method. Moves the host cursor to the specified position.

```
public void setCursorPosition(int inRow, int inCol)
```

## setHostURL (String)

Method. Sets the host URL for the current session, including the transport, host name, and port. If you use this method to change the host URL after a connection has already been established, the new value does not take effect until you disconnect then reconnect.

```
public void setHostURL(String hostURL) throws MalformedURLException
```

## setInteger (String, int)

Property accessor method. Sets the value of an integer property. This method is available in all session types, but the valid properties vary according to the sessions type.

```
public void setInteger( String propName, int value)
```

## setPrintToFileFolder (String)

Method. Sets the name of the folder to which printer output is directed when the output is sent to a file. Use the method `setPrintToFileName` to set the name of the file that should receive the print output. This method is relevant only when the `printToFile` property is set to True.

```
public void setPrintToFileFolder(String folderName)
```

## setPrintToFileName (String)

Method. Sets the name of the file to which printer output is to be sent. Use the method `setPrintToFileFolder` to set the name of the folder in which the output file is to be stored. This method is relevant only when the `printToFile` property is set to True

```
public void setPrintToFileName(String fileName)
```

## setString (String, string)

Property accessor method. Sets the value of a String property. This method is available in all session types, but the valid properties vary according to session type.

Properties that can use this method:

- declans
- deviceName
- printDBCSScale
- terminalModel

```
public void setString(String propName,String value)
```

## sftpAllowUnknownHost

Property. Indicates how SFTP connections should handle unknown hosts. This property is linked to the Allow unknown hosts options in the Secure Shell Client Settings dialog box for FTP. The default value of 2, "ask," causes Reflection to prompt when encountering a host whose authenticity cannot be established. If you are automating a logon procedure and do not want the user to receive any prompts during the logon, set the value of this property to 0, for "always allow." Read/write.

```
Setter: public void setInteger( "sftpAllowUnknownHost",int value)
Getter: public int getInteger("sftpAllowUnknownHost")
```

## showDialog (String)

Method. Opens the specified dialog box. This method is available in all seesion types, but the valid dialog box names vary according to emulation

```
public void showDialog(String dialog)
```

These dialog boxes can be opened using this method:

- aboutBoxDialog
- buttonPaletteConfigure
- colorConfigure
- fileTransferConfigure
- fileTransferUI
- ftpConfigure
- ftpDefaultFolders
- ftpOptions
- ftpUI
- keyMapConfigure
- mouseMapConfigure
- printerConfigure
- sessionConfigure
- terminalConfigure

The `showDialog` method opens Reflection dialog boxes asynchronously; it does not wait for the user to close the dialog box before continuing. This means that you can issue other Reflection method calls and perform other script actions while a dialog box is open. In addition, this method does not return any result codes to indicate how the user closed the dialog box (for example, by clicking OK or Cancel).

## showHotspots

Property. Indicates whether hotspots are set to show on the terminal window display or are set to be hidden. This property is linked to the Show hotspots check box in the Hotspot Setup dialog box. Read/write.

```
Setter: public void setBoolean( "showHotspots", boolean visible)
Getter: public boolean getBoolean( "showHotspots")
```

## sshAllowUnknownHost

Property. Indicates how SSH connections should handle unknown hosts. This property is linked to the Allow unknown hosts options in the Secure Shell Client Settings dialog box. The default value of 2, "ask," causes Reflection to prompt when encountering a host whose authenticity cannot be established. If you are automating a logon procedure and do not want the user to receive any prompts during the logon, set the value of this property to 0, for "always allow." Read/write.

```
Setter: public void setInteger( "sshAllowUnknownHost", int value)
Getter: public int getInteger( "sshAllowUnknownHost")
```

## streamIsEncrypted

Property. Indicates whether the terminal session has an encrypted connection. Read only.

```
public boolean getBoolean( "streamIsEncrypted")
```

## terminalConfigure

Dialog box constant. When used with the showDialog method, opens the Terminal Setup dialog box.

```
public void showDialog("terminalConfigure")
```

## terminalModel

Property. Indicates the terminal model of the current terminal session. This property is linked to the Terminal model list in the Session Setup dialog box. Read/write.

```
Setter: public void setString( "terminalModel", String model)
Getter: public String getString("terminalModel")
```

## transmitString (String)

Method. Transmits a string of characters to the host as if the user had typed the string at the current cursor position. (Compare this with the display method).

```
public void transmitString( String inString)
```

## transportTerminalKey

Method. Transmits a single non-character keystroke to the host as if the user had pressed that key on the host keyboard

```
public void transmitTerminalKey( int inKey)
```

## transportType

Property. Gets the transport type of the current session. This property is linked to the Type list in the Transport options section of the Session Setup dialog box. Read only.

```
public String getString( "transportType")
```

## waitForCursorEntered (int, int, long)

Method. Waits for the cursor to enter a specific row and column location in the terminal window.

```
public boolean waitForCursorEntered( int inRow, int inCol, long timeout)
```

## waitForCursorLeft (int, int, long)

Method. Waits for the cursor to leave a specific row and column location in the terminal window.

```
public boolean waitForCursorLeft(int inRow, int inCol, long timeout)
```

## waitForDisplayString (String, long)

Method. Waits for a string of characters to appear in the terminal window, and optionally, at a specific row and column. For HP emulators, if you want to wait for a string to appear in the datacomm stream (which may also include non-printing characters such as control characters), use the `waitForString` or `waitForHostPrompt` methods instead. For VT emulators, if you want to wait for a string to appear in the datacomm stream (which may also include non-printing characters such as control characters), use the `waitForString` method instead.

```
Form 1: public boolean waitForDisplayString(String inString, long timeout)
Form 2: public boolean waitForDisplayString(String inString, int inRow,   int
inCol, long timeout)
```

## waitForDisplayStrings (Vector, long)

Method. Waits for one or more strings of characters to appear in the terminal window. In HP and VT sessions, `waitForDisplayStrings` returns true if one of the strings appears in the visible terminal window or in the text that has scrolled off the screen into the scrollback buffer. If you are using one of these emulators and want to wait for strings to appear in the datacomm stream (which may include non-printing characters such as control characters), use the `waitForStrings` method. For HP emulators, you can also use the `waitForHostPrompt` method, which waits for the specified string plus the host prompt character.

```
public int waitForDisplayStrings(Vector inStrings,  long timeout)
```

## waitForHostPrompt (String, long)

Method. Waits until the specified string plus the host prompt character is received in the incoming datastream. The HP host prompt character (typically a DC1 character that follows the MPE colon prompt) indicates that the host is ready to receive input. If no host prompt character is being used or the host prompt is disabled (with Inhibit handshake and Inhibit DC2 in the Advanced HP Terminal Items dialog box of the Terminal Setup dialog box), this method waits for only the specified string, acting exactly like the `waitForString` method. You can configure which host prompt character to wait for, and whether the host prompt character is used, with the Host prompt and Use host prompt options in the Advanced HP Terminal Items dialog box of the Terminal Setup dialog box.

```
public boolean waitForHostPrompt(String inString,long timeout)
```

## waitForIncomingData (long)

Method. Waits until any data is received from the host. Although this method is available in any type of emulation, it is used primarily with VT and HP sessions.

```
public boolean waitForIncomingData(long timeout)
```

## waitForKeyboardLock (long, long)

Method. Waits a specified time for the keyboard to be locked for a specified duration.

```
public boolean waitForKeyboardLock(long duration, long timeout)
```

## waitForKeyboardUnlock (long, long)

Method. Waits a specified time for the keyboard to be unlocked for a specified duration.

```
public boolean waitForKeyboardUnlock(long duration, long timeout)
```

## waitForString (String, long)

Method. Waits for a string of characters to appear in the incoming datastream. In contrast to the `waitForDisplayString` method, which waits for characters to appear in the terminal window, this method allows you to wait for characters that may not appear in the terminal window, such as escape sequences and other non-printing characters. For IBM sessions, use `waitForDisplayString` instead of this method.

```
public boolean waitForString(String inString,long timeout)
```

## waitForDisplayString (String, boolean)

Method. Waits for one or more character strings to appear in the incoming datastream. In contrast to the `waitForDisplayStrings` method, which waits for characters to appear in the terminal window, this method allows you to wait for characters that may not appear in the terminal window, such as escape sequences and other non-printing characters. For IBM sessions, use `waitForDisplayString`.

```
public int waitForStrings(Vector inStrings,long timeout)
```

## waitWhileDisplayString (String, int, int, long, boolean)

Method. Waits for a string to leave the display. If the string is not already on the display when this method is called, the method returns True immediately (that is, the string has "left" the display because it was never there to begin with). This method is typically used after determining that a desired string is in place on the display; for instance, as the result of calling the `waitForDisplayString` or `FindText` method.

```
Form 1: public boolean waitWhileDisplayString(String inString, boolean caseSens)
Form 2: public boolean waitWhileDisplayString(String inString, long timeout)
Form 3: public boolean waitWhileDisplayString(String inString,int inRow, nt
inCol,long timeout boolean caseSens)
```

## xfr400ReceiveFile (String, string, boolean)

Method. Transfers data from an AS/400 host to the desktop computer, using the data transfer feature.

```
public int xfr400ReceiveFile(String localFile,String hostFile, boolean showStatus)
```

## xfr400SendFile (String, string, boolean)

Method. Transfers data from the desktop computer to an AS/400 host, using the data transfer feature.

```
public int xfr400SendFile(String localFile,String hostFile, boolean showStatus)
```

# API Examples

The examples below show how to perform different types of tasks with the Reflection for the Web API. You can cut and paste the sample code directly from the example page into a text editor, and then save the document with a .html extension.

If you store your web pages in the deploy folder that's created when you install Reflection, the examples should work with the codebase attribute set to ./ex. If you store your pages in a different folder, you'll need to first change the codebase attribute in the `<applet>` tag of the examples before they will work. Also, in examples that use the `hostURL` parameter to specify the name of a host computer, you should change the value to a host name appropriate to your network.

---

**TIP:** If the sample code is not formatted correctly after pasting it into your text editor, try reselecting the sample code, making sure that you extend your selection to the blank line after the end of the sample. Some browsers, such as Internet Explorer, do not copy all of the code formatting unless an ending paragraph mark is also selected. You can ensure that the paragraph mark is included in the selection by including the blank line after the last line of code.

---

## Basic JavaScript Examples



### Using JavaScript in the same HTML file as Reflection

In this example, JavaScript is used to interact with an IBM 3270 applet embedded in the browser window, where both the applet and the JavaScript code are in the same HTML file. Comments in the JavaScript explain the action. Also see the example of using JavaScript in a different HTML file than Reflection.

To use this example, you must change the hostURL parameter in the <applet> tag below to a host appropriate for your network.

```
<html>
<head>
<title>API Sample: JavaScript in the Same HTML File as Reflection</title>
</head>
<body>

<!--
   The <applet> tag defines a simple Reflection applet for
   IBM 3270 emulation. To use this example, change the hostURL
   parameter to one appropriate for your network.
-->
<applet mayscript name="IBM3270Applet"
        codebase="./ex/"
        code="com.wrq.rweb.Launcher.class"
        width="800" height="500"
        archive="Launcher.jar">
    <param name="hostURL" value="tn3270://payroll">
    <param name="autoconnect" value="true">
    <param name="frame" value="false">
    <param name="launcher.sessions" value="IBM3270">
    <param name="preloadJSAPI" value="true">
</applet>
<!--
```

```
   <Form> buttons give the user access to the Color Setup and
   Button Palette Setup dialog boxes. The onClick event handler
   is used to invoke the API methods without requiring separate
   JavaScript functions. The Reflection applet is referenced
   on the page using the same name as defined above in the <applet> tag.
-->
<form>
<input type="button" Name="ColorSetup" Value="Color Setup"
       onClick='document.IBM3270Applet.getAPI("JSAPI","IBM3270").showDialog(
"colorConfigure" )'>
<input type="button" Name="ButtonPalette" Value="Button Palette Setup"
       onClick='document.IBM3270Applet.getAPI("JSAPI","IBM3270").showDialog(
"buttonPaletteConfigure" )'>
</form>
</body>
</html>
```

## Using JavaScript in a different HTML file than Reflection

In this example, JavaScript is used to interact with a Reflection terminal session applet embedded in a browser window frameset. The applet is in one HTML file, which is displayed in the lower frame of the frameset file, and the JavaScript code is in a different HTML file, which is displayed in the upper frame. Comments in the JavaScript explain the action. Also see the example of using JavaScript in the same HTML file as Reflection.

First, here's the HTML for the frameset file:

```
   <html>
   <frameset rows="50,*">
       <frame name="codeframe" src="codeframe.html">
       <frame name="appletframe" src="appletframe.html">
   </frameset>
   </html>
```

Next, here's the HTML for the codeframe.html file, which is the upper frame of the frameset and contains the JavaScript code:

```
   <html>
   <head>
   <title>Code Frame</title>
   <script language="JavaScript">
   <!--
         /*
          The variable "target" holds a reference to the terminal session
          applet in the lower frame. When the script first runs, the setTarget()
function
          executes to set the target variable.
       */
       var target;
       setTarget();

       /*
          This function sets the variable "target" to refer to the terminal
          session applet in the lower frame. The full window and document hierarchy
          that identifies the location of the applet is required. If the applet is
          not yet loaded and the target is null, a timeout is used to continually
          check for the applet again, waiting 1 second between tries.
       */
       function setTarget()
       {
```

```
            target = parent.appletframe.document.IBM3270Applet;
            if ( target == null )
            {
                setTimeout( "setTarget()", 1000 );
            }
        }

        /*
           This function opens Reflection's Color Setup dalog box
           when the user clicks the Color Setup button. The notation
           "target.showDialog( "colorConfigure" )" uses the target variable
           defined earlier, providing an easier way of specifying

"parent.appletframe.document.IBM3270Applet.getAPI("JSAPI","IBM3270").showDialog(
"colorConfigure" )".
        */
        function doColorSetup()
        {
            target.getAPI("JSAPI","IBM3270").showDialog( "colorConfigure" );
        }

        /*
          This function opens Reflection's Button Palette Setup dialog box when the
user clicks the Button Setup button. The notation
           "target.showDialog( "buttonPaletteConfigure" )" uses the target variable
           defined earlier, providing an easier way of specifying

"parent.appletframe.document.IBM3270Applet.getAPI("JSAPI","IBM3270").showDialog(
"buttonPaletteConfigure" )".
        */
        function doButtonSetup()
        {
            target.getAPI("JSAPI","IBM3270").showDialog( "buttonPaletteConfigure" );
        }
    // -->
    </script>
    </head>
    <body>

    <!--
       <Form> buttons give the user access to the Color Setup and
       Button Palette Setup dialog boxes. When the onClick event handler
       is invoked, one of the JavaScript functions defined above is
       called.
    -->
    <form>
    <input type="button" Name="ColorSetup" Value="Color Setup"
onClick="doColorSetup()">
    <input type="button" Name="ButtonSetup" Value="Button Setup"
onClick="doButtonSetup()">
    </form>
     </body>
    </html>
```

Lastly, here's the HTML for the appletframe.html file, which appears in the lower frame of the frameset and contains the Reflection terminal session applet--make sure to change the `hostURL` parameter in the `<applet>` tag to a host appropriate for your network:

```
<html>
<head>
<title>Applet Frame</title>
</head>
<body>

<!--
   The <applet> tag defines a simple Reflection applet for
   IBM 3270 emulation. To use this example, change the hostURL
   parameter to one appropriate for your network.
-->
<applet mayscript name="IBM3270Applet"
        codebase="./ex/"
        code="com.wrq.rweb.Launcher.class" width="800" height="500"
        archive="Launcher.jar">
    <param name="hostURL" value="tn3270://payroll">
    <param name="autoconnect" value="true">
    <param name="frame" value="false">
  <param name="launcher.sessions" value="IBM3270">
    <param name="preloadJSAPI" value="true">
</applet>

</body>
</html>
```

## Advanced JavaScriptExamples

- Using an onLoad handler to determine when Reflection is initialized
- Using notification to determine when Reflection is initialized
- Logging on to an IBM host
- Logging on to a VMS host
- Logging on to an HP3000 host
- Connecting a printer session
- Transferring files to an IBM mainframe using FTP
- Transferring files to an IBM mainframe using IND$FILE
- Logging out of the host when closing a session

### Using an onLoad handler to determine when Reflection is initialized

If you want to automate a task immediately after a Reflection terminal session loads, you must first ensure that the session is available and initialized before invoking other API methods. You can do this by using the JavaScript onLoad event handler in the web page's BODY tag, or by waiting for notification when the session is done with initialization. (See Timing issues when scripting Reflection for more information about script timing.)

The following JavaScript example shows how to use an `onLoad` handler to determine if a Reflection session is initialized. It also uses an error handler, so if Reflection is not initialized enough to access the `getInitialized` method, the function that performs the initialization check can retry the check after a short delay. Once Reflection is initialized, the Session Setup dialog box opens so the user can enter connection information and proceed with the session. In this example, both the Reflection terminal session applet and the JavaScript code are in the same HTML file.

See Using notification to determine when Reflection is initialized for an example of the notification method.

To use this example, you must change the `hostURL` parameter in the `<applet>` tag below to a host appropriate for your network.

```
<html>
<head>
<title>Example of an onLoad Handler to Determine Reflection Initialization</title>
<script language="JavaScript">
<!--
    /*
       Set an error handler for the window.
    */
    window.onerror = errHandler;

    /*
       Create a few variables to hold the target applet, the API, the
       initialization state, and a couple of counters.
    */
    var appletTarget = null;

    var inited = 0;
    var count = 0;
    var errcount = 0;

    /*
       The error handler is invoked if an attempt to get the applet
       initialization state fails (because Reflection is not yet available),
       or if the api is null and the getInitialized() method is invoked.
       The handler will retry the attempt to get the session's
       initialization state up to 10 more times, waiting 2 seconds between
       attempts. By returning a value of true from the error handler, no
       JavaScript error alert appears.
    */
    function errHandler( msg, url, line )
    {
        // To ensure that the error handler was invoked because the
        // initialization state is not available and not because some
        // other code failed, first check the "inited" variable before
        // setting the timeout.
        if ( inited <= 0 ) {
            if ( errcount < 10 ) {
                errcount++;
                setTimeout( "tryInit()", 2000 );
            }
            else {
                alert( "Unable to initialize Reflection." );
            }
        }
        return true;
    }

    /*
       This function first gets a reference to the Reflection applet on the
       page and stores it in "appletTarget". Then the function tries to get
       the Reflection API, and then the initialization state, using the getAPI()
       and getInitialized() methods. It then sets the "inited" variable
       to the return value of getInitialized(). If the "inited" variable
       is true, the function that opens the Session Setup dialog box is called.
      If the call to Reflection's getInitialized method fails (because the session
       or API isn't available yet), the error handler is invoked to repeat the
       attempt. If the session is available but not yet initialized, a timeout
```

```
       is set to retry this same function every 2 seconds, for up to 10
       attempts or until "inited" is true.
    */
    function tryInit()
    {
        appletTarget = document.IBM3270;
        api = appletTarget.getAPI("JSAPI","IBM3270");
        inited = api.getInitialized();
        if ( inited > 0 )
        {
            doSessionSetup();
        }
        else if ( inited < 0 )
        {
            alert( "Reflection initialization failed." );
        }
        else if ( count < 10 )
        {
            count++;
            setTimeout( "tryInit()", 2000 );
        }
        else
        {
            alert( "Unable to initialize Reflection." );
        }
    }

    /*
       This function calls Reflection's showDialog method to
       open the Session Setup dialog box. This function is invoked
       only after the terminal session has been initialized.
    */
    function doSessionSetup()
    {
       api.showDialog( "sessionConfigure" );
    }
// -->
</script>
</head>
<!--
   The onLoad event handler is used to execute the JavaScript code
   after the page is loaded.
-->
<body onLoad="tryInit()">
<!--
   This is the tag that launches a Reflection IBM 3270 applet.
   To use this example, change the hostURL parameter to one
   appropriate for your network.
```

```
-->
<applet mayscript name="IBM3270"
        codebase="./ex/"
        code="com.wrq.rweb.Launcher.class"
        width="666" height="448"
        archive="Launcher.jar">
    <param name="hostURL" value="tn3270://accounts">
    <param name="autoconnect" value="false">
    <param name="launcher.sessions" value="IBM3270">
    <param name="preloadJSAPI" value="true">
</applet>

</body>
</html>
```

## Using notification to determine when Reflection is initialized

If you want to automate a task immediately after a Reflection terminal session loads, you must first ensure that the session is available and initialized before invoking other API methods. You can do this using some built-in features of the API in which the Reflection session sends notification to a JavaScript function on your web page when session initialization is complete, or by using an onLoad event handler in the web page's BODY tag. (See Timing issues when scripting Reflection for more information about script timing.)

Allowing Reflection to send a notification message to your JavaScript is much simpler and is more reliable than using an onLoad event handler, but there are some situations, such as when using VBScript, where the event handler method is needed. See Using an onLoad Handler to Determine When Reflection Is Initialized for an example of the polling method from an onLoad event handler.

There are two ways to use notification to determine Reflection's initialization state:

- If you plan to use only those API methods available from the JavaScript API, add the parameter preloadJSAPI with a value of true to the Reflection applet tag. When this parameter is present, Reflection invokes a JavaScript function called jsapiInitialized when it has completed its initialization, passing the function a reference to the JavaScript API. Once your script receives this reference, you are guaranteed that the Reflection terminal session is initialized, and your JavaScript can proceed with any tasks it needs to perform. An example of this method is shown below.
- If you plan to use both JavaScript API and ECL API functions from JavaScript, you can add an applet parameter to the Reflection session that specifies a built-in ECL module to run after the session is finished with initialization. When the API module runs, it calls a predefined JavaScript function on your web page, and as above, once your script receives this reference, you are guaranteed that the session is initialized. See Accessing the ECL API from JavaScript for more information about accessing ECL API methods from JavaScript.

Unless you need access to ECL API functions, the first method described above is recommended, since it requires a smaller download from the Reflection Management Server than when using the JSEventNotifier module.

The following JavaScript example shows how to use the `jsapiInitialized` function to receive notification when Reflection initialization is complete. Once notification is received, the Session Setup dialog box opens so the user can enter connection information and proceed with the session. In this example, both the Reflection terminal session applet and the JavaScript code are in the same HTML file.

To use this example, you must change the `hostURL` parameter in the `<applet>` tag below to a host appropriate for your network.

```
<html>
<head>
<title>Example of using Notification to Determine Reflection Initialization</title>
<script language="JavaScript">
<!--

   /*
      Variable to hold the API reference when it is retrieved.
   */
   var api = null;

   /*
      jsapiInitialized is the name of the function that Reflection calls when
      Reflection initialization is complete. It passes a reference to the
      JSAPI, which is stored in a variable for subsequent use by other script
      functions.
   */
   function jsapiInitialized( inapi )
   {
      api = inapi;
      if ( api == null )
      {
         alert( "Reflection API initialization failed." );
      }
      else
      {
         doSessionSetup();
      }
   }

    /*
       This function calls Reflection's showDialog method to
       open the Session Setup dialog box. This function is invoked
       only after notification is received that the terminal
       session has been initialized.
    */
    function doSessionSetup()
    {
       api.showDialog( "sessionConfigure" );
    }
// -->
</script>
</head>
<!--
   The onLoad event handler is used to execute the JavaScript code
   after the page is loaded.
-->
<body>
<!--
   This is the tag that launches a Reflection IBM 3270 applet.
   To use this example, change the hostURL parameter to
   one appropriate for your network.
```

```
-->
<applet mayscript name="IBM3270"
        codebase="./ex/"
        code="com.wrq.rweb.Launcher.class"
        width="666" height="448"
        archive="Launcher.jar">
    <param name="hostURL" value="tn3270://accounts">
    <param name="autoconnect" value="false">
    <param name="launcher.sessions" value="IBM3270">
    <param name="preloadJSAPI" value="true">
</applet>

</body>
</html>
```

## Logging on to an IBM host

In this example, JavaScript is used to launch an IBM 3270 session and log on to the host computer.

The logon procedure for an AS/400 computer is quite similar to an IBM mainframe, and you could easily modify this script to use an IBM 5250 session.

Two <form> fields on the web page are used to get the user's ID and password before creating the Reflection session applet and performing the logon. The Reflection session is created in a new browser window that has only a status bar. This is done so the JavaScript writeln statements don't replace the contents of the current window and make the rest of the script unavailable. A related example shows a VBScript version of this script for Internet Explorer.

To use this example, you must change the `hostURL` parameter in the `<applet>` tag that's part of the `writeApplet()` function below to a host appropriate for your network.

```
<html>
<head>
<title>Sample Logon Script for IBM 3270 (JavaScript)</title>
<script language="JavaScript">
<!--
    var userid;
    var pword;
    var targetdoc;
    var targetwin;
    var appletTarget = null;
    var api = null;
    var count = 0;
    var errcount = 0;
    var inited = 0;

    /*
       Copy relevant symbolic constants from
       api_ibm3270_constants.js and modify as needed for VBScript.
    */
    var ANY_COLUMN = -1;
    var IBM3270_ENTER_KEY = 35;
    var IBM3270_TAB_KEY = 52;

    /*
       Set an error handler. The error handler function is at the end of
       the script.
    */
    window.onerror = errHandler;
```

```
/*
   This is the main function that performs the logon tasks.
*/
function handleLogonButton()
{
    // Validate the userid and password.
    if ( validateInputs() )
    {
        writeApplet();
    }
}

/*
   This function checks for a user ID and password, and if either is
   not present, displays an alert message.
*/
function validateInputs()
{
    userid = document.LogonForm.UserID.value;
    pword = document.LogonForm.Password.value;
    if ( userid == "" )
    {
        alert( "You must enter a User ID to log on!" );
        document.LogonForm.UserID.focus();
        return false;
    }
    if ( pword == "" )
    {
        alert( "You must enter a Password to log on!" );
        document.LogonForm.Password.focus();
        return false;
    }
    return true;
}

/*
   This function writes the applet tag to launch an IBM 3270 session in
   a new browser window called "IBM3270Applet."
*/
function writeApplet()
{
    targetwin = window.open( "", "IBM3270Applet",
                             "status=yes, width=650, height=550" );
    targetdoc = targetwin.document;

    // Use the "with (object)" statement to simplify the following lines.
    with ( targetdoc ) {
        writeln( '<html>' );
        writeln( '<body onLoad="window.opener.doLogon()">' );
        writeln( '<h1>Reflection for the Web -- IBM 3270</h1>' );
        writeln( '<applet mayscript name="IBM3270"' );
        writeln( '       codebase="./ex/"' );
        writeln( '       code="com.wrq.rweb.Launcher.class"' );
        writeln( '       width="600" height="400"' );
        writeln( '       archive="Launcher.jar">' );
        // Change the hostURL to a value appropriate for your network.
        writeln( '   <param name="hostURL" value="tn3270://payroll">' );
        writeln( '   <param name="autoconnect" value="false">' );
        writeln( '   <param name="frame" value="false">' );
        writeln( '   <param name="launcher.sessions" value="IBM3270">' );
```

```
        writeln( '   <param name="preloadJSAPI" value="true">' );
        writeln( '</applet>' );
        writeln( '<form>' );
        writeln( '<input type="button" ' );
        writeln( '       name="Close" value="Close Window" ' );
        writeln( '       onClick="window.close()"></form>' );
        writeln( '</body>' );
        writeln( '</html>' );
        close();
    }
}

/*
   This function is called by the applet window's onLoad event
   handler after the new child window has been loaded. This function
   is the main entry point for the rest of the logon processing. It
   works similar to the tryInit() function in the example
   Using an onLoad Handler to Determine When Reflection Is Initialized.
   Once Reflection is initialized, the logon procedure continues.
*/
function doLogon()
{
    appletTarget = targetdoc.IBM3270;
    api = appletTarget.getAPI("JSAPI","IBM3270");
    if ( api != null )
    {
        inited = api.getInitialized();
        if ( inited > 0 )
        {
            doConnect();
        }
        else if ( inited < 0 )
        {
            alert( "Reflection initialization failed." );
        }
    }
    else
    {
        if ( count < 10 )
        {
            count++;
            setTimeout( "doLogon()", 1000 );
        }
        else
        {
            alert( "Unable to initialize Reflection." );
        }
    }
}

/*
   This function starts the actual logon tasks. It determines if the session
   is connected, and if so, calls the function that finds the userid and
   password prompts on the display and transmits the user's data.
*/
function doConnect()
{
    // If there's no connection, try to connect.
    if ( !api.isConnected() )
    {
```

```
    targetwin.status = "Connecting...";
    api.connect();
}
else
{
    alert( "You're already connected!" );
    return;
}

// Use the status line to display an informational message, and then
// wait 5 seconds for the USERID prompt to appear on the display.
// The coordinates in which to start looking are any column in
// row 19. The API uses 0-based coordinates, in contrast to
// the 1-based coordinates that are used for the status line.
targetwin.status = "Finding USERID prompt.";
if ( !api.waitForDisplayString( "USERID", 19,
                        ANY_COLUMN, 5000 ) )
{
    targetwin.alert( "Unable to find USERID prompt. The " +
           "script will stop, but you may continue manually." );
    return;
}

// Wait 5 seconds for the cursor to appear in row 19, column 16.
// This is the location for the user's ID.
targetwin.status = "Waiting for cursor to enter USERID field.";
if ( !api.waitForCursorEntered( 19, 16, 5000 ) )
{
    targetwin.alert( "Cursor was not found in the USERID field. " +
           "The script will stop, but you can try completing the " +
           "logon manually." );
    return;
}

// userid field was found, and cursor is in place, so transmit the
// user's ID, followed by the Tab key to move to the next field.
api.transmitString( userid );
api.transmitTerminalKey( IBM3270_TAB_KEY );

// Wait for the cursor to enter the password field.
targetwin.status = "Waiting for the cursor to enter the Password field.";
if ( !api.waitForCursorEntered( 20, 16, 5000 ) )
{
    targetwin.alert( "Cursor was not found in the Password field. " +
             "The script will stop, but you can try completing " +
             "the logon manually." );
    return;
}

// Cursor is in Password field, so transmit the user's password,
// followed by the Enter key.
api.transmitString( pword );
api.transmitTerminalKey( IBM3270_ENTER_KEY );

// Wait 5 seconds for the "Ready" message to appear on the display.
// If the "Ready" message is not found, display an alert message.
targetwin.status = "Waiting for Ready prompt.";
if ( !api.waitForDisplayString( "Ready;", 5000 ) )
{
    targetwin.alert( "Ready signal not found. The session may not " +
```

```
                            "be ready for input." );
            targetwin.status = "";
            return;
        }

        // Done with logon, so display final status message and set
        // focus to the terminal display.
        targetwin.status = "Ready to proceed.";
        api.requestDisplayFocus();
    }

    /*
       This is the error handler function that will catch script errors.
       The error we're most interested in may occur when trying to
       get Reflection's initialized state in the doLogon() function.
    */
    function errHandler( msg, url, line )
    {
        if ( inited <= 0 ) {
            if ( errcount < 10 ) {
                errcount++;
                setTimeout( "doLogon()", 2000 );
            }
            else {
                alert( "Unable to initialize Reflection." );
            }
        }
        return true;
    }
// -->
</script>
</head>
<body>
<h1>Reflection for the Web -- IBM 3270 Logon</h1>
<!--
   Use <form> items to accept the User ID and Password, and to create
   a Logon button.
-->
<form name="LogonForm">
<p>User ID: <input type="text" name="UserID" default size="25"></p>
<p>Password: <input type="password" name="Password" size="25"></p>
<p><input type="button" Name="Logon" value="Logon"
onClick="handleLogonButton()"></p>
</form>
</body>
</html>
```

## Logging on to a VMS host

In this example, a VT terminal session is embedded in the browser window, and JavaScript is used to log on to the host computer. Two <form> fields on the web page are used to get the user's name and password before performing the logon. A related example shows a VBScript version of this script for Internet Explorer.

To use this example, you must change the `hostURL` parameter in the `<applet>` tag below to a host appropriate for your network.

```
<html>
<head>
<title>Sample Logon Script for VT (JavaScript)</title>
<script language="JavaScript">
<!--
    var uname;
    var pword;
    var api = null;

    /*
       The jsapiInitialized function is called by Reflection
       when it has finished its initialization. Reflection
       passes the function a reference to the JavaScript API.
    */
    function jsapiInitialized( jsapi )
    {
       api = jsapi;
    }

    /*
       This is the main function that performs the logon tasks.
    */
    function handleLogonButton()
    {
        // Validate the username and password.
        if ( validateInputs() )
        {
            doConnect();
        }
    }

    /*
       This function checks for a username and password, and if either is
       not present, displays an alert message.
    */
    function validateInputs()
    {
        uname = document.LogonForm.Username.value;
        pword = document.LogonForm.Password.value;
        if ( uname == "" )
        {
            alert( "You must enter a Username to log on!" );
            document.LogonForm.Username.focus();
            return false;
        }
        if ( pword == "" )
        {
            alert( "You must enter a Password to log on!" );
            document.LogonForm.Password.focus();
            return false;
        }
        return true;
    }

    /*
       This function starts the actual logon tasks. It determines if the session
       is connected, and if so, calls the function that finds the Username and
       Password prompts on the display and transmits the user's data.
    */
    function doConnect()
```

```
    {
        // If there's no connection, try to connect.
        if ( !api.isConnected() )
        {
            self.status = "Connecting...";
            api.connect();
        }
        else
        {
            alert( "You're already connected!" );
            return;
        }

        // Wait 5 seconds for the Username prompt.
        self.status = "Waiting for Username prompt.";
        if ( !api.waitForString( "Username:", 5000 ) )
        {
            alert( "Unable to find Username prompt. The script will stop, " +
                   "but you may continue manually." );
            return;
        }

        // Username prompt was found, so transmit the user's username,
        // followed by a carriage return. The octal representation of the
        // carriage return character--which is decimal 13--is used. You
        // can also use the carriage return symbol \r or the hex value
        // \x0D. You CANNOT use the decimal value 13, because this will be
        // converted to a string and appended to the username.
        api.transmitString( uname + "\015" );

        // Wait 5 seconds for the Password prompt.
        self.status = "Waiting for Password prompt.";
        if ( !api.waitForString( "Password:", 5000 ) )
        {
             alert( "Unable to find Password prompt. The script will stop, " +
                   "but you can try completing the logon manually." );
            return;
        }

        // Password prompt was found, so transmit the user's password,
        // followed by a carriage return. As above, the octal
        // representation of the carriage return character is used.
        api.transmitString( pword + "\015" );

        // Done with logon, so display final status message, and set
        // focus to the terminal display.
        self.status = "Ready to proceed.";
        api.requestDisplayFocus();
    }
// -->
</script>
</head>
<body>
<h1>Reflection for the Web -- VT Logon</h1>
<p>
<!--
   Create a basic VT applet on the page, but don't connect until
   the user fills in the username and password information.
   To use this example, change the hostURL parameter to one
   appropriate for your network.
```

```
-->
<applet mayscript name="VT"
        codebase="./ex/"
        code="com.wrq.rweb.Launcher.class"
        width="600" height="400"
        archive="Launcher.jar">
    <param name="hostURL" value="telnet://accounts">
    <param name="autoconnect" value="false">
    <param name="frame" value="false">
    <param name="launcher.sessions" value="VT">
    <param name="preloadJSAPI" value="true">
</applet>
</p>
<!--
   Use <form> items to accept the Username and Password, and to create
   a Logon button.
-->
<form name="LogonForm">
<p>Username: <input type="text" name="Username" default size="25"></p>
<p>Password: <input type="password" name="Password" size="25"></p>
<p><input type="button" name="Logon" value="Logon"
onClick="handleLogonButton()"></p>
</form>
</body>
</html>
```

## Logging on to an HP3000 host

In this example, an HP terminal session applet is embedded in the browser window, and JavaScript is used to log on to the host computer. Four <form> fields on the web page are used to get the user's name, group, account, and password before performing the logon.

To use this example, you must change the `hostURL` parameter in the `<applet>` tag below to a host appropriate for your network.

```
<html>
<head>
<title>Sample Logon Script for HP (JavaScript)</title>
<script language="JavaScript">
<!--
    var uname, group, account, pword;
    var api = null;

    /*
       The jsapiInitialized function is called by Reflection
       when it has finished its initialization. Reflection
       passes the function a reference to the JavaScript API.
    */
    function jsapiInitialized( jsapi )
    {
       api = jsapi;
    }

    /*
       This is the main function that performs the logon tasks.
    */
    function handleLogonButton()
    {
        // Validate the username and password.
        if ( validateInputs() )
```

```
    {
        doConnect();
    }
}

/*
   This function checks for a group, account, and password, and
   if any of these is not present, displays an alert message.
*/
function validateInputs()
{
    uname = document.LogonForm.Username.value;
    pword = document.LogonForm.Password.value;
    group = document.LogonForm.Group.value;
    account = document.LogonForm.Account.value;

    // For the host in this example, a username is not required,
    // but a group and account are.
    if ( group == "" )
    {
        alert( "You must enter a Group to log on!" );
        document.LogonForm.Group.focus();
        return false;
    }
    if ( account == "" )
    {
        alert( "You must enter an Account to log on!" );
        document.LogonForm.Account.focus();
        return false;
    }

    if ( pword == "" )
    {
        alert( "You must enter a Password to log on!" );
        document.LogonForm.Password.focus();
        return false;
    }
    return true;
}

/*
   This function starts the actual logon tasks. It determines if the session
   is connected, and if so, calls the function that finds the Username and
   Password prompts on the display and transmits the user's data.
*/
function doConnect()
{
    // If there's no connection, try to connect.
    if ( !api.isConnected() )
    {
        self.status = "Connecting...";
        api.connect();
    }
    else
    {
        alert( "You're already connected!" );
        return;
    }

    // Wait 5 seconds for the MPE XL prompt to appear in the datacomm stream.
```

```
        // By using the HP-specific waitForHostPrompt method, the wait will be
        // for the string "MPE XL" plus the DC1 host prompt character.
        self.status = "Waiting for host prompt.";
        if ( !api.waitForHostPrompt( "MPE XL:", 5000 ) )
        {
            alert( "Unable to find MPE XL prompt. The script will stop, " +
                   "but you may continue manually." );
            return;
        }

        // MPE XL prompt was found, so transmit the username, group, and
        // account, followed by a carriage return. The octal representation
        // of the carriage return character--which is decimal 13--is used. You
        // can also use the carriage return symbol \r or the hex value
        // \x0D. You CANNOT use the decimal value 13, because this will be
        // converted to a string and appended to the username.
        var helloStr = "hello ";
        if ( uname != "" )
        {
            helloStr += uname + ",";
        }
        helloStr += group + "." + account;
        api.transmitString( helloStr + "\015" );

        // Wait 5 seconds for the Password prompt.
        self.status = "Waiting for Password prompt.";
        if ( !api.waitForHostPrompt( "Password:", 5000 ) )
        {
             alert( "Unable to find Password prompt. The script will stop, " +
                    "but you can try completing the logon manually." );
            return;
        }

        // Password prompt was found, so transmit the user's password,
        // followed by a carriage return. As above, the octal
        // representation of the carriage return character is used.
        api.transmitString( pword + "\015" );

        // Done with logon, so display final status message, and set
        // focus to the terminal display.
        self.status = "Ready to proceed.";
        api.requestDisplayFocus();
    }
// -->
</script>
</head>
<body>
<h1>Reflection for the Web -- HP Logon</h1>
<p>
<!--
   Create a basic HP applet on the page, but don't connect until
   the user fills in the username, group, account, and password
   information. A default group (DOCS) and account (WRITERS) are
   already in the text fields, so the minimum information the
   user needs to supply is the password, because the username is
   optional for the host in this example.

   To use this example, change the hostURL parameter to one
   appropriate for your network, and change the default group and
   account in the form fields.
```

```
-->
<applet mayscript name="HP"
        codebase="./ex/"
        code="com.wrq.rweb.Launcher.class"
        width="640" height="400"
        archive="Launcher.jar">
    <param name="hostURL" value="nsvt://techpubs">
    <param name="autoconnect" value="false">
    <param name="frame" value="false">
    <param name="launcher.sessions" value="HP">
    <param name="preloadJSAPI" value="true">
</applet>
</p>
<!--
   Use <form> items to accept the Username, Group, Account, and Password,
   and to create a Logon button.
-->
<form name="LogonForm">
<p>Username: <input type="text" name="Username" default size="25">
Group: <input type="text" name="Group" value="Docs" default size="25">
Account: <input type="text" name="Account" value="Writers" default size="25"></p>
<p>Password: <input type="password" name="Password" size="25">
<input type="button" name="logon" value="Logon" onClick="handleLogonButton()"></p>
</form>
</body>
</html>
```

## Connecting a printer session

In this example, an IBM 3270 Printer session is embedded in the browser window, and JavaScript is used to connect to the host computer. An HTML form list lets the user select the name of the printer device to which to connect, and two buttons let the user connect and disconnect the session.

This example can easily be modified to connect to an AS/400 Printer session. One difference is that it is not necessary to specify a device name when connecting to an AS/400 Printer session (although the AS/400 Printer applet can be configured to require a device name from the user by setting the `promptForDeviceName` parameter to True).

To use this example, you must change the `hostURL` parameter (including the port number) in the `<applet>` tag below to a host appropriate for your network. You must also change the device names in the <form> tag to appropriate device names.

```
<html>
<head>
<title>Sample Script for IBM 3270 Printer Emulation (JavaScript)</title>
<script language="JavaScript">
<!--
    var api = null;
    var deviceName = "";

   /*
       The jsapiInitialized function is called by Reflection
       when it has finished its initialization. Reflection
       passes the function a reference to the JavaScript API.
    */
    function jsapiInitialized( jsapi )
    {
       api = jsapi;
    }
```

```
/*
   This function handles a click in the Connect button.  If there's no
   connection already established, it calls the doConnect() function to
   make a connection attempt. If there is a connection already, an alert
   message is displayed.
*/
function handleConnectButton()
{
    if ( api.isConnected() )
    {
        alert( "You already have a connection to '" +
               api.getString( "deviceName" ) + "'. You must first\n" +
               "disconnect the current connection before " +
               "establishing a new\nconnection.");
        return;
    }
    doConnect();
}

/*
   This function handles a click in the Disconnect button. If there is
   no current connection, it does nothing. Otherwise, it disconnects
   the current session and displays a message in the form field.
*/
function handleDisconnectButton()
{
    if ( api.isConnected() )
    {
        api.disconnect();
        document.DevicePicker.ConnectedDevice.value = "<disconnected>";
    }
}

/*
   This function establishes the connection. It gets the name of the
   item selected in the form list, sets the Reflection session's device
   name to the same name, and then opens the connection. After the connection
   is opened, a timer is set to check the connection status after 2
   seconds (2000 milliseconds), using the setTimeout() method.
*/
function doConnect()
{
    var deviceList = document.DevicePicker.DeviceName;
    deviceName = deviceList.options[deviceList.selectedIndex].text;
    api.setString( "deviceName", deviceName );
    api.connect();
    setTimeout( "checkConnected()", 2000 );
}

/*
   This function checks whether the connection to the host using the
   selected device name was successful or not. If the connection failed,
   Reflection will display an alert message. This function will update
   the document's text field with either a success or failure message.
*/
function checkConnected()
{
    var isConnected = api.isConnected();
    var theDevice = document.DevicePicker.ConnectedDevice;
    if ( !isConnected )
```

```
        {
            theDevice.value = "<failed>";
        }
        else if ( isConnected )
        {
            theDevice.value = api.getString( "deviceName" );
        }
    }
// -->
</script>
</head>
<body>
<h1>Reflection for the Web -- IBM 3270 Printer: Session Picker</h1>
<p>
<!--
   Create a basic IBM 3270 Printer applet on the page, but don't
   connect until the user selects a device name and clicks the
   Connect button.

   To use this example, change the hostURL parameter to one
   appropriate for your network (including the port number), and
   change the names of the devices to devices for your host.
-->
<applet mayscript name="IBM3287"
        codebase="./ex/"
        code="com.wrq.rweb.Launcher.class"
        width="450" height="160"
        archive="Launcher.jar">
    <param name="hostURL" value="tn3270e://ibmprinter:5002">
    <param name="autoconnect" value="false">
    <param name="frame" value="false">
    <param name="launcher.sessions" value="IBM3287">
    <param name="preloadJSAPI" value="true">
</applet>
</p>
<!--
   Use <form> items to display a list from which the user
   can select a device to connect to and buttons to connect and
   disconnect the session.
-->
<form name="DevicePicker">
<p>
Device name: <select name="DeviceName">
<option selected>PRINTDEV1</option>
<option>PRINTDEV2</option>
<option>PRINTDEV3</option>
<option>PRINTDEV4</option>
</select>
<input type="button" name="Connect" value="Connect"
onClick="handleConnectButton()">
<input type="button" name="Disconnect" value="Disconnect"
                    onClick="handleDisconnectButton()">
</p>
<p>
Connection established to device: <input type="text"
            name="ConnectedDevice" value="<disconnected>" disabled>
</p>
</form>
</body>
</html>
```

## Transferring files to an IBM mainframe using FTP

In this example, an IBM 3270 session is in a separate browser window, and HTML form elements are used to provide controls for performing FTP file transfers between the desktop computer and the mainframe. The Transfer type radio buttons allow users to choose the method for sending or receiving files. Because the user is given access only to the basic menus, the administrator may want to preconfigure transfer options in the Connection Setup dialog box, then save those settings to a configuration file.

To use this example, you must change the `hostURL` parameter in the `<applet>` tag below to a host appropriate for your network.

```
<html>
<head>
<title>FTP API Example</title>
<script language="JavaScript">
<!--

    var api = null;

    /*
        Copy relevant symbolic constants from file server_root/webapps/rweb/admin/
en/html/advanced/api_constants/api_all_constants.js
    */
    var FTP_ASCII      = 0;
    var FTP_BINARY     = 1;
    var FTP_SMART      = 2;

    /*
        Returns the JSAPI for the terminal session.
    */
    function getJSAPI()
    {
       if ( api == null )
       {
         if ( document.ibm3270 != null )
         {
            api = document.ibm3270.getAPI( "JSAPI", "IBM3270" );
         }
       }
       return api;
    }

    /*
        This function performs the login to the FTP server. It first checks for
        the presence of a hostname, username, and password, and displays an alert
        message if any of these entries is missing. An alert message displays
        the status of the login attempt.
    */
    function doftpLogin()
    {
        var hostname = document.ftp.hostname.value;
        var username = document.ftp.username.value;
        var password = document.ftp.password.value;
        if ( hostname == "" || username == "" || password == "" )
        {
            alert( "You must enter a hostname, username, and " +
                   "password to login. One of these entries is missing." );
            return;
        }
```

```
    var result = getJSAPI().ftpLogin(hostname, username, password, null, true);
    if ( result )
        alert( "FTP login was successful." );
    else
        alert( "FTP login failed." );
}

/*
    This function disconnects from the FTP server. An alert message
    displays the status of the disconnect attempt.
 */
function doftpDisconnect()
{
    var result = getJSAPI().ftpDisconnect();
    if ( result )
        alert( "FTP disconnect was successful." );
    else
        alert( "FTP disconnect failed." );
}

/*
    This function sends a file to the FTP server. It does not perform any
    validity checking of the file names other than ensuring that there
    are entries in the local and remote file name fields. The transfer
    is performed as either ASCII, binary, or smart transfer, depending
    on the state of the Transfer type radio buttons, and server directories
    are created automatically if the "Automatically create directory" check
    box is selected. An alert message displays the status of the transfer.
 */
function doftpSendFile()
{
    var remotefile = document.ftp.remoteFile.value;
    var localfile = document.ftp.localFile.value;
    if ( !validateXferInputs(remotefile, localfile) )
        return;
    var makedir = document.ftp.makeDirectory.checked ? true : false;
    var xfertype = FTP_ASCII;   // default to ASCII
    if ( document.ftp.AsciiBinary[1].checked )
        xfertype = FTP_BINARY;   // binary
    else if ( document.ftp.AsciiBinary[2].checked )
        xfertype = FTP_SMART;   // smart transfer
    var result = getJSAPI().ftpSendFile(remotefile,localfile,makedir,xfertype);
    if ( result )
        alert( "FTP transfer to host was successful." );
    else
        alert( "FTP transfer to host failed." );
}

/*
    This function transfers a file from the FTP server to the local machine.
    No validity checking of file names is performed other than ensuring that
    there are entries in the local and remote file name fields. The transfer
    is performed as either ASCII, Binary or smart transfer, depending on the
    state of the Transfer type radio buttons. An alert message displays the
    status of the transfer.
 */
function doftpReceiveFile()
{
    var remotefile = document.ftp.remoteFile.value;
    var localfile = document.ftp.localFile.value;
```

```
        if ( !validateXferInputs(remotefile, localfile) )
            return;
        var xfertype = FTP_ASCII;   // default to ASCII
        if ( document.ftp.AsciiBinary[1].checked )
            xfertype = FTP_BINARY;   // binary
        else if ( document.ftp.AsciiBinary[2].checked )
            xfertype = FTP_SMART;   // smart transfer
        var result = getJSAPI().ftpReceiveFile(remotefile,localfile,xfertype);
        if ( result )
            alert( "FTP transfer from host was successful." );
        else
            alert( "FTP transfer from host failed." );
    }

    /*
        This function validates the entries in the Local File and Remote File
        text fields. There must be entries in both of these fields before you
        can transfer files.
    */
    function validateXferInputs(remotefile, localfile)
    {
        if ( remotefile == "" || localfile == "" )
        {
            alert( "You must specify both a remote file name and a local file " +
                   "name before starting the transfer." );
            return false;
        }
        return true;
    }

    /*
        This function gets the last response from the FTP server. If one of the
        other functions fails to complete successfully, the last server response
        contains the specific message.
     */
    function doftpGetLastServerResponse()
    {
        var result = getJSAPI().ftpGetLastServerResponse();
        alert( "Last response from FTP server:\n" + result );
    }

//-->
</script>
</head>
<body>
<h1>Reflection for the Web: FTP API Example</h1>
<hr>
<!--
    Use form items to collect the information needed to perform the FTP API
    functions.
-->
<form name="ftp">
<table>
    <tr>
        <td><b>Connection:</b></td>
        <td> </td>
    </tr>
    <tr>
        <td>Host name:</td>
        <td><input type="text" value="" name="hostname"></td>
```

```
    </tr>
    <tr>
        <td>User name:</td>
        <td><input type="text" value="" name="username"></td>
    </tr>
    <tr>
        <td>Password:</td>
        <td><input type="password" value="" name="password"></td>
    </tr>
</table>
<p><input type="button" name="ftplogin" value="Log In" onClick="doftpLogin()">
<input type="button" name="ftpdisconnect" value="Disconnect"
onClick="doftpDisconnect()">
</p>
<table>
    <tr>
        <td><b>File to transfer:</b></td>
        <td> </td>
    </tr>
    <tr>
        <td>Local file:</td>
        <td><input type="text" value="" name="localFile"></td>
    </tr>
    <tr>
        <td>Remote file:</td>
        <td><input type="text" value="" name="remoteFile"></td>
    </tr>
    <tr>
        <td> </td>
        <td><input type="checkbox" value="makeDirectory"
            name="makeDirectory">Automatically create directory</td>
    </tr>
    <tr>
        <td>Transfer type:</td>
        <td><input type="radio" value="ascii" name="AsciiBinary" checked>ASCII
            <input type="radio" value="binary" name="AsciiBinary">Binary
            <input type="radio" value="smart" name="AsciiBinary">Smart</td>
    </tr>
</table>
<p>
<input type="button" name="ftpSendFile" value="Send File"
onClick="doftpSendFile()">
<input type="button" name="ftpReceiveFile" value="Receive File"
onClick="doftpReceiveFile()">
</p>
<p>
<b>Last server response:</b><br>
<input type="button" name="ftpGetLastServerResponse" value="Display"
onClick="doftpGetLastServerResponse()">
</p>
</form>
<p>
<!--
   Create a basic IBM 3270 applet in a separate frame, with Advanced user
   menus. To establish a connection, you need to change the hostURL
   parameter to one appropriate for your network.
```

```
-->
<applet mayscript name="ibm3270" codebase="./ex"
    code="com.wrq.rweb.Launcher.class"
    width=0 height=0 archive="Launcher.jar">
    <param name="frame" value="true">
    <param name="menuType" value="advanced">
    <param name="hostURL" value="tn3270://mymainframe:23">
    <param name="ftpenabled" value="true">
    <param name="launcher.sessions" value="IBM3270">
    <param name="preloadJSAPI" value="true">
</applet>
</p>
</body>
</html>
```

## Transferring files to an IBM mainframe using IND$FILE

In this example, an IBM 3270 session is embedded in the browser window, and HTML form elements are used to provide controls for performing IND$FILE file transfers between the desktop computer and the mainframe. A "file" type input field is used on the form; this provides both a text field and a Browse button for selecting the local file to send to the host. Because the user is not given access to any of the Reflection menus or dialog boxes in this example, the administrator may want to preconfigure all file transfer settings in the File Transfer Setup dialog box, then save those settings to a configuration file.

To use this example, you must change the `hostURL` parameter in the `<applet>` tag below to a host appropriate for your network.

```
<html>
<head>
<title>Example of Transferring Files to an IBM Mainframe Using IND$FILE</title>
<script language="JavaScript">
<!--
    /*
       This function handles a click in either the Send File or Receive
       File button. It first ensures that Reflection is available and
       initialized, and then it gets the values of the <form> elements into an
       array, determines the direction of the transfer, and transfers the
       specified file.
    */
    function transferFile( direction )
    {
       // Get the Reflection session applet and make sure it is initialized.
       var rw;
       if ( (rw = getRWebAPI()) == null )
       {
          alert( "Unable to find Reflection, or Reflection is not initialized." );
          return;
       }

       // Create an Array object to store the <form> values, and get
       // the values. If any of the required values could not be
       // returned, exit the function without transferring the file.
       var fields = new Array();
       if ( getFields( fields ) )
       {
          // Determine the direction of the transfer, and then issue the
          // appropriate API method, passing the four field values
          // as parameters.
          if ( direction == "send" )
```

```
            rw.indSendFile( fields[0], fields[1], fields[2], fields[3] );
          else if ( direction == "receive" )
            rw.indReceiveFile( fields[0], fields[1], fields[2], fields[3] );
          else
            alert( "Invalid parameter specified for transfer direction." );
        }

        // After the transfer, request the display focus for the session,
        // so the user can type in the terminal session window.
        rw.requestDisplayFocus();
    }

    /*
       This function gets a reference to the IBM 3270 applet on the
       page, and then gets a reference to the JSAPI for the session,
       and then ensures that it is initialized. The function returns
       the API reference if the session is available, or null if either
       the session or API is not available or not initialized.
    */
    function getRWebAPI()
    {
       var targetApplet = document.IBM3270;
       if ( targetApplet == null ) return null;
       var api = targetApplet.getAPI( "JSAPI", "IBM3270" );
       if ( api == null ) return null;
       if ( api.getInitialized() <= 0 ) return null;
       return api;
    }

    /*
       This function fills in the array object with the values from
       the <form> elements. The function returns true if all of the values
       are valid, or false if the data is not valid. In this example,
       both the local file and the host file are required information,
       so if either is missing, an alert message is displayed.
    */
    function getFields( fields )
    {
       var localFile, hostFile, isAscii, showStatus;

       if ( (localFile = document.FileTransfer.localfile.value) == "" )
       {
          alert( "You must enter a local file name." );
          return false;
       }
       if ( (hostFile = document.FileTransfer.hostfile.value) == "" )
       {
          alert( "You must enter a host file name." );
          return false;
       }

       var typeList = document.FileTransfer.FileType;
       isAscii = (typeList.options[typeList.selectedIndex].index == 0) ? true :
false;

       showStatus = true; // Currently unused; can be either true or false.

       // Fill in the array with the valid data.
       fields[0] = localFile;
       fields[1] = hostFile;
```

```
       fields[2] = isAscii;
       fields[3] = showStatus;

       return true;
    }
//-->
</script>
</head>
<body>
<h1>Reflection for the Web -- IBM 3270</h1>
<p>
<!--
   This is the tag that launches an IBM 3270 terminal session.
   To use this example, change the hostURL parameter to one
   appropriate for your network.
-->
<applet mayscript name="IBM3270"
        codebase="./ex/"
        code="com.wrq.rweb.Launcher.class"
        width="600" height="400"
        archive="Launcher.jar">
   <param name="hostURL" value="tn3270://payroll">
   <param name="launcher.sessions" value="IBM3270">
   <param name="preloadJSAPI" value="true">
</applet>
</p>
<!--
   A series of <form> elements are created to let the user enter
   the name of the local file, the name of the host file, and
   the format for transferring the file (either ASCII or Binary).
   Two buttons trigger the file transfer. By using an INPUT type
   of "FILE", the local file name field looks like a text field,
   but also gets a Browse button for browsing for a local file.
   Note, however, that the Browse dialog box cannot be used to
   specify a file to save when receiving a file from the host.
-->
<form name="FileTransfer">
<p>Local File Name: <input type="file" size="40" name="localfile"></p>
<p>Host File Name:  <input type="text" value="" size="40"
name="hostfile"></p>
<p>File type: <select name="FileType">
<option selected>ASCII</option>
<option>Binary</option>
</select>
</p>
<p>
<input type="button" name="RecvFile" value="<< Receive File"
onClick="transferFile( 'receive' )"> 
<input type="button" name="SendFile" value="Send File >>"
onClick="transferFile( 'send' )">
</p>
</form>
</body>
</html>
```

## Logging out of the host when closing a session

When a user closes a Reflection session window without first logging out of the host, a process may remain running on the host computer, preventing the user from logging on again. This example shows how to prevent this problem; it creates the Reflection session in a separate JavaScript window

containing a script that logs the user out of the host computer when the new window is closed. In this example, the script simply checks whether the session is connected, and if it is, transmits the "logout" string.

To use this example, you must change the `hostURL` parameter in the `<applet>` tag that's part of the JavaScript function that writes out the Reflection session to a host appropriate for your network. You may also need to modify the code that transmits the "logout" string.

```
<html>
<head>
<title>Sample Logout on Close Script</title>
<script language="JavaScript">
<!--
    /*
       Copy relevant symbolic constants from .js file.
    */
    var IBM3270_ENTER_KEY = 35;
    var VT_RETURN_KEY = 2;

    /*
       This function writes the applet tag to launch an IBM 3270 session in
       a new browser window called "IBM3270Applet." In addition to the
       IBM 3270 session applet, some JavaScript functions are written to the
        tag in the new window: an "onLoad" event handler is added to
       give the terminal session the input focus when it is ready, and an
       "onUnload" event handler is added to logout of the host when the
       browser window is closed.
    */
    function launchIBMSession()
    {
        var targetwin = window.open( "", "IBM3270Applet",
                                "status=yes, width=650, height=550" );
        var targetdoc = targetwin.document;

        // Use the "with (object)" statement to simplify the following lines.
        with ( targetdoc ) {
            writeln( '<html>' );
           writeln( '<body onLoad="setSessionFocus()" onUnload="doDisconnect()">'
);
            writeln( '<script language="JavaScript">' );
            writeln( '   window.onerror = errHandler;' );
            writeln( '   var api = null;' );
            writeln( '   function errHandler(msg, url, line) {' );
            writeln( '      setTimeout("setSessionFocus()",1000);' );
            writeln( '      return true;' );
            writeln( '   }' );
            writeln( '   function setSessionFocus() {' );
            writeln( '      api = document.IBM3270.getAPI("JSAPI","IBM3270");' );
            writeln( '      if ( api.getInitialized() <= 0  )' );
            writeln( '         setTimeout("setSessionFocus()",1000);' );
            writeln( '      else' );
            writeln( '         api.requestDisplayFocus();' );
            writeln( '   }' );
            writeln( '   function doDisconnect() {' );
            writeln( '      if ( api.isConnected() ) {' );
            writeln( '         api.transmitString("logout");' );
           writeln( '        api.transmitTerminalKey(" + IBM3270_ENTER_KEY + ");'
);
            writeln( '      }' );
            writeln( '   }' );
```

```
            writeln( '</script>' );
            writeln( '<h1>Reflection for the Web -- IBM 3270</h1>' );
            writeln( '<applet mayscript name="IBM3270"' );
            writeln( '       codebase="./ex/"' );
            writeln( '       code="com.wrq.rweb.Launcher.class"' );
            writeln( '       width="600" height="400"' );
            writeln( '       archive="Launcher.jar">' );
            // Change the hostURL to a value appropriate for your network.
            writeln( '   <param name="hostURL" value="tn3270://payroll">' );
            writeln( '   <param name="autoconnect" value="true">' );
            writeln( '   <param name="frame" value="false">' );
            writeln( '   <param name="launcher.sessions" value="IBM3270">' );
            writeln( '</applet>' );
            writeln( '</body>' );
            writeln( '</html>' );
            close();
        }
    }

    /*
       This function writes the applet tag to launch a VT session in
       a new browser window called "VTApplet." In addition to the
       VT session applet, some JavaScript functions are written to the
        tag in the new window: an "onLoad" event handler is added to
       give the terminal session the input focus when it is ready, and an
       "onUnload" event handler is added to logout of the host when the
       browser window is closed.
    */
    function launchVTSession()
    {
        var targetwin = window.open( "", "VTApplet",
                                  "status=yes, width=650, height=550" );
        var targetdoc = targetwin.document;

        // Use the "with (object)" statement to simplify the following lines.
        with ( targetdoc ) {
            writeln( '<html>' );
           writeln( '<body onLoad="setSessionFocus()" onUnload="doDisconnect()">'
);
            writeln( '<script language="JavaScript">' );
            writeln( '   window.onerror = errHandler;' );
            writeln( '   var api = null;' );
            writeln( '   function errHandler(msg, url, line) {' );
            writeln( '      setTimeout("setSessionFocus()",1000);' );
            writeln( '      return true;' );
            writeln( '   }' );
            writeln( '   function setSessionFocus() {' );
            writeln( '      api = document.VT.getAPI("JSAPI","VT");' );
            writeln( '      if ( api.getInitialized() <= 0  )' );
            writeln( '         setTimeout("setSessionFocus()",1000);' );
            writeln( '      else' );
            writeln( '         api.requestDisplayFocus();' );
            writeln( '   }' );
            writeln( '   function doDisconnect() {' );
            writeln( '      if ( api.isConnected() ) {' );
            writeln( '         api.transmitString("logout");' );
            writeln( '         api.transmitTerminalKey(" + VT_RETURN_KEY + ");' );
            writeln( '      }' );
            writeln( '   }' );
            writeln( '</script>' );
```

```
            writeln( '<h1>Reflection for the Web -- VT</h1>' );
            writeln( '<applet mayscript name="VT"' );
            writeln( '       codebase="./ex/"' );
            writeln( '       code="com.wrq.rweb.Launcher.class"' );
            writeln( '       width="600" height="400"' );
            writeln( '       archive="Launcher.jar">' );
            // Change the hostURL to a value appropriate for your network.
            writeln( '   <param name="hostURL" value="telnet://accounts">' );
            writeln( '   <param name="autoconnect" value="true">' );
            writeln( '   <param name="frame" value="false">' );
            writeln( '   <param name="launcher.sessions" value="VT">' );
            writeln( '</applet>' );
            writeln( '</body>' );
            writeln( '</html>' );
            close();
        }
    }
// -->
</script>
</head>
<body>
<h1>Reflection for the Web -- Logout on Close Example</h1>
<p>

<form name="LaunchSession">
<input type="button" name="LaunchIBM" value="Launch IBM 3270"
onClick="launchIBMSession()">
<input type="button" name="LaunchVT" value="Launch VT"
onClick="launchVTSession()">
</p>
</form>
</body>
</html>
```

## VBScript Examples

- Logging on to an IBM 3270 host (VBScript)
- Logging on to a VMS host (VBScript)

### Logging on to an IBM 3270 host (VBScript)

In this example, which runs only in Internet Explorer, Microsoft's VBScript is used to launch an IBM 3270 session and log on to the host computer. Two <form> fields on the web page are used to get the user's name and password before creating the Reflection session applet and performing the logon. The Reflection session is created in a new browser window that has only a status bar. This is done so the VBScript WriteLn statements don't replace the contents of the current window and make the rest of the script unavailable. A related example shows a JavaScript version of this script.

To use this example, you must change the `hostURL` parameter in the `<applet>` tag that's part of the `DoWriteApplet()` function below to a host appropriate for your network. In addition, the HTML page must be placed in the session folder rather than in the `ReflectionData/deploy` folder. If the session were delivered from the deploy folder as a protected session, dynamically generated code that gets added to the applet tag written out by the WriteLn statements would contain nested double quotation marks and would generate a syntax error.

```
<html>
<head>
<title>Sample Logon Script for IBM 3270 (VBScript)</title>
<script language="VBScript">
<!--
    ' Script-level variables shared by various functions.
    Dim strUserid
    Dim strPword
    Dim intInited
    Dim intCount
    Dim objAppletTarget
    Dim objAPI
    Dim objTargetdoc
    Dim objTargetwin

    ' Copy relevant symbolic constants from
    ' api_ibm3270_constants.js and modify as needed for VBScript.
    Dim ANY_COLUMN
    Dim IBM3270_ENTER_KEY
    Dim IBM3270_TAB_KEY
    ANY_COLUMN = -1
    IBM3270_ENTER_KEY = 35
    IBM3270_TAB_KEY = 52

    '*************************************************
    ' Handle a click on the Logon button. This first
    ' initializes the script-level variables, and then validates
    ' the form inputs. If the form inputs are okay, the
    ' procedure DoWriteApplet is called to write the applet
    ' tag to a new browser window.
    '*************************************************
    Sub Logon_OnClick()
        Call DoInitVariables()
        If blnValidateInputs() = True Then
            Call DoWriteApplet()
        End If
    End Sub

    '*************************************************
    ' Initialize the script-level variables.
    '*************************************************
    Sub DoInitVariables()
        strUserid = ""
        strPword = ""
        intInited = 0
        intCount = 0
        objAppletTarget = Null
        objAPI = Null
        objTargetdoc = ""
        objTargetwin = ""
    End Sub

    '*************************************************
    ' Check for a username and password, and if either is
    ' not present, display a message box and return False.
    ' If the User ID and Password are filled in, return
    ' True.
    '*************************************************
    Function blnValidateInputs()
        strUserid = Document.LogonForm.Userid.Value
```

```
        strPword = Document.LogonForm.Password.Value
        If strUserid = "" Then
            Alert "You must enter a User ID to log on!"
            Document.LogonForm.Username.Focus()
            blnValidateInputs = False
            Exit Function
        End If

        If strPword = "" Then
            Alert "You must enter a Password to log on!"
            Document.LogonForm.Password.Focus()
            blnValidateInputs = False
            Exit Function
        End If
        blnValidateInputs = True
    End Function

    '*************************************************
    ' This function writes the applet tag to launch an
    ' IBM 3270 session in a new browser window called
    ' "IBM3270Applet." When the new window is done loading
    ' the new window's onLoad event handler is invoked,
    ' which calls the DoLogon procedure in this script.
    '*************************************************
    Sub DoWriteApplet()
        Set objTargetwin = Window.Open( "", "IBM3270Applet", "status=yes,
width=650, height=550" )
        Set objTargetdoc = objTargetwin.Document

        objTargetdoc.Open
        objTargetdoc.WriteLn "<html>"
        objTargetdoc.WriteLn "<body language='VBScript'
OnLoad='Window.Opener.DoLogon'>"
        objTargetdoc.WriteLn "<h1>Reflection for the Web -- IBM 3270</h1>"
        objTargetdoc.WriteLn "<applet mayscript name='IBM3270'"
        objTargetdoc.WriteLn "        codebase='../ex/'"
        objTargetdoc.WriteLn "        code='com.wrq.rweb.Launcher.class'"
        objTargetdoc.WriteLn "        width='600' height='400'"
        objTargetdoc.WriteLn "        archive='Launcher.jar'>"
        ' Change the hostURL parameter to a host appropriate for your network.
        objTargetdoc.WriteLn "    <param name='hostURL' value='tn3270://payroll'>"
        objTargetdoc.WriteLn "    <param name='autoconnect' value='false'>"
        objTargetdoc.WriteLn "    <param name='frame' value='false'>"
        objTargetdoc.WriteLn "    <param name='preloadJSAPI' value='true'>"
       objTargetdoc.WriteLn "    <param name='launcher.sessions' value='IBM3270'>"
        objTargetdoc.WriteLn "</applet>"
       objTargetdoc.WriteLn "<form><input type='button' name='Close' value='Close
Window' "
        objTargetdoc.WriteLn "OnClick='Window.Close()'></form>"
        objTargetdoc.WriteLn "</body>"
        objTargetdoc.WriteLn "</html>"
        objTargetdoc.Close()
    End Sub

    '*************************************************
    ' This procedure is called by the applet window's onLoad
    ' event handler after the new child window has been loaded.
    ' It gets a reference to the Reflection Launcher applet on the
    ' web page, and if the reference is null, it shuts the
    ' child window. Otherwise, it calls the DoGetAPI()
```

```
' procedure.
'*************************************************
Sub DoLogon()
    On Error Resume Next
    Set objAppletTarget = objTargetdoc.IBM3270
    If objAppletTarget = Null Then
        Alert "Could not get applet reference. The script is stopping now."
        objTargetwin.Close()
        Exit Sub
    Else
        DoGetAPI
    End If
End Sub

'*************************************************
' This procedure gets the Reflection JSAPI object for
' the terminal session on the web page. If the API
' object is null, it rechecks every 2 seconds. When
' the API object is available, the procedure then
' gets the Reflection session's initialization status,
' again pausing 2 seconds between attempts, before calling
' DoConnect. For more information about determining
' Reflection's initialization status, see Using an onLoad
' Handler to Determine When Reflection Is Initialized.
'*************************************************
Sub DoGetAPI()
   Set objAPI = objAppletTarget.getAPI("JSAPI", "IBM3270")
   If objAPI = Null Then
      SetTimeout "DoGetAPI()", 2000
   Else
      On Error Resume Next
      intInited = objAPI.getInitialized()
      If intInited = 0 Then
         If intCount < 10 Then
            intCount = intCount + 1
            SetTimeout "DoGetAPI()", 2000
         Else
            Alert "Unable to initialize Reflection."
         End If
      ElseIf intInited > 0 Then
         DoConnect
      Else
         Alert "Reflection initialization failed."
      End If
   End If
End Sub

'*************************************************
' This procedure starts the actual logon tasks. It
' determines if the session is connected, and if so,
' finds the Userid and Password prompts on the display
' and transmits the user's data.
'*************************************************
Sub DoConnect()
    ' If there's no connection, try to connect.
    If objAPI.isConnected() = False Then
        objAPI.connect()
    Else
        objTargetwin.Alert "You're already connected!"
        Exit Sub
```

```
        End If

        ' Use the status line to display informational messages.
        objTargetwin.Status = "Waiting for USERID prompt."

        ' Wait 5 seconds for the USERID prompt to appear on the display.
        ' The coordinates in which to start looking are any column in
        ' row 19. The API uses 0-based coordinates, in contrast to the
        ' 1-based coordinates that are used for the status line.
        If objAPI.waitForDisplayString( "USERID", 19, _
                                ANY_COLUMN, 5000 ) = False Then
           objTargetwin.Alert "Unable to find USERID prompt. The script will stop,
" & _
                   "but you may continue manually."
            Exit Sub
        End If

        ' Wait 5 seconds for the cursor to appear in row 19, column 16.
        ' This is the location for the user's ID.
        If objAPI.waitForCursorEntered( 19, 16, 5000 ) = False Then
           objTargetwin.Alert "Cursor was not found in the USERID field. The script
" & _
                   "will stop, but you may continue manually."
            Exit Sub
        End If

        ' USERID field was found, and cursor is in place, so transmit the user's
        ' ID, followed by the Tab key to move to the next field.
        objAPI.transmitString( strUserid )
        objAPI.transmitTerminalKey( IBM3270_TAB_KEY )

        ' Wait for the cursor to enter the Password field.
        objTargetwin.Status = "Waiting for Password prompt."
        If objAPI.waitForCursorEntered( 20, 16, 5000 ) = False Then
            objTargetwin.Alert "Cursor was not found in the Password field. The
script will " & _
                       "stop, but you can try completing the logon manually."
            Exit Sub
        End If

        ' Cursor is in Password field, so transmit the user's password, followed
        ' by the Enter key.
        objAPI.transmitString( strPword )
        objAPI.transmitTerminalKey( IBM3270_ENTER_KEY )


        ' Wait 5 seconds for the "Ready" message. If not found, display
        ' a message box.
        objTargetwin.Status = "Waiting for Ready prompt."
        If objAPI.waitForDisplayString( "Ready;", 5000 ) = False Then
           objTargetwin.Alert "Ready signal not found. The session may not be ready
for input."
            Exit Sub
        End If

        ' Display a final status message, and set the focus to the
        ' terminal display.
        objTargetwin.Status = "Ready to proceed."
        objAPI.requestDisplayFocus()
```

```
    End Sub

// -->
</script>
</head>
<body bgcolor="#FFFFFF">
<h1>Reflection for the Web -- IBM 3270 Logon</h1>
<!--
   Use <form> items to accept the User ID and Password, and to create
   a Logon button.
-->
<form name="LogonForm">
<p>User ID: <input type="text" name="UserID" default size="25"></p>
<p>Password: <input type="password" name="Password" size="25"></p>
<p><input type="button" name="logon" value="Logon"></p>
</form>
</body>
</html>
```

## Logging on to a VMS host (VBScript)

In this example, a VT session is embedded in the browser window, and VBScript, which is available only in Internet Explorer, is used to log on to the host computer. Two <form> fields on the web page are used to get the user's name and password before performing the logon. Before the logon is actually initiated, a loop with a time delay is used to determine if Reflection is initialized and ready for additional commands. A related example shows a JavaScript version of this script.

```
<html>
<head>
<title>Sample Logon Script for VT (VBScript)</title>
<script language="VBScript">
<!--
    ' Script-level variables shared by various functions.
    Dim strUname
    Dim strPword
    Dim intInited
    Dim intCount
    Dim objAppletTarget
    Dim objAPI

    '*************************************************
    ' Handle a click on the Logon button. This first
    ' initializes the script-level variables, and then validates
    ' the form inputs. If the form inputs are okay,
    ' DoTryInit is called to start the action.
    '*************************************************
    Sub Logon_OnClick()
        Call DoInitVariables()
        If blnValidateInputs() = True Then
            Call DoTryInit()
        Else
            Exit Sub
        End If
    End Sub

    '*************************************************
    ' Initialize the script-level variables.
    '*************************************************
    Sub DoInitVariables()
        strUname = ""
```

```
    strPword = ""
    intInited = 0
    intCount = 0
    objAppletTarget = Null
    objAPI = Null
End Sub

'**************************************************
' Check for a username and password, and if either is
' not present, display a message box and return False.
' If the username and password are filled in, return
' True.
'**************************************************
Function blnValidateInputs()
    strUname = Document.LogonForm.Username.Value
    strPword = Document.LogonForm.Password.Value
    If strUname = "" Then
        Alert "You must enter a Username to log on!"
        Document.LogonForm.Username.Focus()
        validateInputs = False
        Exit Function
    End If

    If strPword = "" Then
        Alert "You must enter a Password to log on!"
        Document.LogonForm.Password.Focus()
        blnValidateInputs = False
        Exit Function
    End If
    blnValidateInputs = True
End Function

'**************************************************
' This procedure gets a reference to the Reflection
' applet on the page and stores it in "objAppletTarget".
' If that succeeds, it then calls DoGetAPI() to get
' the Reflection API.
Sub DoTryInit()
    On Error Resume Next
    Set objAppletTarget = Document.VT
    If objAppletTarget = Null Then
        Alert "Could not get applet reference. The script is stopping now."
        Exit Sub
    Else
        DoGetAPI
    End If
End Sub

'**************************************************
' This procedure gets the Reflection JSAPI object for
' the terminal session on the web page. If the API
' object is null, it rechecks every 2 seconds. When
' the API object is available, the procedure then
' gets the Reflection session's initialization status,
' again pausing 2 seconds between attempts, before calling
' DoConnect. For more information about determining
' Reflection's initialization status, see Using an onLoad
' Handler to Determine When Reflection Is Initialized.
'**************************************************
Sub DoGetAPI()
```

```
   Set objAPI = objAppletTarget.getAPI("JSAPI", "VT")
   If objAPI = Null Then
      SetTimeout "DoGetAPI()", 2000
   Else
      On Error Resume Next
      intInited = objAPI.getInitialized()
      If intInited = 0 Then
         If intCount < 10 Then
            intCount = intCount + 1
            SetTimeout "DoGetAPI()", 2000
         Else
            Alert "Reflection is not initialized. Unable to continue."
         End If
      ElseIf intInited > 0 Then
         DoConnect
      Else
         Alert "Reflection initialization failed. Unable to continue."
      End If
   End If
End Sub

'*************************************************
' This function starts the actual logon tasks. It
' determines if Reflection is connected, and if so,
' calls the function that finds the Username and
' Password prompts on the display and transmits the
' user's data.
'*************************************************
Sub DoConnect()

    ' If there's no connection, try to connect.
    If objAPI.isConnected() = False Then
        objAPI.connect()
    Else
        Alert "You're already connected!"
        Exit Sub
    End If

    ' Wait 5 seconds for the Username prompt.
    If objAPI.waitForString( "Username:", 5000 ) = False Then
        Alert "Unable to find Username prompt. The script will stop, " & _
                "but you may continue manually."
        Exit Sub
    End If

    ' Username prompt was found, so transmit the user's username
    ' followed by a carriage return. The VBScript string constant
    ' "vbCR" is used to represent the carriage return. You can
    ' also use the expression "Chr(13)," which is the decimal
    ' representation of the carriage return character.
    objAPI.transmitString( strUname & vbCR )

    ' Wait 5 seconds for the Password prompt.
    If objAPI.waitForString( "Password:", 5000 ) = False Then
        Alert "Unable to find Password prompt. The script will stop, " & _
                "but you can try completing the logon manually."
        Exit Sub
    End If

    ' Password prompt was found, so transmit the user's password,
```

```
        ' followed by a carriage return. As above, the VBScript constant
        ' "vbCR" is used for the carriage return character.
        objAPI.transmitString( strPword & vbCR )

        ' Set the focus to the terminal display.
        objAPI.requestDisplayFocus()

    End Sub

// -->
</script>
</head>
<body>
<h1>Reflection for the Web -- VT Logon</h1>
<p>
<!--
   Create a basic VT applet on the page but don't connect until
   the user fills in the username and password information.
   To use this example, change the hostURL parameter to one
   appropriate for your network.
-->
<applet mayscript name="VT"
        codebase="./ex/"
        code="com.wrq.rweb.Launcher.class"
        width="600" height="400"
        archive="Launcher.jar">
    <param name="hostURL" value="telnet://payroll">
    <param name="autoconnect" value="false">
    <param name="frame" value="false">
    <param name="launcher.sessions" value="VT">
</applet>
</p>
<!--
   Use <form> items to accept the Username and Password, and to create
   a Logon button.
-->
<form name="LogonForm">
<p>Username: <input type="text" name="Username" default size="25"></p>
<p>Password: <input type="password" name="Password" size="25"></p>
<p><input type="button" name="logon" value="Logon"></p>
</form>
</body>
</html>
```

# 3 Applet Attributes and Parameters

This reference provides a listing of the applet parameters and attributes used by Reflection. It includes a description, valid values, and examples for each parameter and attribute.



## About Applets in Reflection for the Web

When a user requests a Reflection URL, Reflection for the Web dynamically generates a web page containing a launcher applet that loads the module named by the launcher.sessions parameter. (You can also save a Reflection web page as a static session; see HTML Samples. The launcher.sessions module determines which tools or sessions to run and presents the appropriate page.

When a user selects the Reflection base URL, the launcher applet loads the request manager client module. The module checks with the server to determine if a login is required, and if so presents the login form. After the user logs in, the module presents the customized list of links. When the user clicks a link for a session that appears in its own window, the same module creates a child window in which the emulator session appears. When the user clicks a link for an embedded session, on the other hand, Reflection generates a new browser window with a new launcher applet and a different module--the terminal session itself.

You can change emulator session behavior by using **Manage Sessions** to add parameters to the session applet tag. These session-specific parameters are added to embedded sessions when they're launched in new browser windows, and to sessions that appear in their own window ("framed" sessions) when they're launched from the links list.

Launcher parameters (such as launcher.keepalive and launcher.sessions) apply to the module-launching applet and not to the module itself. Because these parameters are applied only when the launcher applet first loads and framed sessions are children of the initial launcher module, launcher parameters cannot be set for specific framed sessions.

Define the contents of the session web page using **Manage Sessions**.

### Customizing applet page presentation

HTML Samples provide instructions on how to customize an applet page and examples of HTML code to run Reflection applets.

### Applet security

A Java applet is required to follow security rules, which limit the applet's access to a user's computer.

**Related Topics**

- Applet Attributes and Parameters
- Index of Attributes and Parameters

# Applet Attributes and Parameters

Use these additional settings to customize the way a session is displayed, launched, and delivered.

## Applet Attributes

Reflection applet attributes are the standard Java attributes used by all applets. You can find additional detail about these and other valid attributes in many HTML and Java references.

## Applet Parameters

HTML `applet` tags contain `param` sub-tags. The `param` tags specify parameter names and values that the applet uses when it loads. The basic syntax for the `param` tag is:

```
<param name="name" value="value">
```

**Related Topics**

- About Applets in Reflection for the Web
- Index of Attributes and Parameters

# Index of Attributes and Parameters

Use this index to locate detailed information about the Reflection terminal emulation applet attributes and parameters. For general information about which names and values are case sensitive, go to Case Sensitivity in Attributes and Parameters.

---

**NOTE:** Attribute values are case sensitive, all other attributes and parameters are not. The one exception is a parameter value that references a file or Java class. In that case the parameter value is case sensitive.

---

A-B-C | D-E-F | G-H-I | J-K-L | M-N-O | P-Q-R | S-T-U | V-W-X-Y-Z | Numbers

## A-B-C

## allowStoredMacroPassword

This parameter determines if a user can store passwords or other hidden variables. If set to true, the user can change the macro variable from **Always prompt User for Value** to **Embed fixed user response in macro** on the Save Macro dialog box after recording a macro. You must configure this parameter for each individual session you create.

---

**NOTE:** This parameter only affects the behavior of the Save Macro dialog box. If you have existing macros with passwords or hidden variables embedded in them, their behavior will not change.

This parameter does not apply to single sign-on macros or express logon macros for IBM 3270 sessions.

---

**Value**

`true` (Default)

`false`

**Example**

```
<param name="allowStoredMacroPassword" value="true">
```

## answerback

see declans parameter

## archive

This optional attribute specifies the JAR (Java Archive) file containing the applet code. The archive attribute is supported by browsers using the Java Plug-in.

When applet code is packaged into a JAR file for running in the Java Plug-in, the `archive` attribute is used to specify the name and location of the archive. Although the `archive` attribute is optional in the applet tag, all Reflection for the Web applet code is packaged into JAR files, and to run Reflection using the Java Plug-in, the archive attribute is required.

**Value**

`Launcher.jar`

**Example**

```
<applet mayscript name="IBM3270"
        code="com.wrq.rweb.Launcher.class"
        width="400" height="300"
        archive="Launcher.jar">
</applet>
```

## Autoconnect

This parameter specifies whether the Reflection session should connect to the host specified in the host URL parameter when the session starts. If you choose not to connect automatically, you can connect to a specified host using the Reflection menu (if it is available). For embedded emulations, it is recommended that you automatically connect to the host when you start the session.

**NOTE:** The recommended method for configuring most settings is to use the dialog box equivalents where available. In these cases, parameters should be restricted to static pages or scripted sessions. The equivalent setting for the `autoconnect` parameter is **Connect at startup.**

#### Value

`true` (Default)

`false`

#### Example

```
<param name="autoconnect" value="true">
```

## BackgroundColor

This parameter determines the color of the background on the login and links list interface. The default background color is white.

#### Value

You can specify a color in one of three ways:

- Hexadecimal - `<param name="backgroundColor" value="#CCCCCC">`
- RGB (red, green, blue) - `<param name="backgroundColor" value="204,204,204">`
- One of 13 named colors: black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, and yellow - `<param name="backgroundColor" value="lightGray">`

#### Example

In the following applet tag, the background of the login form and links list interfaces is set to pale yellow.

```
<applet name="Launcher" code="com/wrq/rweb/Launcher.class" archive="Launcher.jar"
codebase="./ex" width="500" height="440">
<param name="backgroundColor" value="#FFFFCC">
</applet>
```

## certAndCRLSignedBySamePrivateKey

This parameter, If set to true, instructs the certificate verification code to verify that the certificate and the CRLs (Certificate Revocation Lists) are signed by the same CA (Certificate Authority) private key. The default is false.

Certificate Authorities may use one private key to digitally sign certificates and CRLs, but CAs can also use separate private keys to digitally sign certificates and CRLs. This applies to SSL/TLS sessions only, and requires that the CRL be already present.

#### Value

`true`

`false` (Default)

#### Example

```
<param name="certAndCRLSignedBySamePrivateKey" value="true">
```

## certExplicitPolicyIndicator

This parameter, when set to true, instructs the certificate verification code to verify that the first CA certificate in a certificate chain has the Certificate Explicit Policy Indicator extension set. This parameter allows you to require an explicit policy identifier for certificate path processing. This applies to SSL/TLS sessions only.

**Value**

`true`
`false` (Default)

**Example**

```
<param name="certExplicitPolicyIndicator" value="true">
```

## certPolicyOIDs

This parameter specifies the initial certificate policies. It instructs the certificate verification code to verify that a certificate chain meets the policy requirement. Specify the initial policy OIDs (Object Identifiers), separated by commas. The default is empty. This applies to SSL/TLS sessions only.

**Value**

Any string; may be empty (Default).

**Example**

```
<param name="certPolicyOIDs"
value="2.16.840.1.101.2.1.48.1,2.16.840.1.101.3.1.48.3">
```

## Code

This required attribute specifies the name of the class file used to start the applet. The value is relative to the URL specified in the `codebase` attribute, or if no `codebase` attribute is present, to the URL of the document containing the applet. This value is case sensitive.

**Value**

`com.wrq.rweb.Launcher.class`

The `code` attribute must be used in conjunction with the `Launcher.sessions` parameter.

**Example**

```
<applet mayscript name="IBM3270"
        code="com.wrq.rweb.Launcher.class"
        width="400" height="300">
</applet>
```

## Codebase

This attribute specifies the URL for the folder that contains the terminal emulation applet JAR files. The URL can be relative to the location of the HTML page or it can specify the entire path.

**Value**

```
./ex/ (Default)
<Any valid URL>
```

#### Example

`Codebase` specifies entire path:

```
<applet mayscript name="Reflection"
        codebase="http://ReflectionWeb.example.com/connections/"
        code="com.wrq.rweb.Launcher.class"
        width="400" height="300">
</applet>
```

`Codebase` is relative to the location of the HTML page:

```
<applet mayscript name="Reflection"
        codebase="java/"
        code="com.wrq.rweb.Launcher.class"
        width="400" height="300">
</applet>
```

## Configuration

This parameter specifies the URL or name of a configuration file (usually with a .config extension) created by a system administrator. For more information, go to the Overview of Configuring Reflection.

The value of the configuration parameter is treated initially as an absolute URL that points directly to the file. If the file cannot be opened, Reflection appends the value to the URL of the current document base (the location of the web page that is running the applet). If using the document base does not locate the file, then the value is appended to the codebase for the page, named in the Codebase attribute of the applet. (If no codebase is specified, the codebase is the same as the document base.)

#### Value

```
<any valid URL>
```

#### Example

```
<param name="configuration" value="salesdept.config">
```

## crlIssuers

This parameter specifies the URLs of the CRLs (Certificate Revocation Lists) that are used by certificate path processing for checking the certificate revocation status of a certificate chain. The default is empty. This applies to SSL/TLS sessions only. LDAP, file, and HTTP URLs are supported.

#### Value

Any series of URLs separated by semicolons; may be empty (Default).

#### Example

```
<param name="crlIssuers" value="ldap://myCAServer.example.com/CA/
  certificaterevocationlist; ldap://rootCA.verisign.com/CRL">
<param name="crlIssuers" value="file://localhost/c:/crls/TrustAnchorCRL.crl">
<param name="crlIssuers" value="http://server1.example.com/CertEnroll/
  server1.example.com.crl">
```

Top of index

# D-E-F



## declans

This parameter specifies the text of an answerback message that is sent to the host in response to an ENQ character. This text is linked to the Answerback message box in the Advanced VT Terminal Items dialog box. (The name of the parameter derives from the mnemonic for the VT terminal's "Load Answerback Message" control sequence.) The `declans` parameter applies to VT emulations only.

### Value

Any string.

### Example

```
<param name="declans" value="Hello world.">
```

## DestinationName

This parameter indicates the ultimate host to which the Reflection security proxy server connects. This parameter is valid only when user authorization is enabled on the proxy server; otherwise, the destination host and port are predefined by the proxy server for a given server port. The value -1 can be used to indicate that the user may specify a destination host at runtime; in this case, the `DestinationName` and `DestinationPort` parameters must both be -1.

This parameter is used in conjunction with the `securityEnabled` parameter, and is valid for dynamically generated or protected static sessions only. When the HTML is delivered to the user, the `DestinationName` and `DestinationPort` parameters are removed from the HTML to hide the name and port of the ultimate destination host if the user views the HTML source from the web browser.

---

**NOTE:** The recommended method for configuring most settings is to use the dialog box equivalents where available. In these cases, parameters should be restricted to static pages or scripted sessions. The equivalent setting for the `DestinationName` parameter is Destination host.

---

#### Value

The name of the actual destination host to which the proxy server will connect.

#### Example

```
<param name="DestinationName" value="myHost">
```

## DestinationPort

This parameter indicates the ultimate port to which the Reflection security proxy server connects. `DestinationPort` is valid only when user authorization is enabled on the proxy server; otherwise, the destination host and port are predefined by the proxy server for a given server port. The value -1 can be used to indicate that the user may specify a destination port at runtime; in this case, the `DestinationName` and `DestinationPort` parameters must both be -1.

This parameter is used in conjunction with the `securityEnabled` parameter, and is valid for dynamically generated or protected static sessions only. When the HTML is delivered to the user, both `DestinationName` and `DestinationPort` parameters are removed from the HTML to hide the name and port of the ultimate destination host if the user views the HTML source from the web browser.

#### Value

The name of the actual destination port to which the proxy server will connect.

#### Example

```
<param name="DestinationPort" value="23">
```

## deviceName

This parameter is only used with IBM terminal emulation. This parameter specifies the device name (also known as an LU--logical unit) or pool to use when the session connects to the host. If no device or pool is specified for a terminal session, the host dynamically assigns an LU to the session.

**NOTE:** The recommended method for configuring most settings is to use the dialog box equivalents where available. In these cases, parameters should be restricted to static pages or scripted sessions. Otherwise, use an equivalent setting for the `deviceName` parameter, such as **Specify device name**.

**Value**

`<any valid device name>`

**Example**

```
<param name="deviceName" value="LU12orion">
<param name="deviceName" value="NWpool">
```

## disableAutoRepeat

This parameter turns off the automatic repeat feature for a specified key or keys.

**Value**

A whitespace-delimited string of decimal numbers that correspond to any KeyEvent names. `KeyEvent` names are available in the JDK's KeyEvent source. Note that values must be specified in decimal rather than hexadecimal form.

**Example**

The following example disables autorepeat on the Enter/Return (VK_ENTER = \n = 10) and F1 (VK_F1 = 0X70 = 112) keys.

```
<param name="disableAutoRepeat" value="10 112">
```

## displayDeviceName

This parameter specifies whether or not the Reflection session frame displays the device name established with the host. The default is false. This parameter applies to IBM terminal and printer emulators (IBM 3270, IBM 5250, IBM 3287, and IBM 3812) only.

**Value**

`true`
`false` (Default)

**Example**

```
<param name="displayDeviceName" value="true">
```

## emailtechsupport

This parameter redirects the Contact Technical Support command in the Help menu for the terminal session to the specified URL.

**Value**

`http://support.attachmate.com` (Default)
`<any valid URL>`

**Example**

```
<param name="emailtechsupport" value="http://support.example.com">
<param name="emailtechsupport" value="mailto:helpdesk@example.com">
```

## encryptStream

This parameter directs the terminal session to use SSL/TLS datastream encryption.

**Value**

`true`
`false` (Default)

**Example**

```
<param name="encryptStream" value="true">
```

## enterviewhomeid

This parameter redirects the Reflection for the Web Home Page command in the Help menu for the terminal session to the specified URL.

**Values**

`http://www.attachmate.com/Products/Terminal+Emulation/refweb/refweb.htm` (Default)
`<any valid URL>`

**Example**

```
<param name="enterviewhomeid" value="http://www.example.com/Reflection.html">
```

## errorAlarm

This parameter applies to IBM 5250 emulation and controls whether a keyboard error is accompanied by an audible beep. If this parameter is not present, keyboard errors are reported quietly.

**Value**

`true`
`false` (Default)

**Example**

```
<param name="errorAlarm" value="true">
```

## forceCRLF

This parameter is valid only for VT terminal sessions. If users are having problems with the output from VT controller mode printing not properly aligning on the page, use the `forceCRLF` parameter to insert a carriage return (CR) and line feed (LF) when a <LF> is received in the host data intended to be printed. Inserting a <CR><LF> ensures each line of printer output starts at the left margin.

**Value**

`true`
`false` (default)

#### Example

```
<param name="forceCRLF" value="true">
```

## foregroundColor

This parameter determines the color of the text on the login and links list interface. To use this parameter, add it to the applet tag in`ReflectionData/ReflectionClient.html`.

#### Value

You can specify a color in one of three ways:

- Hexadecimal - `<param name="foregroundColor" value="#CCCCCC">`
- RGB (red, green, blue) - `<param name="foregroundColor" value="204,204,204">`
- One of 13 named colors: black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, and yellow - `<param name="foregroundColor" value="lightGray">`

The default foreground color is black.

#### Example

In the following applet tag, the background of the login form and links list interfaces is set to pale yellow.

```
<applet name="Launcher" code="com/wrq/rweb/Launcher.class" archive="Launcher.jar"
codebase="./ex" width="500" height="440">
<param name="foregroundColor" value="0,51,102">
</applet>
```

## frame

The frame parameter determines whether the terminal window is shown in a separate window. When frame is set to true, the session starts in a new window and the Reflection menu is available. When frame is set to false, the Reflection menu is unavailable unless you set the `shortcutMenu` parameter to true.

When frame is set to true, use the `framewidth` and `frameheight` parameters to determine the size of the separate terminal window. If frame is set to true and you do not include the `framewidth` and `frameheight` parameters, Reflection determines the best frame dimensions for each user based on the user's monitor resolution.

#### Value

`true`
`false` (Default)

#### Example

```
<param name="frame" value="false">
```

## frameheight

Together with `framewidth`, this parameter specifies the dimensions of the separate terminal window when frame is set to true. If frame is set to true and you do not include the `framewidth` and `frameheight` parameters, Reflection determines the best frame dimensions for each user based on the user's monitor resolution.

If frame is set to false, frameheight is ignored, and Reflection uses the applet height attribute to determine the height of the applet within the browser window.

**Value**

```
<height in pixels>
```

**Example**

```
<param name="frameheight" value="440">
```

## framewidth

Together with `frameheight`, this parameter specifies the dimensions of the separate terminal window when frame is set to true. If frame is set to true and you do not include the `framewidth` and `frameheight` parameters, Reflection determines the best frame dimensions for each user based on the user's monitor resolution.

If frame is set to false, `framewidth` is ignored, and Reflection uses the applet height attribute to determine the height of the applet within the browser window.

**Value**

```
<width in pixels>
```

**Example**

```
<param name="framewidth" value="660">
```

## FTPDestinationName

This parameter indicates the ultimate FTP server to which the Reflection security proxy server connects. This parameter is valid only when user authorization is enabled on the proxy server; otherwise, the destination host and port are predefined by the proxy server for a given server port.

This parameter is used in conjunction with the `securityEnabledFTP` parameter, and is valid for dynamically generated or protected static sessions only. When the HTML is delivered to the user, the `FTPDestinationName` and `FTPDestinationPort` parameters are removed from the HTML to hide the name and port of the destination FTP server if the user views the HTML source from the web browser.

**Value**

The name of the actual destination FTP server to which the proxy server will connect.

**Example**

```
<param name="FTPDestinationName" value="myHost">
```

## FTPDestinationPort

This needs to be complete

Top of index

# G-H-I



## GenerateUniqueDeviceName

This parameter is valid for IBM 3270 emulator, IBM 3270 printer, IBM 5250 emulator, and IBM 5250 printer sessions. When the parameter is set to true and no device name is specified in the configuration file or the Session Setup dialog, Reflection generates a unique device name when a connection to the host is initiated. The name is based on the machine name or the user's IP address.

**Value**

`true`
`false` (Default)

**Example**

```
<param name="generateUniqueDeviceName" value="true">
```

## Height

The required height attribute controls the initial height of the session applet embedded in the browser window. The height is measured in pixels. When the frame parameter is set to true, use the frameheight parameter to set the height of the separate Reflection terminal window.

**Example**

```
<applet mayscript name="IBM3270"
        code="com.wrq.rweb.Launcher.class"
        width="400" height="300">
</applet>
```

## Heartbeat

A periodic “heartbeat” is maintained between the applet that launches the client and the Management and Security Server to prevent the server from timing out due to inactivity. If a heartbeat between the client and server is not maintained, configuration work can be lost when the inactivity timer expires.

To prevent the loss of work when undergoing lengthy configurations, such as keyboard mappings or macros, you can modify the heartbeat services that affect session management and the links list.

---

**NOTE:** If you use SiteMinder (or something similar) to control access to terminal sessions, see Using Crumb Mode (below) to set a heartbeat.

---

### Management and Security Server heartbeat

When you are configuring Reflection for the Web sessions in Management and Security Server, a heartbeat is automatically sent to the server every 3 minutes to keep the Administrative Server from timing out. To change this default or to modify other heartbeat behavior, you can add parameters on the **Configure Session** panel.

1. Open launchsession.jsp, located here with a default installation:`C:\Program Files\Micro Focus\MSS\server\web\webapps\mss\aws\smt\launchsession.jsp`
2. In launchsession.jsp, locate the section that begins with `<rweb:applet code="com.wrq.rweb.Launcher">`.
3. Add the parameters you want to set. See the available Heartbeat parameters, described below.

   For example, to change the heartbeat interval from 3 minutes to 5 minutes, add the following parameter:`<rweb:param name="HeartbeatInterval" value="5"/>`

---

**NOTE:** Edits to launchsession.jsp may be overwritten when you upgrade Reflection for the Web or Management and Security Server.

---

### Links list heartbeat

The list of links does not, by default, maintain a heartbeat with the server. Users are automatically logged out of the Management and Security Server after 60 minutes if they are not actively using the links list.

To enable a heartbeat for the links list, you must modify the source of the login page. It is recommended that you work in a custom login page to avoid having your modifications overwritten by product upgrades.

1. Create a custom login page, as described in Technical Note 2386.
2. In your custom login page, locate the section that begins with `<mss:applet userequest="true"/>`.
3. Add the heartbeat parameters you would like to set.

## Heartbeat parameters

You can add these parameters to the Management and Security Server applet and/or the links list applet to modify the default heartbeat behavior. Use the basic syntax:`<mss:param name="name" value="value"/>`.

- **enableHeartbeat** – Enables the heartbeat. The default for Session Manager is true. The default for the links list is false.
- **adminHeartbeatOnly** – Enables the heartbeat only for a user logged in as the administrator. Applies only to the links list and only when “enableHeartbeat” is true. The default is false (links list heartbeat will apply to both users and administrators).
- **heartbeatInterval** - Specifies the interval between heartbeats, in minutes. The default is 3 minutes, the minimum is 1 minute, and the maximum is 720 minutes (12 hours).
- **heartbeatMax** – Specifies the maximum lifetime of the heartbeat, in minutes. The default is 0, which lets the heartbeat run indefinitely. The maximum is 720 minutes (12 hours).
- **heartbeatRetry** – Specifies the maximum number of consecutive failed heartbeats before the heartbeat is stopped permanently. The default is 5, the minimum is 1, and the maximum is 100.
- **heartbeatCrumbMode** – Indicates whether to use “servlet request mode” or “crumb mode” for heartbeat operations. The default is false, and the heartbeat operates in “servlet request mode”. If set to true, an additional parameter, “heartbeatCrumbFilename”, can be set to specify the name of the “crumb” file if the default filename of “crumb.txt” is not desired. See Using Crumb Mode.

## Using Crumb mode

If you use SiteMinder (or something similar) to control access to terminal sessions, you can set the heartbeat to operate in “crumb mode” instead of the default “servlet request mode”. In servlet request mode, the emulator client makes a request to the servlet to maintain activity. In crumb mode, heartbeat requests are sent to fetch a crumb of data from the codebase directory on the Management and Security Server.

The crumb of data is simply a text file called crumb.txt, and the file contents must be only the string ok. You must create the crumb file manually and place it in the applet codebase directory. In a default installation, locate the directory here:`C:\Program Files\Micro Focus\MSS\server\web\webapps\mss\ex`

Access management products, such as SiteMinder, record the heartbeat crumb request as activity to prevent session timeouts, whereas they may not record activity generated by heartbeats in servlet request mode. Crumb mode does not maintain Management and Security Server activity.

Two parameters control crumb mode: `heartbeatCrumbMode` and `heartbeatCrumbFilename`, described above. The following example shows how crumb mode may be enabled in the links list and configured to use an alternate crumb file:

```
<mss:applet userequest="true">
   <mss:param name="heartbeatCrumbMode"value="true"/>
   <mss:param name="heartbeatCrumbFilename"value="mycrumb.txt"/>
</mss:applet>
```

## helpbase

This parameter specifies the path to the starting folder of the Reflection online help for the terminal session. The value should be formatted as a URL. If this parameter is not specified, Reflection uses the URL specified in the Codebase applet attribute to find the starting folder

If a relative URL is specified, Reflection starts searching for the page from the Ex folder in the terminal emulation component. By default, Reflection looks in the Help folder (considered the starting folder), installed at the same level as the Ex folder.

### Value

```
http://<terminal emulation component installation>/ (Default)  <any valid URL>
```

### Example

```
<param name="helpbase"
       value="http://Reflection.example.com/RWebHelp/">
```

## hostURL

This parameter can be used to specify three values: transport, host, and port. These values are named using the following syntax: `transport://host:port`.

You can define all or part of the hostURL. If you omit the host name, a dialog box prompts for a host when the session starts; if you omit the port or transport, Reflection automatically includes default values. The default value for the transport is determined by the transport parameter if one is specified. Otherwise, the transport and the port are both selected from Reflection program defaults. If you know which transport or port you need, however, it is recommended that you specify them in this parameter instead of depending on the defaults.

---

**NOTE:** The recommended method for configuring most settings is to use the dialog box equivalents where available. In these cases, parameters should be restricted to static pages or scripted sessions. The equivalent setting for the hostURL parameter is Host.

---

### Values

| Host | Values |
|---|---|
| IBM 3270 | `<host name or IP address>`<br>`<host name or IP address>:<port>`<br>`tn3270://`<br>`tn3270://<host name or IP address>`<br>`tn3270://:<port>`<br>`tn3270://<host name or IP address>:<port>`<br>`tn3270e://`<br>`tn3270e://<host name or IP address>`<br>`tn3270e://:<port>`<br>`tn3270e://<host name or IP address>:<port>`<br>`demo://ibm3270` |
| IBM 3270 Printer<br><br>The default port number for the tn3270e transport is 23. The port may be different for your organization. To change the default, specify the port number in this parameter. | `<host name or IP address>`<br>`<host name or IP address>:<port>`<br>`tn3270e://`<br>`tn3270e://<host name or IP address>`<br>`tn3270e://:<port>`<br>`tn3270e://<host name or IP address>:<port>` |

| Host | Values |
| --- | --- |
| IBM 5250 | <host name or IP address><br><host name or IP address>:<port><br>tn5250://<br>tn5250://<host name or IP address><br>tn5250://:<port><br>tn5250://<host name or IP address>:<port><br>demo://ibm5250 |
| IBM AS/400 Printer | <host name or IP address><br><host name or IP address>:<port><br>tn5250://<br>tn5250://<host name or IP address><br>tn5250://:<port><br>tn5250://<host name or IP address>:<port> |
| IBM AS/400 Data Transfer<br><br>The transport and port values are not used for IBM AS/400 data transfer; only the host name or IP address is used. | <host name or IP address> |
| HP | <host name or IP address><br><host name or IP address>:<port><br>nsvt://<br>nsvt://<host name or IP address><br>nsvt://:<port><br>nsvt://<host name or IP address>:<port><br>telnet://<br>telnet://<host name or IP address><br>telnet://:<port><br>telnet://<host name or IP address>:<port><br>demo://HP3000<br>demo://UNIX |
| VT | <host name or IP address><br><host name or IP address>:<port><br>telnet://<br>telnet://<host name or IP address><br>telnet://:<port><br>telnet://<host name or IP address>:<port><br>demo://digital<br>demo://UNIX |
| UTS | <host name or IP address><br><host name or IP address>:<port><br>int1://<br>int1://<host name or IP address><br>int1://<port><br>int1://<host name or IP address>:<port><br>airgate://<br>airgate://<host name or IP address><br>airgate://<port><br>airgate://<host name or IP address>:<port><br>matip://<br>matip://<host name or IP address><br>matip://<port><br>matip://<host name or IP address>:<port><br>pepgate://<br>pepgate://<host name or IP address><br>pepgate://<port><br>pepgate://<host name or IP address>:<port> |

| Host | Values |
|---|---|
| T27 and T27 Printer | <host name or IP address><br><host name or IP address>:<port><br>tcpa://<br>tcpa://<host name or IP address><br>tcpa://<port><br>tcpa://<host name or IP address>:<port> |
| ALC | <host name or IP address><br><host name or IP address>:<port><br>lantern://<br>lantern://<host name or IP address><br>lantern://<port><br>lantern://<host name or IP address>:<port><br>airgate://<br>airgate://<host name or IP address><br>airgate://<port><br>airgate://<host name or IP address>:<port><br>matip://<br>matip://<host name or IP address><br>matip://<port><br>matip://<host name or IP address>:<port><br>atstcp://<br>atstcp://<host name or IP address><br>atstcp://<port><br>atstcp://<host name or IP address>:<port><br>sabre://<br>sabre://<host name or IP address><br>sabre://<port><br>sabre://<host name or IP address>:<port><br>tcpfrad://<hostname>:<port><br>tcpfrad://<host name or IP address>:<port><br>udpfrad://<br>udpfrad://<host name or IP address><br>udpfrad://<port><br>udpfrad://<host name or IP address>:<port> |

#### Example

The transport, host, and port are all specified: `<param name="hostURL" value="tn3270://jupiter:23">`

Only the host is specified, and the default transport and port are used:`<param name="hostURL" value="jupiter">`

The transport and host are specified, and the default port is used: `<param name="hostURL" value="tn3270://jupiter">`

The host and port are specified, and the default transport is used. `<param name="hostURL" value="jupiter:23">`

## hphelptoc

This parameter redirects the Help Topics command in the Help menu for HP terminal sessions to the specified URL.

#### Value

`http://<terminal emulation component installation>/help/hphelp.html (Default)`

`<any valid URL>`

#### Example

```
<param name="hphelptoc"
        value="http://ReflectionWeb.example.com
                    /customHelp/hphelp.html">
```

## httpsProxy

Use this parameter to specify the port and name or IP address of a secure HTTP proxy to use while running encrypted Reflection sessions. The secure HTTP proxy specified in this parameter overrides any secure HTTP proxies that are set in the browser's properties.

You can use the httpsProxy parameter in combination with the proxyExcept parameter to bypass authentication schemes on HTTP proxies that are not supported in Reflection. The httpsProxy parameter overrides the browser settings and then the proxyExcept parameter overrides the setting for Reflection sessions. The Reflection session--encrypted using the Reflection security proxy server--can then pass directly through the firewall (an opening must be made in the firewall).

When the httpsProxy parameter is used without the proxyExcept parameter, it can be used to direct all incoming Reflection sessions to one specific secure HTTP server.

---

**TIP:** In general, this parameter is unnecessary. It should be used only when Reflection for the Web cannot automatically detect the secure HTTP proxy.

---

#### Values

Any valid secure HTTP proxy address in the following format:`address:port`

#### Example

```
<param name="httpsProxy" value="myHTTPSProxy:443">
```

## ibm3270helptoc

This parameter redirects the Help Topics command in the Help menu for IBM 3270 terminal sessions to the specified URL.

#### Value

```
http://<terminal emulation component installation>/help/ibm3270help.html (Default)

<any valid URL>
```

#### Example

```
<param name="ibm3270helptoc"
        value="http://ReflectionWeb.example.com
                    /customHelp/ibm3270help.html">
```

## ibm3287helptoc

This parameter redirects the Help Topics command in the Help menu for IBM 3270 Printer sessions to the specified URL.

#### Value

```
http://<terminal emulation component installation>/help/ibm3287help.html (Default)
```

```
<any valid URL>
```

**Example**

```
<param name="ibm3287helptoc"
        value="http://ReflectionWeb.example.com
                    /customHelp/ibm3287help.html">
```

## ibm3812helptoc

This parameter redirects the Help Topics command in the Help menu for IBM AS/400 Printer sessions to the specified URL.

**Value**

```
http://<terminal emulation component installation>/help/ibm3812help.html (Default)

<any valid URL>
```

**Example**

```
<param name="ibm3812helptoc"
        value="http://ReflectionWeb.example.com
                    /customHelp/ibm3812help.html">
```

## ibm5250helptoc

This parameter redirects the Help Topics command in the Help menu for IBM 5250 terminal sessions to the specified URL.

**Value**

```
http://<terminal emulation component installation>/help/ibm5250help.html (Default)

<any valid URL>
```

**Example**

```
<param name="ibm5250helptoc"
        value="http://ReflectionWeb.example.com
                    /customHelp/ibm5250help.html">
```

## ibmxfrhelptoc parameter

This parameter redirects the Help Topics command in the Help menu for IBM AS/400 data transfer sessions to the specified URL.

**Value**

```
http://<terminal emulation component installation>/help/ibmxfrhelp.html (Default)

<any valid URL>
```

**Example**

```
<param name="ibmxfrhelptoc"
        value="http://ReflectionWeb.example.com
                    /customHelp/ibmxfrhelp.html">
```

## ignoreASCIIData

This parameter is used in IBM 5250 emulation and IBM AS/400 printer emulation. The parameter directs the emulator to ignore any ASCII data records that may precede the first valid block-mode data (5250 data stream commands) received after connecting. By default, when the parameter is set to false, such ASCII data will result in an error message. When set to true, such ASCII data will not result in an error.

**Value**

`true`
`false` (Default)

**Example**

```
<param name="ignoreASCIIData" value="false">
```

## ignoreHostPrintRequest

This parameter specifies whether host-initiated print requests in a 3270 terminal session are ignored. The default is true. This parameter applies to 3270 sessions only.

**Value**

`true` (Default)
`false`

**Example**

```
<param name="ignoreHostPrintRequest" value="true">
```

## indAutoPositionCursor

This parameter controls whether Reflection enters IND$FILE commands into the last unprotected field on the screen. If true (the default), Reflection sends a HOME and a BACKTAB key to move the cursor to the last unprotected field before issuing the IND$FILE command.

**Value**

`true` (Default)
`false`

**Example**

```
<param name="indAutoPositionCursor" value="true">
<param name="indAutoPositionCursor" value="false">
```

## int1InsertPads parameter

This parameter is used with UTS emulation sessions only. If Insert Pads is set to true, a Pad code is inserted into the INT1 transmit buffer before sending if the buffer is not completely filled. The value of the Pad code is 0x9F.

**Value**

`false` (Default)

`true`

**Example**

```
<param name="int1InsertPads" value="true">
<param name="int1InsertPads" value="false">
```

Top of index

# J-K-L



## jceprovider

This parameter specifies an alternative JCE provider for an SSH session. Use this parameter if you have a VT session that runs over a FIPS certified SSH connection. In this case, use the value "JsafeJCE" to invoke the RSA Jsafe crypto module. This value is case sensitive. If the JsafeJCE provider is successfully loaded, information in the Java console will indicate whether the module was loaded and if it is in FIPS-140 mode. If you do not use the correct case, the module will still load, but the client won't be able to connect, and you'll see this exception in the java console: `kSecurityFactoryNoSuchProvider?com.wrq.fipsmodule.resources.Resources`

**Value**

A valid JCE provider name. The default, "", uses JCRYPTO.

**Example**

```
<param name="jceProvider" value="JsafeJCE">
```

## launcher.keepalive

This parameter determines the behavior of the session applet when the user browses away from the page that launched the applet.

---

**NOTE:** The browse-away behavior of sessions accessed from the Reflection links list and that launch in their own window is controlled by the `launcher.keepalive` parameter for the links list applet itself, so changing the value of this parameter in the Administrative Console session management panel will not have any effect on these sessions. The `launcher.keepalive` parameter can be set for embedded sessions, which use a new launcher applet for each session.

---

If the value of launcher.keepalive is **framed** (the default when you are not using a portal for authentication), sessions that launch in their own window remain open when the user browses away from applet page. Embedded sessions are closed.

If the value is **all**, both embedded sessions and sessions launched in their own windows, remain active when the user browses away from the applet page or closes the browser window containing the applet (unless the window closed is the only browser window remaining, in which case the session is closed). Users can navigate back to the session and resume work.

A value of **portal** (the default when you are using a portal for authentication) results in the same behavior as all, with additional optimizations for a portal environment.

To close both embedded sessions and sessions launched in their own window when the user browses away from the applet page, set the launcher.keepalive parameter to **none**, and also set the parameter `legacy_lifecycle` to false. The `legacy_lifecycle` parameter is processed by both Reflection and the Java plug-in itself, and a value of `false` will allow the plug-in to terminate the applet upon browse-away

---

**CAUTION:** When `launcher.keepalive` is set to all or portal, embedded sessions remain active even after the user browses away from a session page. If the user does not log out of the session, an unauthorized user with access to the authorized user's computer could gain access to the session by clicking a session link or using the browser's Back function.

---

**Value**

`framed` (default for non-portal sessions)
`none`
`all`
`portal` (default for portal sessions

**Example**

```
<param name="launcher.keepalive" value="none">
```

## launcher.sessions

This parameter launches a session of the type specified. This parameter is case-sensitive.

`HP`
`IBM3270`
`IBM3287` (for IBM 3270 printer sessions)
`IBM5250`
`IBM3812` (for IBM 5250 printer sessions)
`IBM5250Xfer` (for AS/400 data transfer sessions)
`VT`
`UTS`
`T27`
`T27 Printer` (for T27 printer sessions)
`ALC`
`ePrint`

**Example**

```
<param name="launcher.sessions" value="IBM3270">
```

## launcher.splash

This parameter determines whether the Reflection splash screen is displayed while Reflection is loading. The splash screen includes a progress indicator, so the loading progress is not displayed when the splash screen is suppressed.

---

**NOTE:** If your session appears in its own window and users will access it via the Reflection links list, changing the value of `launcher.splash` in session management will appear to have no effect. Such sessions use the same launcher applet as the links list; because this applet has already loaded, the parameter is not applied when the session opens. You can set Launcher parameters for embedded sessions, which use a new launcher applet. To specify an embedded session, choose Display session embedded in a web browser window under Appearance in the session management panel of the Administrative Console.

---

**Value**

`true` (default)
`false`

**Example**

```
<param name="launcher.splash" value="true">
```

## loadJavaClassName

This parameter specifies the name of a Java "attachment class" to insert into the Load Java Class dialog box when it first opens. This lets you provide a default Java attachment class, without requiring users to remember the class name.

Java attachment classes are a feature of the Reflection Emulator Class Library (ECL) that allow Java code to attach to the currently running terminal session and perform automated tasks. If you write your own Java attachment class, it must be packaged into a user archive file, and the `userArchive` parameter must be used to specify the name of the archive file. If you use an attachment class built into Reflection, you do not need to add the `userArchive` parameter when specifying a startup Java class. Java attachment classes must implement the interface ECLAppletInterface.

**Value**

```
<a fully qualified Java class name>
```

**Example**

```
<param name="loadJavaClassName" value="com.mycorp.reflection.MyAutomatedTask">
```

## loadUserPrefs

This parameter determines whether user preferences are loaded when a user starts a Reflection session and a preferences file exists for that session. There is more information about user preferences in the Management and Security Server documentation, Configuring Reflection: User Preferences.

**Value**

`true` (default)
`false`

**Example**

```
<param name="loadUserPrefs" value="true">
```

Top of index

# M-N-O



## macroTimeout

This parameter specifies the number of seconds that a macro waits for the next screen during playback. If the timeout elapses without the screen being received, the macro stops and an error message displays. You can also set this value in the Macro Playback dialog box. You may need to increase the timeout if your host is slow to respond.

**Value**

`10` Default, in seconds `1-9999`

**Example**

```
<param name="macroTimeout" value="20">
```

## mayscript

This attribute allows a Java applet to access JavaScript. If an applet attempts to access JavaScript when the `mayscript` attribute is not specified, the browser generates an exception (for example, `netscape.javascript.JSException: MAYSCRIPT is not enabled for this applet`).

The`mayscript` attribute does not take a value such as true or false.

**Example**

```
<applet name="IBM3270"
        code="com.wrq.rweb.Launcher.class"
        width="400" height="300"
        mayscript>
</applet>
```

## meteringContext

This parameter specifies the web application context for the metering server. This is used in the URL that accesses the metering server, and is specified when the metering component is installed. The default, `rwebmeter`, is the correct value if you used the automated installer and have only one metering server.

The usage metering component monitors and logs connection information and queries. For more information, see Overview of Usage Metering in the Management and Security Server documentation.

**Value**

`rwebmeter` (Default)
`<descriptive string>`

**Example**

```
<param name="meteringContext" value="rwebmeter">
```

## meteringEnabled

This parameter determines whether usage metering is enabled for Reflection sessions. The usage metering component monitors and logs connection information and queries. For more information, go to Overview of Usage Metering.

When this parameter is set to true, you must specify the host on which the metering servlet resides, using the meteringHost parameter.

**Value**

`true`
`false` (default)

**Example**

```
<param name="meteringEnabled" value="false">
```

## meteringFTPenabled

This parameter determines whether usage metering is enabled for FTP connections. The usage metering component monitors and logs connection information and queries. For more information, go to Overview of Usage Metering.

When this parameter is set to true, the FTP connection is monitored. (However the `meteringFTPenabled` parameter is always ignored when meteringEnabled is set to false.)

Note that a metered FTP session does not use a second license in addition to the terminal session--licensing is based upon the number of computers connecting, not the number of connections made. However, when `meteringFTPenabled` is set to true, the FTP connection counts as an additional session, which is monitored when the `perUserLimit` parameter is enabled in the metering servlet.

**Value**

`true` (default)
`false`

**Example**

```
<param name="meteringFTPenabled" value="false">
```

## meteringHost

This parameter identifies the host on which the metering servlet resides. You can use a full host name or its full Internet Protocol (IP) address. The usage metering functionality monitors and logs connection information and queries. For more information, go to Overview of Usage Metering.

**Value**

`<host name>`
`<IP address>`

**Example**

```
<param name="meteringHost" value="myServer.example.com">
```

## meteringHostRequired

This parameter specifies whether host connections can be made when the metering servlet is unresponsive for any reason. Setting this parameter to true prevents all host connections when the metering servlet is unavailable.

When this parameter is set to true, you must specify the host on which the metering servlet resides, using the `meteringHost` parameter. The usage metering functionality monitors and logs connection information and queries. For more information, go to Overview of Usage Metering.

**Value**

`true`
`false` (default)

**Example**

```
<param name="meteringHostRequired" value="false">
```

## meteringPort

This parameter specifies the port on which the metering servlet resides. The default is 80. The usage metering functionality monitors and logs connection information and queries. For more information, go to Overview of Usage Metering.

**Value**

`80` (Default)
`<any valid port number>`

**Example**

```
<param name="meteringPort" value="80">
```

## meteringProtocol

This parameter specifies whether the metering server uses the HTTP or HTTPS protocol for connections to the client. The usage metering component monitors and logs connection information and queries. For more information, go to Overview of Usage Metering.

When the parameter is set to true, the HTTPS protocol is used; when it is set to false, HTTP is used. The HTTPS protocol provides SSL/TLS encryption and is consequently more secure, but requires that the servlet runner be SSL/TLS-enabled. If you used an automated installer and installed the default servlet runner, SSL/TLS is enabled with a self-signed certificate.

### Value

`true` (use HTTPS)
`false` (use HTTPS, default)

### Example

```
<param name="meteringProtocol" value="true">
```

## model

This parameter specifies the terminal model for the session. To use this parameter, enter only values that are compatible with the type of applet specified in the applet tag. If you do not use this parameter to specify a model, the default model for the applet is used.

### Value

| Host type | Value |
|---|---|
| IBM3270 | (E means Enhanced)<br>2E (default)<br>3E<br>4E<br>5E<br>2<br>3<br>4<br>5 |
| IBM3270 printer | 3287 |

| Host type | Value |
|---|---|
| IBM5250 | 3179-2 (Default)<br>3180-2<br>3196-A1<br>3477-FC<br>3477-FG<br>3486-BA<br>3487-HA<br>3487-HC<br>5251-11<br>5291-2 |
| IBM AS/400 printer | 3812-1 |
| IBM AS/400 data transfer | The model parameter is not used for AS/400 data transfer. |
| HP | 2392A<br>70092 (Default--includes 70096)<br>70094 (Includes 70098) |
| VT | VT52<br>VT102<br>VT400-7 (Default)<br>VT400-8 |
| UTS | UTS20<br>UTS40<br>UTS60 (Default) |
| T27 | UNISYS-TD830<br>UNISYS-TD830-ASCII<br>UNISYS-TD830-INTL<br>UNISYS-TD830-NDL (Default) |
| T27 printer | The model parameter is not used for T27 printer. |
| ALC | The model parameter is not used for ALC. |

**Example**

```
<param name="model" value="2E">
```

## name

This optional attribute specifies the name of the applet. The applet name is used by other programs and applets to reference and communicate with the applet. It is important that you assign each applet a unique name if you are running multiple applets on a page, enabling user preferences, or using the Reflection API.

Also, the value for the name attribute is displayed in the title bar of the session when the applet session is running in a separate window (the `frame` parameter is set to true) and the title parameter is not specified.

**Example**

```
<applet mayscript name="IBM3270"
        code="com.wrq.rweb.Launcher.class"
        width="400" height="300">
</applet>
```

## onExitJavaClass

This parameter specifies a Java "attachment class" to run when you select the Exit command in the File menu, you click the session's close box, or you invoke the API exit() method. iThe attachment class can then perform any "clean-up" tasks it needs to before the session is exited.

Java attachment classes are a feature of the Reflection Emulator Class Library (ECL) that allow Java code to attach to the currently running terminal session and perform automated tasks. If you write your own Java attachment class, it must be packaged into a user archive file, and the `userArchive` parameter must be used to specify the name of the archive file. If you use an attachment class built into Reflection, you do not need to add the `userArchive` parameter when specifying a startup Java class. Java attachment classes must implement the interface ECLAppletInterface.

**Value**

```
<a fully qualified Java class name>
```

**Example**

```
<param name="onExitJavaClass" value="com.mycorp.reflection.MyCleanUpClass">
<param name="onExitJavaClass"
  value="com.wrq.eNetwork.ECL.modules.PreventSessionClose">
```

---

**NOTE:** The `<param name="onExitJavaClass" value="com.wrq.eNetwork.ECL.modules.PreventSessionClose">` parameter/value combination helps to prevent a user from closing down their session prior to disconnecting. This parameter/value combo should be used only with sessions that are in their own frame.

---

## onStartupJavaClass parameter

This parameter specifies a Java "attachment class" to load immediately after the Reflection terminal session is done loading.

Java attachment classes are a feature of the Reflection Emulator Class Library (ECL) that allow Java code to attach to the currently running terminal session and perform automated tasks. If you write your own Java attachment class, it must be packaged into a user archive file, and the `userArchive` parameter must be used to specify the name of the archive file. If you use an attachment class built into Reflection, you do not need to add the `userArchive` parameter when specifying a startup Java class. Java attachment classes must implement the interface ECLAppletInterface.

**Value**

```
<a fully qualified Java class name>
```

**Example**

```
<param name="onStartupJavaClass" value="com.mycorp.reflection.MyAttachmentClass">
<param name="onStartupJavaClass"
  value="com.wrq.eNetwork.ECL.modules.JSEventNotifier">
```

Top of index

# P-Q-R



## pasteDelay

If users are having problems pasting multiple lines into host applications, use the `pasteDelay` parameter to increase the number of milliseconds the emulator waits between each line sent to the host. This parameter is valid for HP and VT terminal sessions only.

**Value**

`350` (Default--in milliseconds)`<any whole number, including 0>`

**Example**

```
<param name="pasteDelay" value="350">
```

## preloadJSAPI

This parameter enables the use of JavaScript with the Java plug-in. Setting this parameter to true causes the Reflection session to preload a minimal set of code needed by the Reflection API for executing JavaScript commands when running with the plug-in.

**Value**

`true`
`false` (default)

**Example**

```
<param name="preloadJSAPI" value="true">
```

## printerColumnsTiedToDisplay

This parameter applies to HP and VT sessions only.

When `printerColumnsTiedToDisplay` is true (the default value), the number of columns in a printout changes to match the number of columns in the display. The display columns can be read from a configuration file or user preferences file when the session loads, set in the Terminal Setup dialog box, or set by a host application using an escape sequence. If the display column setting changes during a session and you want to retain the original printer column setting, you must open the Print Setup dialog and reset printer columns.

When `printerColumnsTiedToDisplay` is false, the number of printer columns is always taken from the printer column setting and changing the display columns does not affect the printer columns.

#### Value

`true` (default)
`false`

#### Example

```
<param name="printerColumnsTiedToDisplay" value="false">
```

## printerPassthroughStripFF parameter

This parameter strips trailing form feed characters from T27 Printer data when using pass-through printing mode. When false, the form-feed characters are sent to the printer device.

#### Value

`true` (default)
`false`

#### Example

```
<param name="printerPassthroughStripFF" value="False">
```

## promptForDeviceName

This parameter specifies whether a printer session will prompt the user for a device name when a connection is requested. Because the normal behavior of printer sessions varies, the results of setting this parameter are also slightly different depending on the type of printer session involved.

In an IBM 3270 Printer session, the normal behavior is to prompt the user for a device name (`promptForDeviceName` is true). If you set `promptForDeviceName` to false, you must either save the device name in the configuration or user preference file or provide some other method for the host to locate a printer device

In an IBM AS/400 Printer session, normal behavior is not to prompt for a device name (`promptForDeviceName` is false) but to cycle through all printer devices to find one that is available. If the parameter is set to true, the session prompts the user for a device name.

#### Value

`true` (default for IBM 3270 printer session)
`false` (default for IBM AS/400 printer session)

#### Example

```
<param name="promptForDeviceName" value="true">
```

## proxyExcept

Use this parameter to specify the IP addresses or host names of machines that should not use the specified secure HTTP proxy to connect. (The secure HTTP proxy is specified with the httpsProxy parameter and is valid only for secure Reflection sessions.) If a secure HTTP proxy is not specified, this parameter is ignored.

The `httpsProxy` parameter can be used in combination with the `proxyExcept` parameter to bypass authentication schemes on HTTP proxies that are not supported in Reflection. The `httpsProxy` parameter overrides the browser settings and then the `proxyExcept` parameter overrides the setting for Reflection sessions. The Reflection session--encrypted using the Reflection security proxy server--can then pass directly through the firewall, once an opening is available.

**Value**

```
<any valid host name or IP address>
```

```
<param name="proxyExcept" value="machine1, machine2, machine3">
```

## renegotiateEcho

This parameter enables the renegotiation of an echo. Passwords are not displayed on the local screen, but all other typed text is visible. Reflection supports the Telnet Suppress Local Echo (SLE) option when connected to a host in half-duplex mode. This means that Reflection will suppress character echo to the host computer, and with SLE support Reflection can be instructed to suppress echo locally. This parameter applies to HP with Telnet and VT emulations only.

**Value**

`true`
`false` (default)

**Example**

```
<param name="renegotiateEcho" value="true">
```

## retryWithoutHTTPProxy parameter

This parameter configures secure connections that go through an HTTP proxy using an unrecognized authentication method to make a second connection attempt that bypasses the proxy. Reflection understands only Basic authentication; this parameter may be useful in situations where NTLM is being used but the HTTP proxy isn't necessary for the connection.

**Value**

`true`
`false` (default)

**Example**

```
<param name="retryWithoutHTTPProxy" value="true">
```

Top of index

# S-T-U



## secureDestHost

This parameter has been replaced by the DestinationName parameter.

## secureDestPort

This parameter has been replaced by the DestinationPortparameter.

## securityEnabled

This parameter directs the terminal session to make a secure connection to the Reflection security proxy server and port specified by the hostURL parameter (or configuration file setting).

If client authorization is enabled on the proxy server, the destination host and port can be specified using the DestinationNameand DestinationPort parameters. If client authorization is not enabled on the proxy, the ultimate destination host and port are defined by the proxy server.

If you set `securityEnabled` to true, you must also set encryptStreamto true.

#### Value

`true`
`false` (default)

#### Example

```
<param name="securityEnabled" value="true">
```

## securityEnabledFTP

This parameter directs the FTP session to make a connection through the Reflection security proxy. If client authorization is enabled on the proxy server, the destination host and port can be specified using the FTPDestinationName and FTPDestinationPort parameters. If client authorization is not enabled on the proxy, the ultimate destination host and port are defined by the proxy server.

#### Value

`true`
`false` (default)

#### Example

```
<param name="securityEnabledFTP" value="true">
```

## serverIdentityOverride

This parameter overrides the global setting Enable server identify verification for SSL/TLS connections. This option is located in the Administrative Console under Set Security Options.

If you need to have the server identity check turned on for most sessions, you can override this setting for a particular session. For example, you may wish to turn off server identity verification if an SSL/TLS direct to host connection tunnels through the security proxy and the host's certificate doesn't list the proxy as a subject alternate name.

#### Value

`default` (Use the global setting)
`true` (Perform a server identity check regardless of the global setting)
`false` (Do not perform a server identity check regardless of the global setting)

#### Example

```
<param name="serverIdentityOverride" value="true">
```

## sessionInactivityTimeout

This parameter specifies the time, in seconds, of datacomm inactivity that can elapse before the session is disconnected. Inactivity is triggered by absence of data being sent to or read from the host. If associated with a block mode application, the data may not be sent to the host until the Enter key is pressed.

#### Value

`<any integer>`

#### Example

```
<param name="sessionInactivityTimeout" value="15">
```

## sftpSendEnvironment

This parameter specifies a comma-delimited list of name-value pairs (environment variables) to send to the SSH server when an SFTP session is connected.

#### Value

A comma-delimited string of SSH environment variables, with each variable specified as "`<name>=<value>`". If a name or value itself contains an equal sign or comma, escape the equal sign or comma by preceding it with a backslash. For example, "`COMPANY=EXAMPLE\, INC`."

#### Example

```
<param name="sftpSendEnvironment" value="TERM=3270,ID=8000">
```

## shortcutMenu

This parameter determines whether the shortcut menu functionality is enabled in the terminal window. When `shortcutMenu` is set to true, the menu is displayed when a user clicks the context mouse button.

#### Value

`true` (default)
`false`

```
<param name="shortcutMenu" value="false">
```

## showHostURL

This parameter determines whether the host URL appears in the status line at the bottom of the terminal display. Any value other than true prevents the URL from appearing.

#### Value

`true` (default)
`false`

#### Example

```
<param name="showHostURL" value="false">
```

## showPrintDestination

This parameter controls whether the printer or print to file destination that is configured for a session is displayed in the status bar. Any value other than "`true`" will prevent the print destination from appearing in the status line.

#### Value

`true`

`false` (default)

**Example**

```
<param name="showPrintDestination" value="true">
```

## showStatusLine

This parameter controls whether the status line is presented. The status line appears at the bottom of the window display and includes information such as the cursor position, whether the connection is encrypted, and the type and status of the connection.

This parameter can be set for static sessions only. For dynamic sessions, use the Display status bar setting on the Manage Settings panel in the Administrative Console when creating or modifying a session.

**Value**

`true` (default)
`false`

**Example**

```
<param name="showStatusLine" value="true">
```

## SOCKSProxy

This parameter identifies a SOCKS proxy host with which terminal emulation connections are to be negotiated. The port value is optional and the default is 1080.

**Value**

```
<host> [:<port>]
```

**Example**

```
<param name="SOCKSProxy" value="myHost:1080">
```

## SOCKSProxyUserID

This parameter provides a user ID value to be used with SOCKSProxy. If unspecified (as opposed to empty) then a default ID of the system property user.name is used.

**Value**

Any string; may be empty.

**Example**

```
<param name="SOCKSProxyUserID" value="johndoe556">
```

## splash parameter

This parameter determines whether the Reflection splash screen is displayed while Reflection is loading. The splash screen includes a progress indicator, so if the splash screen is not displayed, the progress indicator is not visible.

**Value**

`true` (default)
`false`

**Example**

```
<param name="splash" value="true">
```

## sshSendEnvironment

This parameter specifies a comma-delimited list of name-value pairs (environment variables) to send to the SSH server when the SSH session is connected.

**Value**

A comma-delimited string of SSH environment variables, with each variable specified as `<name>=<value>`. If a name or value itself contains an equal sign or comma, escape the equal sign or comma by preceding it with a backslash. For example, "`COMPANY=EXAMPLE\, INC`."

```
<param name="sshSendEnvironment" value="TERM=3270,ID=8000">
```

## sslAES256

This parameter specifies whether the emulator includes or excludes AES 256-bit cipher suites when negotiating a secure connection to a host. During the negotiation, the emulator tries to use the highest level of encryption that is supported on the host, but if the host supports 256-bit encryption and the necessary policy files are not installed on the client, the connection fails. Set this parameter to false to exclude AES 256-bit from the list of cipher suites available during the SSL/TLS negotiation. AES 256-bit cipher suites are included in the SSL/TLS negotiation if this parameter is set to true or if this parameter is not used.

If the host supports 256-bit AES encryption and a 256-bit cipher suite is included on the available list on the client, you must download the Unlimited Strength Jurisdiction Policy Files. If you choose not to install the policy files, you can exclude AES 256-bit from the client list of cipher suites using this parameter.

**Value**

`true` (default)
`false`

**Example**

```
<param name="sslAES256" value="true">
```

## sslAllowInnerNullCipher

This parameter applies to the inner SSL/TLS connection in an end to end encryption through the server proxy connection. End to end encryption involves two SSL/TLS handshakes. If the host's SSL/TLS server requires a null cipher, configure this applet parameter. If set to true, TLS_RSA_WITH_NULL_SHA is added to the supported cipher suite list along with the default cipher suite.

**Value**

`true` (default)
`false`

**Example**

```
<param name="sslAllowInnerNullCipher" value="true">
```

## terminalBorder

This parameter determines whether a border is drawn around the terminal display. When the parameter is set to true, a border is drawn around the terminal display in the browser. When the parameter is set to false, no border is drawn, and the terminal display fills the area specified by the height and width attributes.

**Value**

`true` (default)
`false`

**Example**

```
<param name="terminalBorder" value="true">
```

## terminalIDResponse

This parameter determines Reflection's response to a terminal ID status request from an HP host computer. This setting does not affect how Reflection for the Web operates, but it may change the way a host program works: some applications require a given Terminal ID response. Normally, you should use the default response.

The terminalIDResponse value changes automatically when you change the **Terminal type** (Setup > Terminal > Emulation).

**Value**

```
<HP terminal ID; a string of five or fewer characters>
```

**Example**

```
<param name="terminalIDResponse" value="70092">
```

## title

Use this parameter to determine the value in the window title bar when a Reflection session runs in a separate window. If a title parameter is not specified, Reflection uses the value for the applet name attribute in the window title bar instead. The title parameter is ignored when the frame parameter is set to false.

**Value**

`<any phrase>`

**Example**

```
<param name="title" value="Reflection for IBM 3270">
```

## tn3270eConnectType

This parameter is valid for IBM 3270 printer sessions only. The tn3270eConnectType parameter specifies whether the session connects directly or whether it uses TN association to locate a printer device. For detailed information about setting up TN association using applet parameters, see tnAssociation.

**Value**

- `0` - Associate--the LU name to which users are connecting is the name of a particular printer device
- `1` (Default) - Connect--connect to a printer associated with a specific terminal LU name

**Example**

```
<param name="tn3270eConnectType" value="0">
```

## tnAssociation

This parameter is used with IBM 3270 terminal and printer sessions only. Use this parameter to associate a 3270 terminal session with a specific 3270 printer session (this feature applies to the Telnet Extended transport).

---

**NOTE:** The `tnAssociation` parameter is valid only when the `tn3270eConnectType` parameter is set to 0.

---

When associating a 3270 terminal session with a 3270 printer session, use the `tnAssociation` parameter to assign a unique TN association string in the terminal session. Reflection uses this value to automatically link the two sessions when the same value is used in the printer session for the `tnAssociation` parameter, the `tn3270eConnectType` parameter is set to 0, and the `promptForDevicename` parameter is set to false.

| Applet | Parameter | Value |
|---|---|---|
| IBM3270 | `hostURL` | `tn3270e://myhost` |
| | `tnAssociation` | `anyname` |
| IBM 3270 printer | `hostURL` | `tn3270e://myhost` |
| | `tn3270eConnectType` | `0` |
| | `tnAssociation` | `anyname` |
| | `promptForDevicename` | `false` |

For information about linking 3270 terminal and printer sessions from within a session, open the online help in the emulator session. Then, under How To, go to Using TN Association in the Print section.

Note that both the terminal session and the printer session must be running in the same type of browser (for example, Internet Explorer) for TN association to work. A terminal session running in Internet Explorer and a printer session running in Firefox cannot be associated. (However, the sessions are not required to run in the same browser window.)

**Value**

`<any association name>`

**Example**

```
<param name="tnAssociation" value="anyname">
```

## useANSIColor

This parameter applies to VT emulations only. When `useANSIColor` is true, any ANSI color attributes specified by the host overrides colors configured in the Color Setup dialog box. If `useANSIColor` is false, host color attributes are ignored and colors are selected according to the configuration in the Color Setup dialog box only.

**Value**

`true` (default)
`false`

**Example**

```
<param name="useANSIColor" value="false">
```

## userArchive

This parameter specifies the name of a user archive file containing custom Java code written to use the Reflection Emulator Class Library (ECL). This parameter is required when using the Load Java Class command in the Reflection Macro menu, the `loadJavaClass()` API method, or the `onStartupJavaClass` or `onExitJavaClass` parameters. This parameter is not required, however, when using any of the ECL modules built in to Reflection.

**Value**

`<user archive file name>`

**Example**

```
<param name="userArchive" value="MyArchive">
```

Top of index

# V-W-X-Y-Z



## visible

This parameter applies only to sessions that appear in a separate window and determines whether the display will be visible. This parameter is most useful for custom solutions where it is desirable to hide the display.

**Value**

`true` (default)
`false`

**Example**

```
<param name="visible" value="false">
```

## vthelptoc

This parameter redirects the Help Topics command in the Help menu for VT sessions to the specified URL.

**Value**

`http://<terminal emulation component installation>/help/vthelp.html` (Default)

`<any valid URL>`

**Example**

```
<param name="vthelptoc"
        value="http://ReflectionWeb.example.com
                    /customHelp/vthelp.html">
```

## width

The required width attribute controls the initial width of the session applet embedded in the browser window. The width is measured in pixels. When the frame parameter is set to true, use the `framewidth` parameter to set the width of the separate Reflection terminal window.

**Example**

```
<applet mayscript name="IBM3270"
        code="com.wrq.rweb.Launcher.class"
        width="400" height="300">
</applet>
```

## XFRFTPHostName

This parameter specifies the name of the proxy server that is used for secure FTP connections. This parameter is used in conjunction with the `securityEnabledFTP` parameter, and is valid for dynamically generated or protected static sessions only.

**Value**

The name of the security proxy server that is used for secure FTP connections.

**Example**

```
<param name="XFRFTPHostName" value="hostname.example.com">
```

## XFRFTPHostPort parameter

This parameter specifies the port number of the proxy server that is used for secure FTP connections. This parameter is used in conjunction with the `securityEnabledFTP` parameter, and is valid for dynamically generated or protected static sessions only.

**Value**

The port number used by the security proxy server through which secure FTP connections are made.

**Example**

```
<param name="XFRFTPHostPort" value="3000">
```

Top of index

# Numbers

## 3270_hostKeyboardType

This parameter allows the host keyboard type to be set for an IBM3270 session; thus, which characters can be input into a numeric-only field. The effect of each parameter value is shown in the following table.

| Value | Allowed input |
|---|---|
| `normal` | numbers and some symbols: comma (,), period (.), plus (+), and minus (-). |
| `typewriter` | All characters allowed when `value="normal"`, shifted number key symbols (such as !, @, #, and $); and uppercase letters. (Lowercase letters will appear in uppercase) |
| `dataEntry` | All characters. |

### Value

`normal` (default)
`typewriter`
`dataEntry`

### Example

```
<param name="3270_hostKeyboardType" value="normal">
```

## 3270_insertProtocol

This parameter is used with IBM 3270 emulation sessions only and specifies what Reflection does if a user attempts to insert a character.

### Value

| | |
|---|---|
| `firstNull` (default) | Reflection makes room for the character being inserted by moving all characters to the right of the insertion point one character to the right until a null is encountered. The null is replaced by a character and all subsequent characters are unchanged. If no null is found, the insertion fails. |
| `trailingSpace` | Reflection uses the same logic as for firstNull except that if no null is found it looks for a trailing space. |
| `trailingChar` | Reflection replaces the last character in the insert arena on an insert if neither a null nor a trailing space is found. |

### Example

```
<param name="3270_insertProtocol" value="firstNull">
<param name="3270_insertProtocol" value="trailingSpace">
<param name="3270_insertProtocol" value="trailingChar">
```

## 3270_keyboardErrorReset

This parameter is used with IBM 3270 emulation sessions only. Standard terminal behavior requires the user to press Reset when an error message appears in the operator information area. Setting `3270_keyboardErrorReset` to false (the default) maintains this behavior.

When `3270_keyboardErrorReset` is true, the next key pressed clears the error and restores the previous error line data. Reflection attempts to execute the keystroke as follows:

- If the cursor is in a valid input field and the key is a data key, the data is entered.
- If the cursor is in a valid input field and the key is a function key, the key operation is executed.
- If the cursor is not in a valid input field and the key is a data key, the cursor is moved to the next valid input field and the data is entered there (if the data is valid for that field).
- If the cursor is not in a valid input field and the key is a function key, the cursor is moved to the next valid input field and the key is ignored.
- If the current screen contains no valid input fields, the user hears a beep with each keystroke and no keystrokes are executed.

### Value

`true`
`false` (default)

### Example

```
<param name="3270_keyboardErrorReset" value="true">

<param name="3270_keyboardErrorReset" value="false">
```

## 3270_wordWrap parameter

This parameter is used with IBM 3270 emulation sessions only. The `3270_wordWrap` parameter specifies whether text is truncated at the end of the current line (false) or wrapped to the next available line (true) in a multi-line field.

`true`
`false` (default)

### Example

```
<param name="3270_wordWrap" value="true">

<param name="3270_wordWrap" value="false">
```

Top of index

# 4 HTML Samples

## HTML Code Samples

In Reflection for the Web, lists of session links are provided in a login applet. After a user logs in, a list of links to available sessions opens. If you want to create a custom web page for your users instead of using the provided links list, log in as a user (not as an administrator) to whom sessions are assigned, and then right-click a link to copy the URL to the clipboard. You can then paste the URL into your customized web page.



### About HTML code samples

In Reflection for the Web, you have several options for emulator session files. The **Manage Sessions** interface provides an easy way to create sessions with most of the basic functionality you need. Session files defined here are dynamically generated upon user request and provide full security and access control functionality.

You can also save a customizable form of session files in the `deploy` folder. Files in this folder cannot be edited in **Manage Sessions**, but they can be assigned to users in the **Assign Users & Groups** panel. These session files provide the same security and access control as sessions created using **Manage Sessions**.

To customize Reflection for the Web session files:

1. Use a text editor or  to create the basic HTML and applet tag for the session. If you use the , copy the HTML from the More Settings page and paste it into a text file.
2. Add custom HTML, JavaScript, or Reflection API code to the file.
3. Give the file a name with an `.html` extension and save it to `MSSData/deploy`.
4. Place any image files or other files called by the session html file in the `session` folder.

You can also save the session HTML to the `session` folder. However, files in this folder cannot be assigned to users and do not benefit from Management and Security Server's access control features.

The samples in this section provide examples of sessions that can be stored in the `deploy` folder. Modifications using HTML and JavaScript extend the functionality provided by the basic session files created using **Manage Sessions**.

---

**NOTE:** Micro Focus permits you to edit, manipulate, and change graphics, text, HTML code, JavaScript code, Java code, and other code in Reflection for the Web samples. The sample code and graphics are for use solely with deploying your internal Reflection web pages and for no other purpose. Micro Focus has no warranty, obligation, or liability for the contents of the samples.

---

### Troubleshooting Tip

The sample pages are available so that you can view the page and copy the sample code to a new HTML file. If the sample code is not formatted correctly after you paste it into your text editor, try reselecting the sample code, making sure that you extend your selection to the blank line after the end of the sample.

Some browsers, such as Internet Explorer, do not copy all of the code formatting unless an ending paragraph mark is also selected. You can ensure that the paragraph mark is included in the selection by including the blank line after the last line of code.

# Available HTML code samples

Choose from a number of detailed examples of HTML code samples for running terminal sessions and associated tasks.

More advanced functionality is available using the Reflection API with JavaScript, VBScript, or Java. See the API section for more information.

- Running a terminal session with printer emulation
- Providing access to menu commands with images and the API
- Dynamically allocating ports and assigning device names
- Disabling browser navigation

## Running a terminal session with printer emulation

You can easily run two sessions from a single page. The best way to set up this kind of multiple emulation session is to follow these steps:

1. Create the first session using the session management panel in the Administrative Console.
2. On **Configure a Web-Based Reflection Session**, click **More**. Copy the session HTML from the View HTML box.
3. Paste the HTML into a text editor.
4. Return to session management and create the second session.
5. In the View HTML box, select just the applet tag and its parameters. Copy this text. (You can also copy the entire text to the clipboard, paste it into a second text file, and then copy just the applet text from that file.)
6. Paste the second applet into the text file containing the first applet and save the results in the deploy folder as [session_name.html].

### Sample code

This sample HTML code demonstrates how to start a standard IBM 3270 session with the IBM 3270 printer running in the background. The applet launches the terminal session in a separate window, and the code is designed for Internet Explorer and Firefox browsers.

The first applet in the sample loads a 3270 host session named myHost on port 23, in the browser window. The second applet opens a 3270 printer session on a device named myDevice on port 23, running in its own window. The two sessions are associated using the `tnAssociation` parameter.

```
<html>
<head>
<title>IBM 3270 with Printer Session</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta http-equiv="charset" content="iso-8859-1">
</head>
<body>
<h3>3270</h3>
<applet mayscript name="IBM3270Frame"
   code="com.wrq.rweb.Launcher.class"
   codeBase="./ex/"
   width="600"
   height="400"
   archive="Launcher.jar">
<param name="securityEnabled" value="false">
<param name="encryptStream" value="false">
<param name="hostURL" value="tn3270e://myHost:23">
<param name="deviceName" value="myDevice">
<param name="tnAssociation" value="assocString">
<param name="autoDetect" value="true">
<param name="launcher.sessions" value="IBM3270">
</applet>
<applet mayscript name="IBM3287Device"
   code="com.wrq.rweb.Launcher.class"
   codeBase="./ex/"
   width="0"
   height="0"
   archive="Launcher.jar">
<param name="frame" value="true">
<param name="title" value="Reflection for the Web - IBM3287Device">
<param name="securityEnabled" value="false">
<param name="encryptStream" value="false">
<param name="hostURL" value="tn3270e://myHost:23">
<param name="promptForDeviceName" value="false">
<param name="tnAssociation" value="assocString">
<param name="tn3270eConnectType" value="0">
<param name="autoDetect" value="true">
<param name="launcher.sessions" value="IBM3287">
</applet>
</body>
</html>
```

## Providing access to menu commands with images and the API

This sample code demonstrates how to use images with the Reflection application programming interface (API) to provide access to specific menu functions for an embedded session. This is particularly useful when you want to restrict your user's access to certain functionality. The applet launches the terminal session embedded in the browser window (the shortcut menu is not enabled), and the code is designed for Internet Explorer and Firefox browsers.

**To start a terminal session using this sample**

1. Copy the JavaScript code and applet tag below to an HTML file. (If there are two or more applets on the page, confirm that each applet in the file has a different name.)
2. In the hostURL parameter, replace `myHost` with the name or URL of the host to which you want to connect. If you choose not to connect automatically, delete the `hostURL` and `autoconnect` parameters.
3. Save the file in the `deploy` folder.

4 Copy the images used by this page (Screen.gif, Palette.gif, and Key.gif) to the session folder. (Note that this is a different folder than the one in which the HTML file is stored.) The .gif files used in this sample are stored in the following folder of the Administrative Console: `<WebStation installation folder>\admin\en\html\advanced\sample_images\`

In addition to the images used in this sample, there are other images in the `sample_images` folder that can be used to provide user access to menu functions.

## Sample code

```
<h1>Reflection for IBM 3270 Sample</h1>
<table width="620">
<tr>
  <td align="center">
  <a
href="javascript:document.IBM3270.getAPI('JSAPI','IBM3270').showDialog('terminalCo
nfigure')"
     name="TerminalConfig"
     onMouseOver="window.status='Terminal Settings'"
     onMouseOut="window.status=''">
  <img
     src="/rweb/session/screen.gif"
     width="79" height="51" border="0"
     alt="Terminal Settings"></a>
  </td>
  <td align="center">
  <a
href="javascript:document.IBM3270.getAPI('JSAPI','IBM3270').showDialog('colorConfi
gure')"
     name="ColorConfig"
     onMouseOver="window.status='Color Settings'"
     onMouseOut="window.status=''">
  <img
     src="/rweb/session/palette.gif"
     width="79" height="51" border="0"
     alt="Color Settings"></a>
  </td>
  <td align="center">
  <a
href="javascript:document.IBM3270.getAPI('JSAPI','IBM3270').showDialog('keyMapConf
igure')"
     name="KeyboardConfig"
     onMouseOver="window.status='Map Keyboard'"
     onMouseOut="window.status=''">
  <img
     src="/rweb/session/key.gif"
     width="79" height="51" border="0"
     alt="Keyboard Mapping"></a>
  </td>
</tr>
<tr>
  <td align="center"><span style="font-family: arial, helvetica, sans-serif; color:
0000FF">Terminal Settings</span>
  </td>
  <td align="center"><span style="font-family: arial, helvetica, sans-serif; color:
0000FF">Color Settings</span>
  </td>
  <td align="center"><span style="font-family: arial, helvetica, sans-serif; color:
```

```
0000FF">Keyboard Mapping</span>
  </td>
</tr>
</table>

<applet mayscript name="IBM3270" code="com.wrq.rweb.Launcher.class"
 codeBase="./ex/" width="600" height="400" archive="Launcher.jar">
 <param name="securityEnabled" value="false">
 <param name="encryptStream" value="false">
 <param name="hostURL" value="myHost">
 <param name="shortcutMenu" value="false">
 <param name="autoDetect" value="true">
 <param name="launcher.sessions" value="IBM3270">
</applet>
```

## Dynamically allocating ports and assigning device names

**For Internet Explorer and Firefox browsers**

These two samples demonstrate how to dynamically:

- Specify a port number in a terminal session. The applet launches the session in a separate window.
- Specify a device name or pool in an IBM 3270 Printer session. For example, you could modify this code to assign a device name to specific users based on the user name. The applet launches the terminal session in a separate window. Devices are used in IBM emulation sessions only.

**To start a terminal session using this sample page**

1. Copy the sample JavaScript code below to an HTML file.
2. In the first block of JavaScript code, add custom code to determine the port number or device name you want to use. The inserted code should return the string (in quotes) of the port or device name to use.
3. In the hostURL parameter, replace myHost with the name or URL of the host to which you want to connect.
4. To specify a port number: In the launcher.sessions parameter, replace session type with the appropriate session type indicator:

| Session type | Indicator |
|---|---|
| HP | HP |
| IBM 3270 | IBM 3270 |
| IBM 3270 printer | IBM3287 |
| IBM 5250 | IBM 5250 |
| IBM 5250 printer | IBM 3812 |
| IBM AS/400 data transfer | IBM 5250Xfer |
| VT | VT |

   To specify a device name: skip to step 5.
5. Save the file in the deploy folder.

## Sample code to allocate port numbers

```
<script language="JavaScript">
<!--
// Add JavaScript code here to determine the port number.
function getPort()
{
   return "23";
}
document.writeln( '<applet mayscript name="Dynamic_Port"' );
document.writeln( ' code="com.wrq.rweb.Launcher.class"' );
document.writeln( ' codebase="./ex/"' );
document.writeln( ' width="0" height="0"' );
document.writeln( ' archive="Launcher.jar">' );
// Get the value for the port and insert it into the applet parameter.
port = getPort();
document.writeln( '<param name="hostURL" value="tn3270://MyHost:' + port + '">' );
document.writeln( '<param name="frame" value="true">' );
document.writeln( '<param name="menuType" value="advanced">' );
document.writeln( '<param name="autoDetect" value="true">');
document.writeln( '<param name="launcher.sessions" value="session type">');
document.writeln( '</applet>' );

// End of comment -->
</script>
```

## Sample code to assign device names

```
<script language="JavaScript">
<!-- Comment for non-JavaScript browsers.
// Add JavaScript code here to determine the device name.
    function getDeviceName()
    {
        return "myDevicePool";
    }

document.writeln( '<applet mayscript name="IBM3287Device"' );
document.writeln( ' code="com.wrq.rweb.Launcher.class"' );
document.writeln( ' codebase="./ex/"' );
document.writeln( ' width="0" height="0"' );
document.writeln( ' archive="Launcher.jar">' );
// Get the value for deviceName and put it into the applet parameter.
device = getDeviceName();
document.writeln( '<param name="deviceName" value="' + device + '">' );
// In the hostURL parameter, replace myHost with a valid host name or URL.
document.writeln( '<param name="hostURL" value="myHost">' );
document.writeln( '<param name="autoconnect" value="true">' );
document.writeln( '<param name="frame" value="true">' );
document.writeln( '<param name="menuType" value="advanced">' );
document.writeln( '<param name="autoDetect" value="true">');
document.writeln( '<param name="launcher.sessions" value="IBM3287">');
document.writeln( '</applet>' );

// End of comment -->
</script>
```

## Disabling browser navigation

This sample JavaScript code demonstrates how to start a standard session in a new window with the browser navigation controls disabled in either Internet Explorer or Firefox browsers.

**To start a terminal session using this sample**

1. Copy the sample JavaScript code below to an HTML file
2. In the `hostURL` parameter, replace `myHost` with the name or URL of the host to which you want to connect. If you choose not to connect automatically, delete the hostURL and autoconnect parameters.
3. Save the file in the `deploy` folder.

### Sample code

```
<script language="JavaScript">
<!-- Comment for non-JavaScript browsers.

targetwin = window.open( "", "IBM3270Applet", "status=yes, width=650, height=550"
);
targetdoc = targetwin.document;

// Use the "with (object)" statement to simplify the following lines.
with ( targetdoc ) {
    writeln( '<html>' );
    writeln( '<body>' );
    writeln( '' );
    writeln( '<h1>Reflection for IBM 3270</h1>' );
    writeln( '<applet mayscript name="IBM3270"' );
    writeln( '        code="com.wrq.rweb.Launcher.class"' );
    writeln( '        codebase="./ex/"' );
    writeln( '        width="600" height="400"' );
    writeln( '        archive="Launcher.jar">' );
    writeln( '    <param name="hostURL" value="tn3270://myHost:23">' );
    writeln( '    <param name="autoconnect" value="true">' );
    writeln( '    <param name="shortcutMenu" value="true">' );
    writeln( '    <param name="frame" value="false">' );
    writeln( '    <param name="launcher.sessions" value="IBM3270">' );
    writeln( '</applet>' );
    writeln( '' );
    writeln( '<form>' );
    writeln( '    <input type="button"  ');
    writeln( '           name="Close" ' );
    writeln( '           value="Close Window" ' );
    writeln( '           onClick="window.close()"> ');
    writeln( '</form>' );
    writeln( '' );
    writeln( '</body>' );
    writeln( '</html>' );
    close();
}

// End of comment -->
</script>
```

# 5 Host-initiated RCL Support

## RCL Commands

Reflection Command Language (RCL) is a scripting language used in older versions of Reflection to implement host-initiated commands that allow host programs to control Reflection. Reflection for the Web has implemented a subset of RCL commands to facilitate interaction with existing Reflection for HP installations.

The information presented here is intended to augment existing RCL documentation.



## Supported RCL $ Variables

Reflection for the Web supports $ expansion of RCL commands received from the host. In addition to ${0-9}, the following $ variables are supported:

| Variable | Result |
|---|---|
| $DATE | Returns the current date (MM-DD-YYYY) |
| $DAY | Returns 0 (Sunday) through 6 (Saturday) indicating the day of the week. |
| $DIR | Returns the path to the USER_HOME folder for the client machine running the HP session.1 |
| $SERIAL | Returns the Reflection for the Web serial number.<br>Example: J01-VVVL456789 (where VVV is the version number). |
| $TIME | Returns the current time (HH:MM:SS.CC). |
| $UPI | Returns the Reflection unique product identifier. |

1USER_HOME varies according to operating system, browser, and VM. To find the home folder for a given operating system/browser/virtual machine combination, look for USER_HOME in the Java console.

## Supported RCL Commands

Reflection for the Web supports the CLOSE PRINTER, OPEN PRINTER, LET, LOG, SET, and TRANSFER commands.

---

**NOTE:** Reflection for the Web ignores the QUIET and CONTINUE commands (returns a success completion code, but does nothing). Any other unrecognized RCL command will result in a failure completion code (unless completion codes are disabled or are not requested by the host).

---

#### CLOSE

The CLOSE PRINTER command closes the printer as a "to" device. CLOSE PRINTER has no effect if the printer is already closed.

#### OPEN

Reflection for the Web supports the OPEN PRINTER command.

#### LET

LET assigns a string, number, or logical value to a variable. By default, variables V0 through V9 are available and each variable can contain a 0 to 80 byte string. Strings longer than 80 bytes will be truncated.

Use the SET VARIABLES command to change the number of variables supported. Use the SET VARIABLE-LENGTH command to change the allowable variable length.

#### LOG

The LOG command sends, or logs, incoming data to the printer. Use LOG without any options to turn logging on; use LOG OFF to turn logging off.

#### SET

SET commands are used to configure Reflection. See Supported SET Parameters.

#### TRANSFER

The TRANSFER command does a type 3 block transfer to the host. Either a string or the value of a SET command can be transferred.

## Supported RCL SET Parameters

Use the alphabetical index on this page to locate SET parameters supported by Reflection for the Web. Common synonyms for parameters are shown in parenthesis.

### A-B-C

AUTO-KEYBOARD-LOCK
AUTO-LINE-FEED
AUTO-PRINT (LOG-BOTTOM)
BELL-ENABLED
BLOCK-MODE
BLOCK-TERMINATOR
BREAK-ENABLED
CAPS-LOCK
CHECK-PARITY
CONFIG-LOCKED
CURSOR-STYLE
CURSOR-VISIBLE

Top of page

## D-E-F

DESTRUCTIVE-BACKSPACE
DISABLE-COMP-CODES
DISABLE-MESSAGES
DISPLAY-COLUMNS (SCREEN-COLUMNS)
DISPLAY-CONTROL-CHARACTERS
DISPLAY-MEMORY-RESPONSE (PRIMARY-STATUS-RESPONSE)
DISPLAY-ROWS
DO-FORM-FEED
ENQ-ACK
FORMAT-MODE
FORMS-BUFFER-SIZE

Top of page

## G-H-I

HOST-PROMPT
INHIBIT-DC2
INHIBIT-EOL-WRAP
INHIBIT-HANDSHAKE
INSERT-CHARACTER

Top of Page

## J-K-L

JUMP-SCROLL-SPEED
KEYBOARD-LOCK
KEYBOARD-TYPE (NATIONAL-REPLACEMENT-SET)
LEFT-MARGIN
LIMITED-IM
MEDIATES
LINE-PAGE
LITERAL-ESCAPE
LOCAL-ECHO
LOG-BOTTOM (AUTO-PRINT)
LOG-TOP

Top of page

## M-N-O

MARGIN-BELL
MEMORY-LOCK
MODIFY-ALL
MODIFY-LINE
NATIONAL-REPLACEMENT-SET (KEYBOARD-TYPE)

Top of page

## P-Q-R

PARITY
PASTE-DELAY
PRIMARY-STATUS-RESPONSE (DISPLAY-MEMORY-RESPONSE)
RETURN-DEFINITION
RETURN-ENTER
RIGHT-MARGIN

Top of page

## S-T-U

SEND-CURSOR-POSITION
SCREEN-COLUMN
S (DISPLAY-COLUMNS)
SMOOTH-SCROLL
SPOW
START-COLUMN
TAB-SPACES
TELNET-BREAK
TELNET-LF-AFTER-CR
TELNET-NEGOTIATION
TELNET-TERMINAL
TERMINAL-ID
TERMINAL-TYPE
TRANSMIT
TRANSMIT-FUNCTIONS
TYPE-AHEAD

Top of page

## V-W-X

VARIABLES
VARIABLE-LENGTH
XFER-NULLS-TO-DEVICES

Top of page