MICRO FOCUS

# Reflection for the Web

## Reference Guide

**13.2**

# Contents

# Reflection for the Web - Reference Guide

Reflection for the Web is a standalone client that requires the Host Access Management and Security Server (MSS).

---

**NOTE:** In versions 13.2 and higher, Reflection for the Web no longer supports the use of Oracle's JRE or the Java browser plug-in. Sessions are launched via the Reflection for the Web Launcher, which uses OpenJDK 8 JRE with Web Start (JNLP).

Given these changes, the Java/JavaScript bridge (LiveConnect) feature is no longer supported. For information about using an older version, see Reflection for the Web Documentation.

---

This Reference Guide includes introductory and advanced topics for use with Reflection for the Web 13.2.

In this guide:

- How Reflection for the Web Works
- Configuring Reflection for the Web Sessions
- API and Scripting
- Applet Attributes and Parameters
- Host-initiated RCL Support

## How Reflection for the Web Works

Using Reflection for the Web and MSS, 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 the MSS Administrative Console.

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 MSS
2. The administrator uses the MSS Administrative Console 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 **Manage Sessions** in the **MSS 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)
- 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 enable you to automate routine tasks.

- The JavaScript API, or JSAPI for short, is the language used by Reflection for the Web macros, which are used to automate repetitive tasks. For more information on macros, see Recording macros. 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.

The JSAPI is the API documented in this section. API examples are shown using JavaScript; 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.

- What you need to use the API
- How to use the API
- Timing issues when scripting Reflection
- API Reference

## What you need to use the API

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 1.8 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 and Java from online resources and publications.

## 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.

## 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 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.

## 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.

- aboutBoxDialog
- apiExit
- apvuReceiveFile()
- apvuSendFile()
- autoPrintLineFeed
- buttonPaletteConfigure
- cancelPrintJob()
- colorConfigure
- connect()
- declans
- deviceName
- disconnect()
- display (string, boolean)
- emulateFormFeed
- enableHotspots

- exit()
- exportKeymap (OutputStream)
- fileTransferConfigure
- findField (int, int, int)
- findText (String, int, int, boolean)
- fitHostPageX
- fitHostPageY
- foundFieldEndColumn
- foundFieldEndRow
- foundFieldLength
- foundFieldStartColumn
- foundFieldStartRow
- foundTextStartColumn
- foundTextStartRow
- ftpCd (String)
- ftpConfigure
- ftpDefaultFolders
- ftpDisconnect ()
- ftpGetLastServerResponse()
- ftpLCd (String)
- 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)
```

# 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
- Applet Attributes and Parameters
- Index of Attributes and Parameters

## 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. 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.

You can change emulator session behavior. Use **Manage Sessions** to add session-specific parameters to the session applet tag.

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**.

**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.

**Case sensitivity.** **Attribute values** are case-sensitive; all other attributes and parameters are not — with one exception. A **parameter value** that references a file or Java class 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

- Configuration
- crlIssuers

## 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).

---

**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 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 sessions. 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 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 sessions. 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 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**

`https://www.microfocus.com/support-and-services/` (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**

`https://www.microfocus.com/products/reflection/web/?utm_medium=301&utm_source=attachmate.com` (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 sessions. 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 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 MSS, a heartbeat is automatically sent to the server every 3 minutes to keep the MSS 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 make backup copies of both the original and modified login pages because any modifications will be overwritten by product upgrades.

1. In your login page, locate the section that begins with `<mss:applet userequest="true"/>`.
2. Add the heartbeat parameters you would like to set.

## Heartbeat parameters

You can add these parameters to the MSS 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 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> |
| 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> |

| Host | Values |
|---|---|
| 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> |
| 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> |

| Host | Values |
|---|---|
| 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.

---

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

If the value is **all**, sessions 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 sessions 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

**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.

**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

- meteringFTPenabled
- meteringHost
- meteringHostRequired
- meteringPort
- meteringProtocol
- model
- name
- onExitJavaClass
- onStartupJavaClass parameter

## 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) |
| | 2E (default) |
| | 3E |
| | 4E |
| | 5E |
| | 2 |
| | 3 |
| | 4 |
| | 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">
```

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 only for static sessions, which are no longer supported . 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 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 sessions.

**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 sessions.

**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 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
- Supported RCL Commands
- Supported RCL SET Parameters

### 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 | D-E-F | G-H-I | J-K-L | M-N-O | P-Q-R | S-T-U | V-W-X

### 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

Supported RCL SET Parameters

## 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

Supported RCL SET Parameters

## G-H-I

HOST-PROMPT
INHIBIT-DC2
INHIBIT-EOL-WRAP
INHIBIT-HANDSHAKE
INSERT-CHARACTER

Supported RCL SET Parameters

## 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

Supported RCL SET Parameters

## M-N-O

MARGIN-BELL
MEMORY-LOCK
MODIFY-ALL
MODIFY-LINE
NATIONAL-REPLACEMENT-SET (KEYBOARD-TYPE)

Supported RCL SET Parameters

## P-Q-R

PARITY
PASTE-DELAY
PRIMARY-STATUS-RESPONSE (DISPLAY-MEMORY-RESPONSE)
RETURN-DEFINITION
RETURN-ENTER
RIGHT-MARGIN

Supported RCL SET Parameters

## 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

Supported RCL SET Parameters

## V-W-X

VARIABLES

VARIABLE-LENGTH
XFER-NULLS-TO-DEVICES

Supported RCL SET Parameters