SecureLogin 8.7

Application Definition Guide

December, 2018

NetIQ

Legal Notice

Contents

About This Guide

This guide helps users to write or modify application definitions for single sign-on-enabled applications. Most users will find it quicker and easier to use the Application Definition Wizard but, assuming the relevant permissions have been granted, users may also write their own application definitions to suit their particular requirements.

Additional Documentation

For the latest version of SecureLogin guides, see www.netiq.com/documentation/securelogin/

Contacting Sales Support

For questions about products, pricing, and capabilities, contact your local partner. If you cannot contact your partner, contact our Sales Support team.

Worldwide:	www.netiq.com/about_netiq/officelocations.asp
United States and Canada:	1-888-323-6768
Email:	info@netiq.com
Web Site:	www.netiq.com

Contacting Technical Support

For specific product issues, contact our Technical Support team.

Worldwide:	www.netiq.com/support/contactinfo.asp
North and South America:	1-713-418-5555
Europe, Middle East, and Africa:	+353 (0) 91-782 677
Email:	support@netiq.com
Web Site:	www.netiq.com/support

Contacting Documentation Support

Our goal is to provide documentation that meets your needs. If you have suggestions for improvements, click **Add Comment** at the bottom of any page in the HTML versions of the documentation posted at www.netiq.com/documentation. You can also email Documentation-Feedback@netiq.com. We value your input and look forward to hearing from you.

Contacting the Online User Community

Qmunity, the NetIQ online community, is a collaborative network connecting you to your peers and NetIQ experts. By providing more immediate information, useful links to helpful resources, and access to NetIQ experts, Qmunity helps ensure you are mastering the knowledge you need to realize the full potential of IT investments upon which you rely. For more information, visit http://community.netiq.com.

1 Application Definition Language: an Overview

The capability of SecureLogin to create proprietary application definitions is a powerful feature. This application definition command language facilitates single sign-on of all types of applications.

SecureLogin implements application definition commands to provide a flexible single sign-on and monitoring environment. For example, the SecureLogin Windows Agent watches for application login boxes. When a login box is identified, the agent runs an application definition to enter the username, password, and background authentication information.

This section contains the following information:

What is an Application Definition?

An application definition is essentially a list of instructions that SecureLogin follows in order to perform various tasks on various windows. For example, for a Windows application (`*.exe`), an application definition is written for each executable file that you want SecureLogin to act upon. In that application definition, you are able to assign different instructions to each dialog box or screen that the executable file or application might produce. By doing this, you have the choice of acting upon only the login panel, only selected windows, or every window that is produced by the executable file, such as account locked, invalid username, invalid password, back-end database is down, password expiry, and so on.

SecureLogin processes the application definition from left to right, top to bottom. However, with the use of flow control commands, such as `Call`, it is possible to skip, repeat, or jump to certain parts of the application definition.

Advantages of Using Application Definitions

- Enables you to single sign-on enable almost any Windows, mainframe, internet, intranet, terminal server, or Unix application.
- No need to install software on your application servers.
- The flexibility for you and your application owners to choose what to do once an application generated message is detected, giving you full control over your single sign-on environment.

- Allows more sophisticated single sign-on to supported applications, including the ability to seamlessly handle several versions of one application. This feature is especially important when you upgrade your applications.
- Security. SecureLogin data (for example, user credentials) is stored and protected in the directory.
- Speed. When SecureLogin is started, it locates user data in the directory and caches its encrypted contents in memory (and optionally on disk) for later use by the workstation's SecureLogin agent.

Using Application Definitions

You can use application definitions to:

- Execute the retrieval and entering of correct login details. Application definitions are stored and secured within the directory to ensure maximum security, support for single-point administration, and manageability.
- Automate many login processes, such as multi-page login and login panels requiring other information that you can store in the directory (such as surname or telephone number).
- Application definitions can include commands to automate password changes on behalf of users and to request user input when required.
- Application definitions can accommodate error handling that is generated by the back-end application. For example, handling of invalid logins.

Defining Applications Enabled for Single Sign-On

SecureLogin provides the option to define which applications are enabled for single sign-on. This option gives you:

- Complete control for deciding which applications need to be enabled for single sign-on.
- The ability to update the entire directory database with a new application login application definition by updating a single object.

Corporate Definitions

Corporate applications allow scripts to flow down to all users located within a container, allowing central administrators and maintenance of the script.

Corporate application definitions are stored in a container object rather than on the individual user objects. For users, the result is a less complex system.

For you as the administrator, the improved login mechanisms provide the following:

- A greater level of accountability with increased productivity and security.
- A reduced workload at the help desk because of significantly fewer password resets.

Using Dialog Specifier Commands

Using the `Dialog Specifier` commands, you can assign individual sections of an application definition to the different windows an executable file produces. This allows the login dialog box, for example, to be treated differently from the Error Message box and so on.

There are many commands in the SecureLogin application definition language. Some of the SecureLogin commands such as `Repeat` and `Dialog`, have one or two commands that are used to close them.

Reading from and Writing to Variables

Application definition commands can read from and write to variables. These variables enable SecureLogin to use corporate application definitions, while each individual user's secrets are securely stored in the directory. It is also possible to read attributes, such as the user's full name and phone number, from attributes in the directory.

SecureLogin not only writes information to the screen, but also reads from it with the use of commands such as `ReadText`. Use this to extract usernames, domains in use, error messages, and other useful information. Use `Variable Manipulator` commands to perform calculations, break apart information, and join it back together again.

All these features come together to form an extremely powerful language that is able to accomplish almost any task that is required.

Using Characters Interpretable by SecureLogin

Using interpretable characters in SecureLogin application definitions has implications for definitions that are created in, or copied from, and pasted from a Microsoft Word.

For example, when you are writing an application definition that requires a “`-`” (dash) in the command syntax, make sure you use a short "–" or en dash (Unicode glyph U+2013 (Hex) or 8211(Decimal) and cannot be an extended "—" or em dash as generated in Microsoft Word.

In Microsoft Word, when you type a space and one or two hyphens between text, Microsoft Word automatically inserts an ASCII dash or en dash (–). If you type two hyphens and do not include a space before the hyphens, an em dash (—) is created.

Similarly, when you are writing an application definition that requires quotation mark in the command syntax, make sure you use a straight quotation mark (Unicode glyph U+0022 (Hex) or 0034 (Decimal) or the ASCII printable character 34). For quotation mark syntax example, see “Quotation Marks” on page 46.

In Microsoft Word, when you type a question mark, Word automatically changes straight quotation marks to curly (or smart) quotes, as you type unless the Word **AutoCorrect**, **AutoFormat As You Type** features are disabled.

2 Command Quick Reference

Application Definition Command Quick Reference

Table 2-1 *Application Definition Command Quick Reference*

Command	What it means?
#	Use the this symbol to define a line of text as a comment field. Comment fields are used to leave notes. For more information, see **“#” on page 38**
" "	Use quotation marks to group together text or variables containing spaces. Quotation marks are used with commands like `Type`, `MessageBox`, and `If -Text`. For more information, see **“" "” on page 38**
$	Use the dollar sign to define the use of a SecureLogin variable stored in the directory for later use by that user. For more information, see **“$” on page 13**
?	Use the question mark to define the use of a runtime variable. The values of these variables are not stored in the directory. They are reset each time SecureLogin is started. For more information, see **“?” on page 38**
%	Use the percentage sign to define the use of a directory attribute. The attributes that are available vary depending on the directory in use, and the setup of the directory. For more information, see **“%” on page 39**
****	Use the backslash with the `Type` and `Send Key` commands to specify the use of a special function.
@	Use this symbol in the same way as the backslash symbol, except its use is limited to HLLAPI enabled emulators. For more information, see **“@” on page 39**
-	Use the hyphen as a switch within several commands, such as `If` and `Type`. For more information, see **“-” on page 39**
AAVerify	It is typically used before the application Username and Password are retrieved and entered into the login box. For more information, “AAVerify” on page 59

Command	What it means?
Add	Adds one number to another. The numbers can be hard-coded into the application definition, or they can be variables. The result can be the output of another variable, or one of the original numbers. For more information, see “Add” on page 62
Attribute	Use the `Attribute` specifier in conjunction with the `Tag/EndTag` command to specify which HTML attributes and attribute values must exist for that particular HTML tag. For more information, see “Attribute” on page 63
AuditEvent	Use the `AuditEvent` to audit the following events from an application definition: ◆ SecureLogin client started ◆ SecureLogin client exited ◆ SecureLogin client activated by user ◆ SecureLogin client deactivated by user ◆ Password provided to an application by a script ◆ Password changed by the user in response to a `changepassword` command ◆ Password changed automatically in response to a `changepassword` command For more information, see “AuditEvent” on page 64
BeginSplashScreen/ EndSplashScreen	Use to display a splash screen across the whole Terminal Emulator window. This is used to mask any flashing produced by SecureLogin scraping the screen for text. A `Delay` command at the start of the application definition ensures that the emulator window is in place before the splash screen is displayed. For more information, see “BeginSplashScreen/ EndSplashScreen” on page 64
BooleanInput	Use `BooleanInput` within a `site` block to set the state of a Boolean field (either a check box or radio button). For more information, see “BooleanInput” on page 65
Break	Use `Break` within the `Repeat/EndRepeat` commands to break out of a repeat loop. For more information, see “Break” on page 66
Call	Use the `Call` command to call and run a subroutine. When a subroutine is called, the application definition begins executing from the first line of the subroutine. For more information, see “Call” on page 67
ChangePassword	Use the `ChangePassword` command to change a single variable and is used in scenarios where password expiry is an issue. Set the `<Variable>` to the new password. For more information, see “ChangePassword” on page 68

Command	What it means?
Class	When a window is created, it is based on a template known as a window class. The `Class` command checks to see if the class of the newly created window matches its `<Window-Class>` argument. For more information, see “Class” on page 70
ClearPlat	Use to reset the last chosen platform, causing subsequent calls to ReLoadPlat to do nothing. For more information, see “ClearPlat” on page 70
ClearSite	Use within a `Site` block to clear the ‘matched’ status for a given site. For more information, see “ClearSite” on page 72
Click	When used with windows applications, the `Click` command sends a click instruction to the specified `<#Ctrl-ID>`. For more information, see “Click” on page 73
ConvertTime	Use to convert a numeric time value, for example, `?CurrTime(system)`, into a legible format and store it in `<String Time>`. For more information, see “ConvertTime” on page 77
Ctrl	Use the `Ctrl` command to determine if a window contains the control expressed in the `<#Ctrl-ID>` argument. The control ID number is a constant that is established at the time a program is compiled. For more information, see “Ctrl” on page 77
DebugPrint	Use the `DebugPrint` command to display the text specified in the `<Data>` variable on a Debug console.The command can take any number of text arguments, including variables, (for example, `DebugPrint "The user " $Username " has just been logged onto the system"`). For more information, see “DebugPrint” on page 78
Decrement	Use the `Decrement` command to subtract from a specified variable. For example, you can use `Decrement` to count the number of passes a particular application definition has made. For more information, see “Decrement” on page 79
Delay	Use the `Delay` command to delay the execution of the application definition for the time specified in the `<Time Period>` argument. For more information, see “Delay” on page 80
Dialog/EndDialog	Use the `Dialog/EndDialog` command to identify the beginning and end of a dialog specification block respectively. You can use these commands to construct a dialog specification block, which consists of a series of dialog specification statements (for example Ctrl and Title). For more information, see “Dialog/EndDialog” on page 81

Command	What it means?
DisplayVariables	Use the `DisplayVariables` command to display a dialog box that lists the user's stored variables (for example, $Username and $Password) for the current application. For more information, see “DisplayVariables” on page 82
Divide	Use to divide one number by another. The numbers can be hard coded into the application definition, or they can be variables. The result can be output to another variable, or to one of the original numbers. For more information, see “Divide” on page 84
DumpPage	Use the `DumpPage` command to provide information about the current Web page. Use for debugging Web page application definitions. For more information, see “DumpPage” on page 85
EndScript	Use the `EndScript` command to immediately terminate execution of the application definition. For more information, see “EndScript” on page 85
Event/Event Specifiers	Application definitions generally execute at the point when an application window is created. This corresponds to the `WM_CREATE` message received from an application window at startup. By adding the Event Specifier to a dialog block, you can override this behavior whereby application definition executes only when the specified message is generated. If an Event Specifier is not givem, it is treated as the same as Event WM_CREATE. For more information, see “Event/Event Specifiers” on page 86
FocusInput	Use within a `Site` Block to focus on an input field based on the Boolean value of `"focus"`. For more information, see “FocusInput” on page 87
GenerateOTP	Used to generate a one time password (OTP) as an authentication method in lieu of a traditional fixed and static password. For more information, see “GenerateOTP” on page 87
GetCheckBoxState	Use the `GetCheckBoxState` command to return the current state of the specified checkbooks. For more information, see “GetCheckBoxState” on page 90
GetCommandLine	Use the `GetCommandLine` command to capture the full command line of the program that is loaded, and save it to the specified variable. For more information, see “GetCommandLine” on page 91
GetEnv	Use the `GetEnv` command to read the value of an environment variable and save it in the specified `<variable>`. For more information, see “GetEnv” on page 91

Command	What it means?
GetHandle	Use `GetHandle` to capture the unique handle of the window on which the Windows application definition script is activated. For more information, see “GetHandle” on page 92
GetIni	Use the `GetIni` command to read data from the `INI` file. For more information, see “GetIni” on page 93
GetMD5	Use the `GetMD5` command to generate an MD5 hash value of the current process the script is running for. `GetMD5` works only with the Win32 scripts. For more information, see “GetMD5” on page 93
GetReg	Use the `GetReg` command to read data from the registry and save it in the specified `<variable>`. For more information, see “GetReg” on page 94
GetDirectoryStatus	Use the GetDirectoryStatus command to find out whether SecureLogin can connect to the directory or not. For more information, see “GetDirectoryStatus” on page 95
GetSessionName	Use the `GetSessionName` command to find the current HLLAPI session name that is used to connect and return it to the specified variable. For more information, see “GetSessionName” on page 96
GetText	Use the `GetText` command to get all of the text from the screen and save it to the specified variable. It is used in a large Web application definition that might contain several If -Text statements. For more information, see “GetText” on page 96
GetURL	Use the `GetURL` command to capture the URL of the site that is loaded and save it to the specified variable. For more information, see “GetURL” on page 97
GoToURL	Use the `GoToURL` command to make the browser navigate to the specified <URL>. By default the command opens the new Web page in the main window, rather than the frame that started the application definition. For more information, see “GoToURL” on page 98
If/Else/EndIf	Use the `If` command to establish a block to execute if the expression supplied is true. The `Else` command works inside an If block. The `Else` command is executed if the operator in the If block is false. Use the `EndIf` command to terminate the `If` block. For more information, see “If/Else/EndIf” on page 99

Command	What it means?
Include	Use the `Include` command to share commonly used application definition commands by multiple applications. The application definition identified by <Platform-Name> is included at execution time into the calling application definition. The application definition included with the `Include` command must consists of commands supported by the calling application. For more information, see “Include” on page 102
Increment	Use the `Increment` command to add to a specified variable. For example, you can use increment to count the number of passes a particular application definition has made. For more information, see “Increment” on page 103
KillApp	Use to terminate an application. For more information, see “KillApp” on page 104
Local	Use the `Local` command to declare that a runtime variable will only exist for the lifetime of the application definition. Local runtime variables are used in the same way as normal runtime variables and are still written as `?Variable`. For more information, see “Local” on page 106
MatchDomain	Use `MatchDomain` inside a site block to filter a site based on its domain. If the domain does not match, the site block fails to match. For more information, see “MatchDomain” on page 107
MatchField	Use `MatchField` to filter a form based on the presence of a particular field. If the field fails to match and it is not specified as optional, then the parent form fails to match. For more information, see “MatchField” on page 108
MatchForm	Use `MatchForm` to filter a site based on the presence of a particular field. If the field fails to match and it is not specified as optional, then the site fails to match. For more information, see “MatchForm” on page 110.
MatchOption	Use the `MatchOption` command to filter a field based on the presence of a particular option. For more information, see “MatchOption” on page 112.
MatchReferer	Use `MatchReferer` inside a `Site/EndSite` block to match or filter a site based on a referrer. For more information, see “MatchReferer” on page 114.
MatchRegex	Use the `MatchRegex` command to test whether a regular expression matches against the specified string or not. You can also use it inside a Site–EndSite or Dialog–EndDialog block for example. For more informations, see “MatchRegex” on page 115.

Command	What it means?
MatchTitle	Used inside a site block, `MatchTitle` is used to filter a site based on its title. If the site title does not match, the site block fails to match. For more information, see “MatchTitle” on page 116.
MatchURL	Use `MatchURL` inside a site block to match or filter an HTML page within a site based on its URL. The URL can be a complex Web address or a secure Web site. For more information, see “MatchURL” on page 117.
MessageBox	Use the `MessageBox` command to display a dialog box that contains the text specified in the `<Data>` variable. The application definition is suspended until the user reacts to this message. The `MessageBox` can take any number of text arguments, including variables, (for example MessageBox "The user " $Username " has just been logged onto the system"). For more information, see “MessageBox” on page 118.
Multiply	Use to multiply one number by another. You can hard-code the numbers into the application definition, or you can use variables. The results can be output to another variable, or to one of the original numbers. For more information, see “Multiply” on page 120.
OnException/ ClearException	Use the `OnException` command to detect when certain conditions are met. Currently, this is when **Cancel** is clicked on either of two dialog boxes. When the condition is met, a subroutine is run. Use the `ClearException` command to reset the exceptions value. For more information, see “OnException/ClearException” on page 121.
Parent/EndParent	Use the `EndParent` command to terminate a `Parent` block and set the subject of the application definition back to the original window. You can nest the `Parent` command, thereby allowing the `Parent` block to act on the parent of the parent. For more information, see “Parent/EndParent” on page 126.
PickListAdd	Use the `PickList` command to allow users with multiple accounts for a particular system to choose the account to which they will log in. For more information, see “PickListAdd” on page 128.
PickListDisplay	Use the `PickListDisplay` command to display the pick list entries built by previous calls to `PickListAdd`. The `PickListDisplay` command returns the result in a *<?Variable>* sent to the command. For more information, see “PickListDisplay” on page 129.

Command	What it means?
PositionCharacter	Use the `PositionCharacter` command in a password policy application definition to enforce that a certain character in the password is a numeral, uppercase, lowercase, or a punctuation character. For more information, see “PositionCharacter” on page 130.
PressInput	Used within a site block to simulate a keyboard enter event. For more information, see “PressInput” on page 131.
ReadText	Use the `ReadText` command to run in both Windows and Terminal Launcher application definitions. Although the usage and arguments for the use of ReadText with Windows and Terminal Launcher are different, the results of each command are the same. For more information, see “ReadText” on page 133.
RegSplit	Use the `RegSplit` command to split a string by using a regular expression. `<Output-String1>` and `<Output-String2>` contain the first and second subexpressions. For more information, see “RegSplit” on page 136.
ReLoadPlat	Use to set the current platform to the last one chosen by the application definition, or if a platform is not chosen, leaves the platform unset. For more information, see “ReLoadPlat” on page 138.
Repeat/EndRepeat	Use the `Repeat` command to establish an application definition block similar to the `If` command. The `repeat` block is terminated by an `EndRepeat` command. Alternatively, you can use the `Break` or `EndScript` commands to break out of the loop. For more information, see “Repeat/EndRepeat” on page 141.
RestrictVariable	Use the `RestrictVariable` command to monitor a `<Variable>` and enforce a specified `<Password-Policy>` on the `<Variable>`. Any variable specified must match the policy or it is not saved. For more information, see “RestrictVariable” on page 142.
Run	Use the `Run` command to launch the program specified in `<Command>` with the specified optional `[<Arg1> [<Arg2>] …]` arguments. For more information, see “Run” on page 145.
Select	Use the `Select` command to select entries from a combo box or list box control. For more information, see “Select” on page 147.
SelectListBoxItem	Use the `SelectListBoxItem` command to select entries from a list box. For more information, see “SelectListBoxItem” on page 148.

Command	What it means?
SelectOption	Use the `SelectOption` command to select or deselect options within a list box or combo dialog box. For more information, see “SelectOption” on page 149.
SendEvent	Use the `SendEvent` command to broadcast events. For more information, see “SendEvent” on page 149
SendKey	Use the `SendKey` command to work only with Generic and Advanced Generic emulators. You can use the `SendKey` command in the same manner as the `Type` command. Generally, the `Type` command is the preferred command to use. The `Type` command places the text into the clipboard, and then pastes it into the emulator screen. The `SendKey` command enters the text directly into the emulator screen. For more information, see “SendKey” on page 150.
Set	Use the `Set` command to copy the value of `<Data>` into `<Variable>`. The `<Data>` can be any text, or another variable, whereas the `<Variable>` must be either a ?Variable or $Variable. For more information, see “Set” on page 151.
SetCheckBox	Use the `SetCheckBox` command to select or clear a check box. For more information, see “SetCheckBox” on page 152.
SetCursor	Use the `SetCursor` command to set the cursor to a specified `<ScreenPosition>` or `<X Co-ordinate> <Y Co-ordinate>`. For more information, see “SetCursor” on page 153.
SetFocus	Use the `SetFocus` command to set the keyboard focus to a specified <#Ctrl-ID>. For more information, see “SetFocus” on page 154.
SetPlat	`SetPlat` sets the platform or application from which variables are read and saved. By default, variables are stored directly against the platform or application on which you have SecureLogin enabled. For example, if you enable `Groupwise.exe`, the Groupwise credentials are stored against the `Groupwise.exe` platform. `SetPlat` sets the platform or application from which variables are read and saved. For more information, see “SetPlat” on page 155.
SetPrompt	Use the `SetPrompt` command to customize the text in the Enter SecureLogin Variables dialog boxes. These dialog boxes are used to prompt the user for new variables. You can also use the `DisplayVariables` command to customize the prompt text in the dialog box (for previously stored variables). For more information, see “SetPrompt” on page 158.

Command	What it means?
Site/Endsite	`Site/Endsite` are Web commands added to allow for finer control of site matching. More detailed information within a loaded Web site can now be matched upon an used to execute blocks of scripting commands. Begins and ends an application definition, in place of `Dialog/EndDialog`. For more information, see “Site/Endsite” on page 160.
StrCat	Use the `StrCat` command to append a second data string to the first data string. For example, `StrCat ?Result "SecureRemote " "$Username"`. For more information, see “StrCat” on page 162.
StrLength	Use the `StrLength` command to count the number of characters in a variable and output that value to the destination variable. For more information, see “StrLength” on page 163.
StrLower	Use the `StrLower` command to modify a variable so that all the characters are lowercase. For more information, see “StrLower” on page 164.
StrUpper	Use the `StrUpper` command to modify a variable so that all the characters are uppercase. For more information, see “StrUpper” on page 167.
Sub/EndSub	Use the `Sub/EndSub` commands around a block of lines within an application definition to denote a subroutine. For more information, see “Sub/EndSub” on page 167,
Submit	Use the `Submit` command only in Web application definitions, and only with Internet Explorer, to allow for enhanced control of how and when a form is submitted. The `Submit` command performs a Submit on the form in which the first password field is found. The `Submit` command is ignored if used with Netscape. For more information, see “Submit” on page 168.
Substr	Use the `Substr` command to search for a sub string from a text based on the index and the length which are provided as parameters. For more information, see “Substr” on page 169.
Subtract	Use the `Subtract` command to subtract one value from another. This is useful if you are implementing periodic password change functionality for an application. You can use the `subtract` command (in conjunction with the Divide function and the `Slina DLL`) to determine the number of days that have elapsed since the last password change. Other numeric commands include `Add`, `Divide`, and `Multiply`. For more information, see “Subtract” on page 171.
Tag/EndTag	Use the `Tag/EndTag` commands to find HTML tags. For more information, see “Tag/EndTag” on page 172.

Command	What it means?
TextInput	Use within a site block to input text into a special field. For more information, see “TextInput” on page 173.
Title	Use the `Title` command to retrieve the title of a window and compare it against the string specified in the `<Window-Title>` argument. For this block of the application definition to run, the retrieved window title and the <*Window-Title*> argument must match the text supplied to the `Title` command in the dialog block. For more information, see “Title” on page 173.
Type	Use the `Type` command to enter data, such as usernames and passwords, into applications. There are reserved character sequences that are used to type special characters, for example TAB and ENTER. If it is not possible to determine Control IDs in a Windows application, and the `Type` command is not working, use the `SendKey` command instead. For more information, see “Type” on page 175.
WaitForFocus	Use the `WaitForFocus` command to suspend the running of the application definition until the `<#Ctrl-ID>` has received keyboard focus, or the `<Repeat-Loops>` expire. The `<Repeat-Loops>` is an optional value that defines the number of loop cycles to run. The `<Repeat-Loops>` value defaults to 3000 loops if nothing is set. After focus is received, the application definition continues. For more information, see “WaitForFocus” on page 181.
WaitForText	Use the `WaitForText` command so the Terminal Launcher waits for the specified `<text>` to display before continuing. For example, the user waits for a username field to display before attempting to type a username. For more information, see “WaitForText” on page 182.

3 Managing Application Definitions

Application definitions are generally imported, built, or modified in the Management utility of SecureLogin, tested locally, and then copied to the relevant container, or the organizational unit in multi-user directory environments. Application definitions are imported and exported in the XML file format for ease of distribution and deployment.

SecureLogin application definitions can be created using the application definition wizard.

For more information about the Application Definition Wizard, see the *NetIQ SecureLogin Application Definition Wizard Administration Guide* .

Application Definition Checklist

When you have built or modified your application definitions, it is recommended that you test each supported application or the Web page for the following scenarios:

- Entering a correct username or password.
- Entering an incorrect username or password.
- Cancelling a login by the user.
- Exceeding maximum password retries.
- A user changing his or her own password.
- Attempting to change to an illegal password.

  This illegal password action is relevant when you define a password policy and you try to define a password that does not match the policy.
- An administrator cancelling a password change.
- An administrator changing a user password.
- Expiry of user password.
- Locking out the account.
- Locking out someone from the account.

Exporting and Importing Predefined Applications and Application Definitions

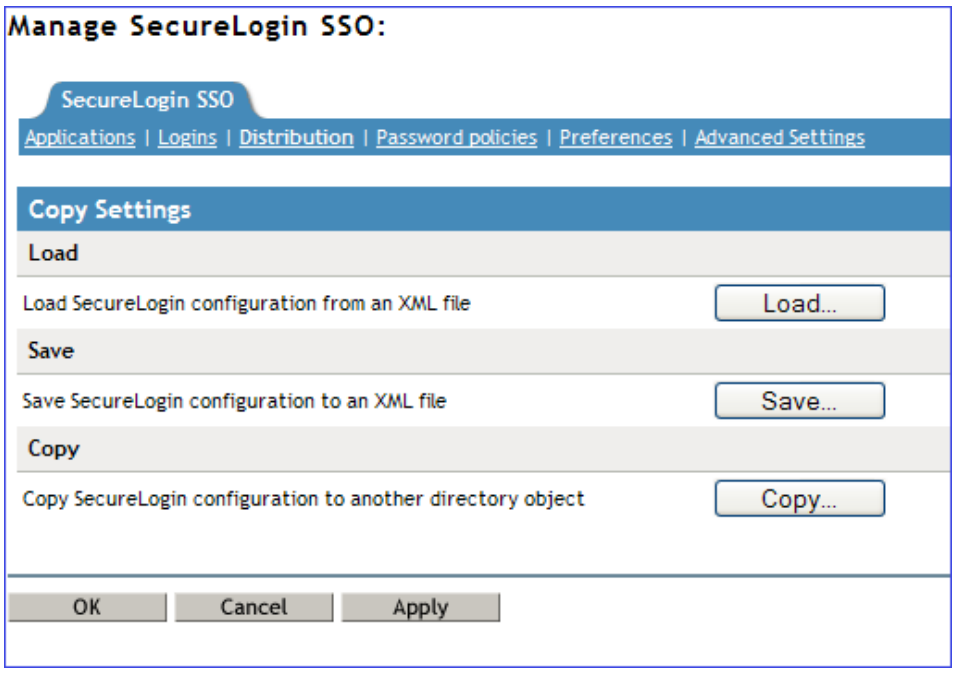
SecureLogin provides export functionality to facilitate distribution of predefined applications and application definitions. Converting predefined applications and application definitions to XML format allows you to distribute and deploy predefined applications and application definitions across directories, software, and hardware platforms.

This section contains the following information:

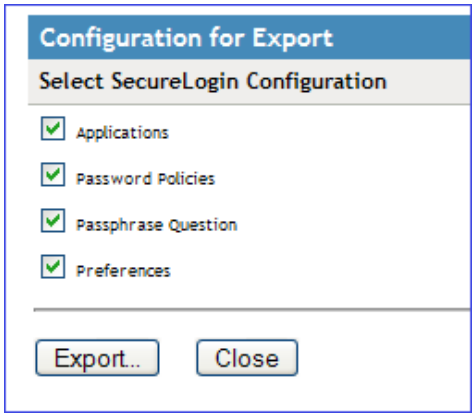
- “Exporting Individual Applications” on page 26
- “Importing Individual Applications” on page 27

Exporting Individual Applications

1. Log in to iManager.
2. Select **Securelogin SSO > Manage Securelogin SSO**. The Manage SecureLogin SSO page is displayed.
3. In the object field, specify your object name, then click **OK**.
4. Click **Distribution**. The distribution details are displayed.



5. Click **Save**. The Configuration for Export dialog box is displayed.
6. Under **Select SecureLogin Configuration**, select the appropriate text boxes.



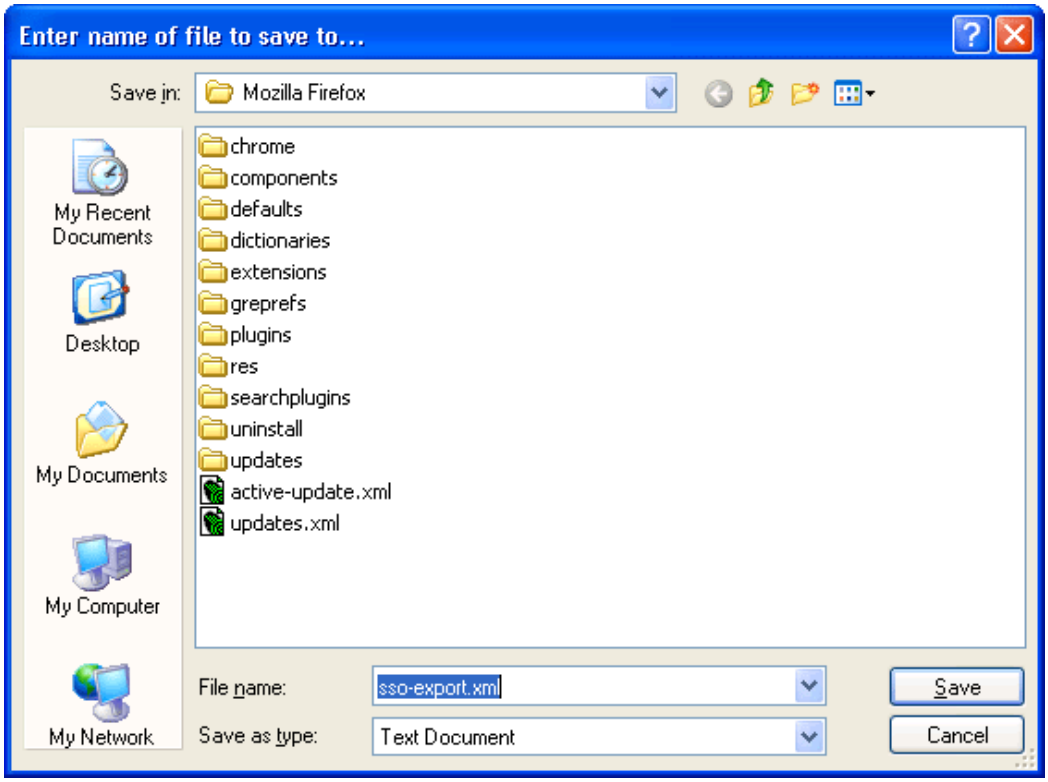
Configuration	Function
Application	Copies, exports, or imports all configured application definitions as displayed in the **Application** pane.
Credentials	Copies, exports, or imports all credentials as displayed in the **Logins** pane, excluding passwords for copy settings and unencrypted export or import.
Password Policies	Copies, exports, or imports password policies as displayed in the **Password Policies Properties** table.
Preferences	Copies, exports, or imports preferences manually set in the **Preferences Properties** tables.

7 Click **Export**. The Select the Applications for Backup page is displayed.

8 Select the applications you want to backup.

9 Click **OK**. The Save File As dialog box is displayed.

10 Provide a name to the file, select the file location, and click **Save**.



NOTE: The file is saved in an XML format.

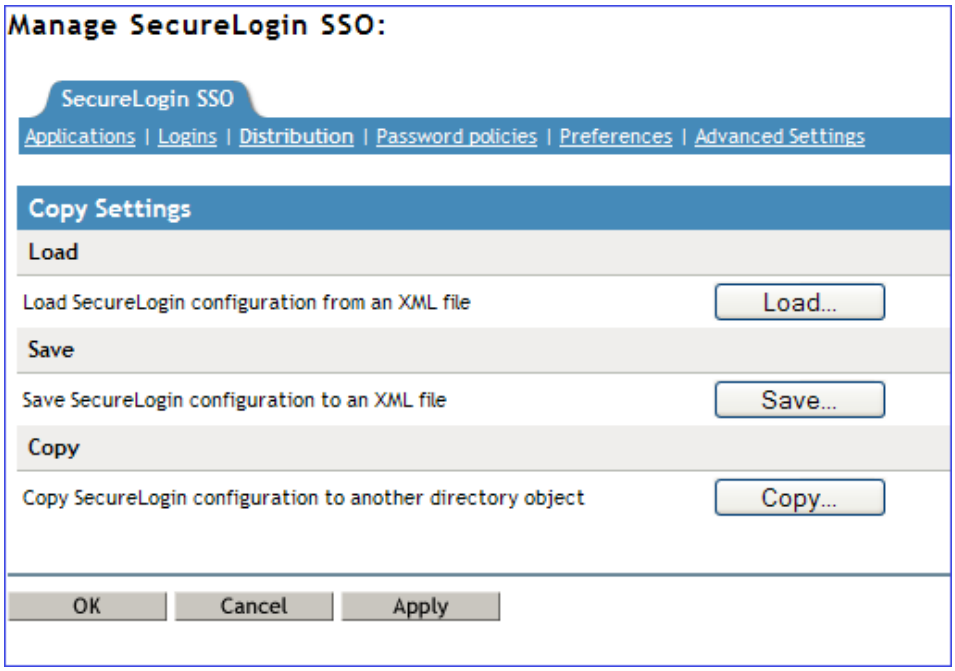
Importing Individual Applications

1 Log in to iManager.

2 Select **Securelogin SSO > Manage Securelogin SSO**. The Manage SecureLogin SSO page is displayed.

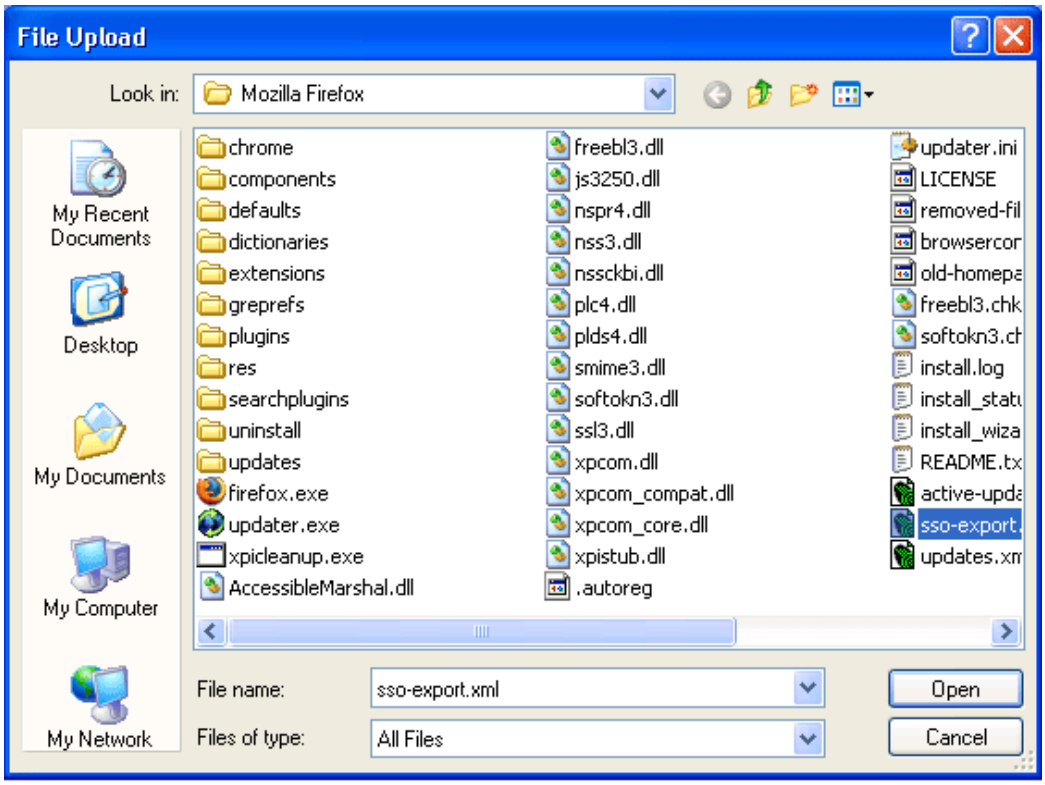
3 In the object field, specify your object name, then click **OK**.

4 Click **Distribution**. The Distribution details are displayed.



5 Click **Load**. The Select SecureLogin Configuration dialog box is displayed.

6 Browse to and select the exported XML file.



7 Click **Open** to select the file.

The selected predefined applications and application definitions are copied across to the receiving organizational unit or container.

The selected Securelogin configuration is copied across to the receiving object.

If predefined applications and application definitions currently exist in the receiving object, a confirmation message is displayed to confirm or reject overwrite with the imported data.

8 Click **Import** to confirm or click **Cancel** to reject overwriting with the imported data.

A SecureLogin message is displayed to confirm SecureLogin data is loaded.

Modifying Predefined Applications and Application Definitions

SecureLogin predefined applications and application definitions are easily modified to cater to your organization's requirements.

Use the following procedure to modify a SecureLogin predefined application or application definition:

1 Double-click the SecureLogin icon in the notification area to display the Personal Management utility.

2 Click **Applications**. The Applications pane is displayed.

3 Double-click the required application definition. The application details are displayed.

4 Select the **Definition** tab. The application definition editor is displayed.

5 Modify the application definition or the predefined application, as required.

It is a good practice to include the date and a description of the changes made for future reference.

The predefined Web applications such as eBay or Hotmail under the **Type** drop-down list are titled **Web** and not **Advanced Web**. There is no difference between a **Web** application definition or an **Advanced Web** application definition.

6 Click **OK** to save changes and close the Personal Management utility.

For information on how to modify specific functions see Chapter 6, “Command Reference,” on page 53.

Building an Application Definition in the Personal Management Utility

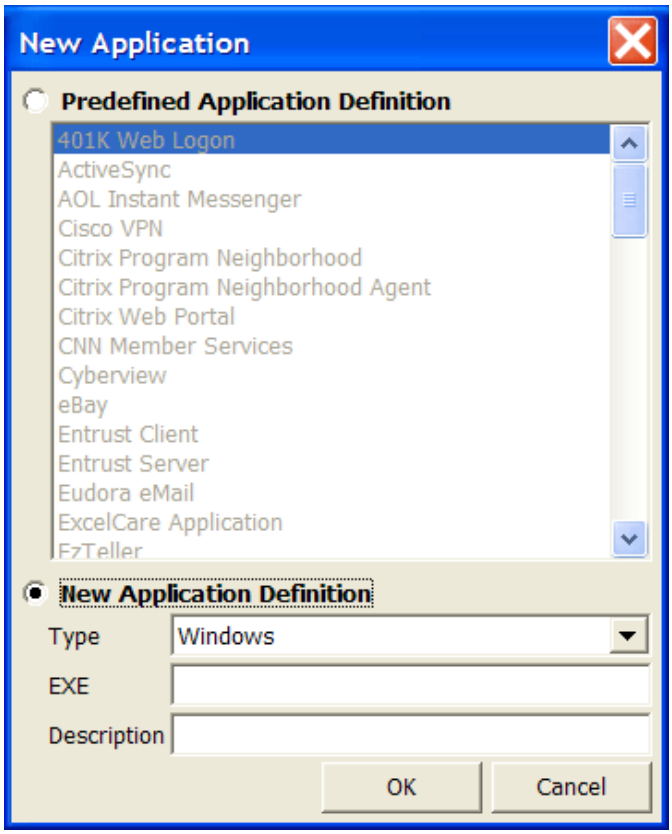
This section describes how to create and modify SecureLogin application definitions in the Personal Management utility. It is recommended that you test the application definitions locally and then copy them to the relevant container or organizational unit in multi-user directory environments.

Use the following procedure to create an application definition for a Windows application:

1 Double-click the SecureLogin icon in the notification area to display the Personal Management utility.

2 Select **File** > **New** > **Application**. The New Application dialog box is displayed.

3 Click **New Application Definition**, and select the required application type from the **Type** drop-down list.



4 Specify other details such as the EXE or the description.

These fields vary based on the application definition type that you have selected. For example, if you select **Windows** as the **Type**, you must fill in the **EXE** and **Description** fields.

5 Click **OK**. The application definition is added to the left pane under applications and the details display in the right pane.

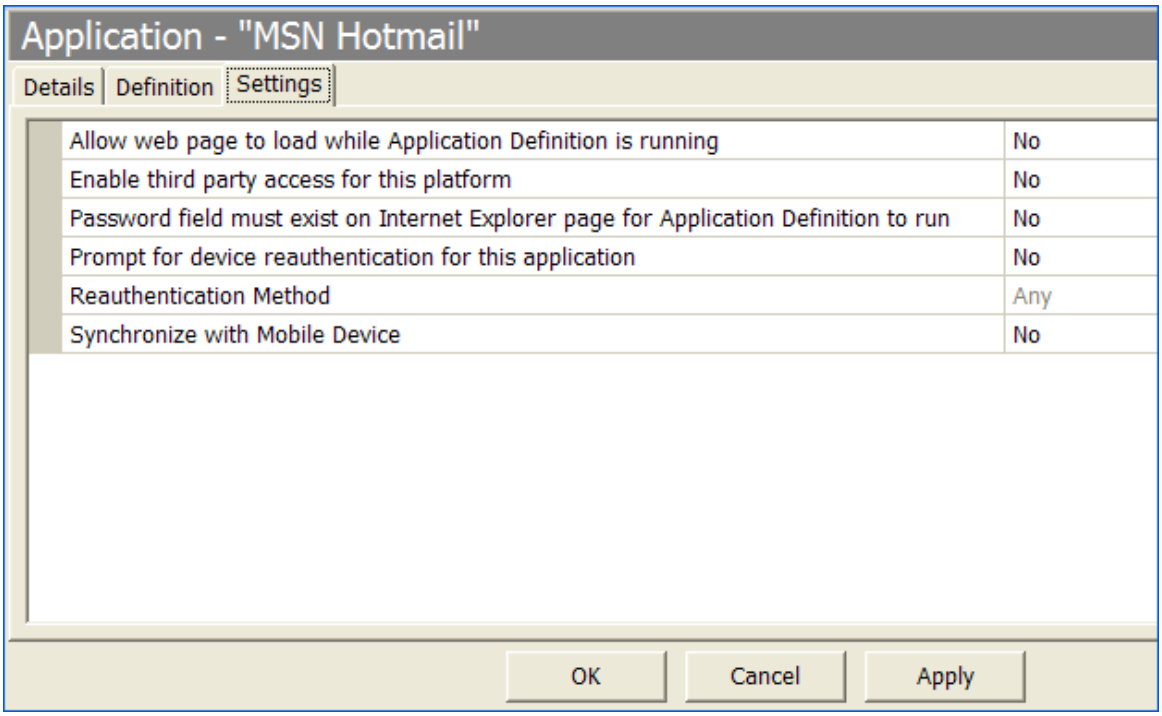
6 Select **Definition**, and delete the text, `# place your application definition here`.

7 Specify your application details, then click **OK** to save the changes and close the Personal Management utility.

NOTE: If you are creating multiple application definitions, click **Apply** to save changes without closing the Personal Management utility.

Settings Tab

Figure 3-1 *The Settings Options*



The **Settings** tab includes the following options for application definitions and predefined applications:

Table 3-1 *Settings Options*

Option	Description
Allow web page to load while application definition is running	Applies to Microsoft Internet Explorer and application definitions created for Web pages and JavaScript login that execute in a Web page. By default, this option is set to **No.** This suspends completion of any other Internet Explorer tasks until the log in is completed. If this option is set to **Yes**, SecureLogin allows Internet Explorer to continue functioning while SecureLogin is executing the login.
Enable third party access for this platform	By default, this option is set to **No.** This disables the API access for this predefined application or the application definition. If this option is set to **Yes**, it disables the API access for this predefined application or application definition.

Option	Description
Password field must exist on Internet Explorer page for application definition to run	Applies to Microsoft Internet Explorer and application definitions created for Web pages and JavaScripts within Web pages. If this option is set to **Yes**, SecureLogin does not execute automated login for pages without a password field. If this option is set to **No**, your Web application returns errors on pages without password fields that you need to handle with SecureLogin. For example, password change successful.
Prompt for device reauthentication for this application	Allows you to reauthenticate an application against an Advanced Authentication (AA) device. By default, this option is set to **No**, which means that users are not prompted for device reauthentication for the application. If this option is set to **Yes,** user are prompted for device reauthentication for the application.
Reauthentication Method	This option allows you to reauthenticate to an application before single sign-on. This option is available only when **Prompt for device reauthentication for this application** is set to **No**. The reauthentication methods available are: ◆ Any ◆ Biometric ◆ Smart card ◆ Token ◆ Password ◆ Passphrase ◆ Directory password
Synchronize with Mobile Device	This option is set to **No** by default, enabling synchronization to an API-enabled hand-held device, for this predefined application or application definition. If this option is set to **Yes**, it disables synchronization to an API-enabled handheld device for this predefined application or application definition.

Windows Application Definition Tools

SecureLogin provides wizards to assist with the creation of basic application definitions. For more complex applications and requirements, SecureLogin provides the following tools to assist with finding the application information required to build an application definition:

- “Finding Application Details with Window Finder” on page 33
- “Finding Application Details with the Login Watcher” on page 35

Finding Application Details with Window Finder

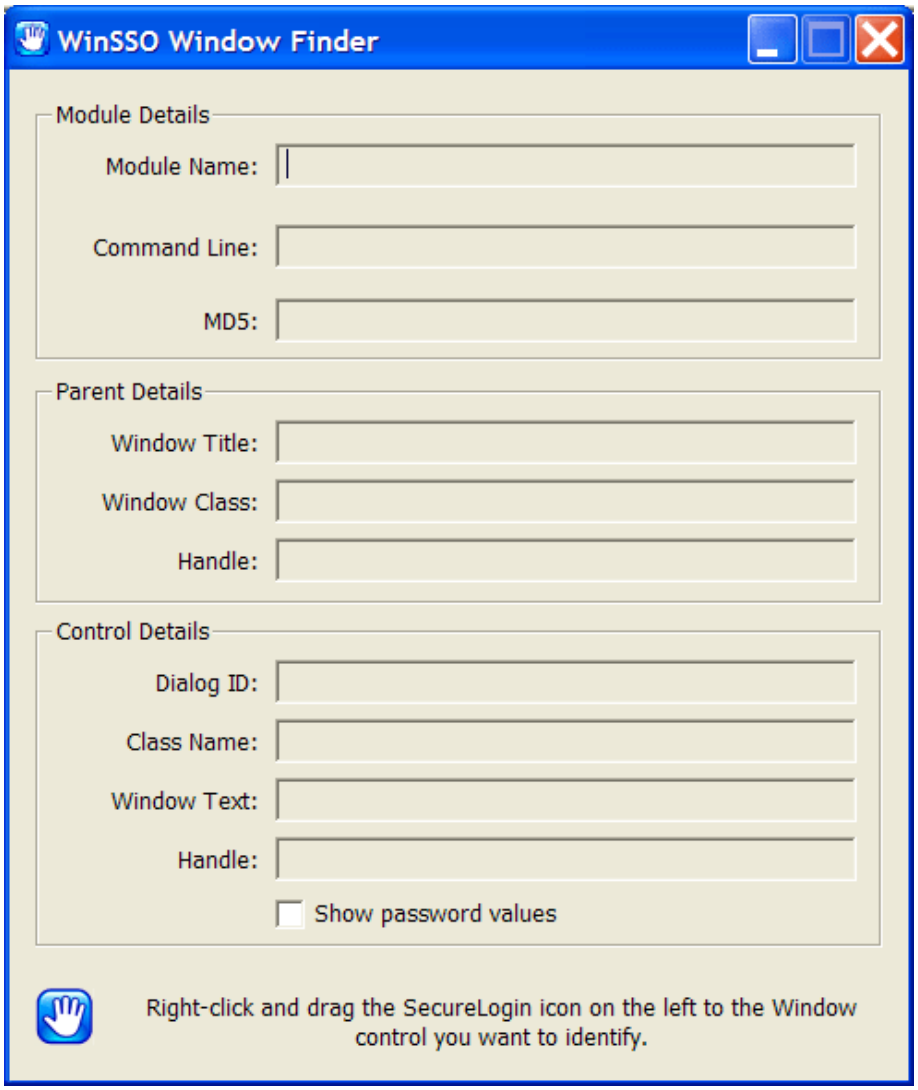
The SecureLogin Window Finder finds windows applications details, including control and dialog box IDs. SecureLogin might require this information to identify specific objects in order to uniquely identify the application.

Control IDs are used to uniquely identify objects within a window. Window Finder extracts this information from the application for use in the application definition.

Starting the Windows Finder

The following procedure uses the SecureLogin test application provided on the SecureLogin product installer package or your other distribution source.

1 On the Windows **Start** menu, select **All Programs > NetIQ Securelogin > Window Finder**. The Window Finder is displayed.



2 Right-click the SecureLogin icon in the dialog box, drag it to the required window, field or control, and release the mouse button.

WINSSO Window Finder Details

The following table lists the fields in the WinSSO Window Finder:

Table 3-2 *Window Finder Details*

Field	Description
Module Details Section	
Module Name	This is the Windows executable name for the selected application. This is the application name for a Windows application definition or the predefined application.

Field	Description
Command Line	This is the full command line used to start the application. You can use this information in along with the `GetCommandLine` command.
Parent Details Section	
Window Title	This is the title of the window of the selected control. Use with the `Title` command in the `Dialog/EndDialog` section of the application definition.
Window Class	This is the Windows class name for this dialog or window. Use with the `Class` command in a `Dialog or EndDialog` section.
Handle	This is the internal Windows handle for this window. This is generally not used in application definitions.
Control Details Section	
Dialog ID	This is the unique number identifying the control. Use it with various commands, including `Type, SetPlat,` and `Click.`
Class Name	This is the Windows class name for the control. SecureLogin supported classes, which include Edit, Combo box, and Static.
Window Text	This is the test that exists on the control. Useful to copy and paste into the application definition editor. 1. Note or copy the required details from the WinSSO Window Finder window from the relevant fields. 2. Click **Close** to quit and close the WinSSO Window Finder window.

Finding Application Details with the Login Watcher

The Login Watcher records login and Windows application data to provide information that you might need for creating an application definition.

- "Order Information Is Recorded and Stored" on page 35
- "Information Details" on page 36
- "SecureLogin Test Application Example" on page 36

Order Information Is Recorded and Stored

Information is recorded and stored in a text file in the following order:

Time||Module Name||Window Handle||Window Text||Class Name||Parent||Visible Flag||Title Flag||Control ID

NOTE: The Login Watcher records all log in information, including usernames and passwords, in a text file. This text file might be a security issue.

Information Details

Information Item	Description
Time	Milliseconds elapsed since the Login Watcher started.
Module name	Name of the executable being recorded.
Window handle	Unique identifier for the window.
Window text	All text displayed in the window, which includes text entered during login and text displayed as labels for fields and buttons.
Class name	Name of the window class.
Parent	Window handle of the parent window.
Visible flag	Refers to top-level windows that have the style set to **Visible**.
	If set to **Visible**, the word Visible displays; otherwise the field is empty.
Title flag	Refers to top-level windows that have the style set to display the Window Title.
	If the title is not displayed, then the field is empty.
Control ID	The unique numerical identifier for the windows object.

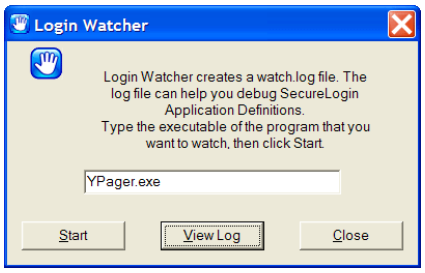
SecureLogin Test Application Example

The following procedure uses the SecureLogin test application:

1. Right-click the SecureLogin icon on the notification area.
2. Select **close** from the menu.
3. Right-click the Windows **Start** menu > **Explore**.
4. Double-click `loginwatch.exe`, by default located at `<...>\program files\novell\securelogin\tools`. The Login Watcher dialog box is displayed.

5. Specify the executable filename in the Login Watcher field. For example, YPager.exe.

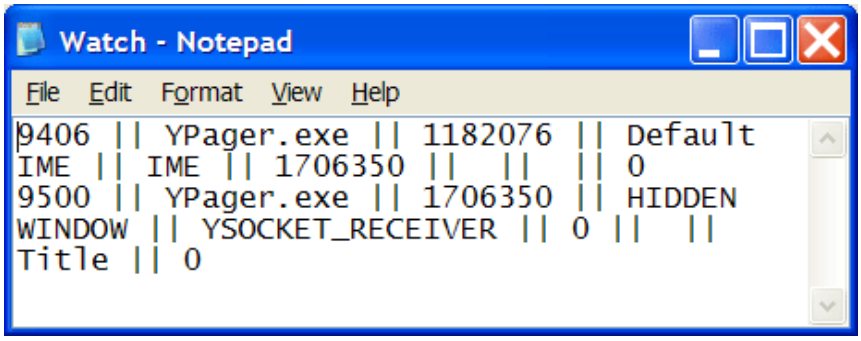
6 Click **Start**. The Now Recording Log confirmation dialog box appears.



7 Log in to the relevant application.

8 Click **Stop** when logged on successfully to return to the Login Watcher dialog box.

9 Click **View Log**. SecureLogin starts the `Notepad` application and displays the `watch.txt` file with login details recorded.



10 Note the required information or save the text file with a different name.

11 Click the Login Watcher dialog box. Click **Close**.

Application Definition Elements

Application definitions use various symbols to define the function of each line. The following table lists the definitions for these symbols.

Table 3-3 *Symbol Definitions*

Symbol	Description
#	Use this symbol to define a line of text as a comment. Comment fields are used to leave notes. Any line that starts with a # is ignored. Use comment lines for the following: ◆ Defining sections of an application definition, for example the login window and Change Password window. ◆ Explaining complex sections. ◆ Removing command lines during creation and editing of the application definition. This saves continuously deleting and rewriting lines while testing. ◆ Making notes such as when the application definition was written, what version of the software it was written for, and so on. When used as part of a command, such as `Class` or `Type`, the symbol precedes a numerical value. You can use these numerical values to specify a target for the command. For example: `Type $Username #1 or Class #32770`
" "	Use quotation marks to group together text or variables that contain spaces. Quotation marks are used with commands such as `Type`, `MessageBox`, and `If -Text`. For these command lines to work, you must use quotation marks in the following method to group the text together: ◆ `Type "Database 2"` ◆ `MessageBox "Please confirm your log in details."` ◆ `If -Text "Login failure"`
$	Use the dollar sign to define a variable to be stored by SecureLogin as part of application credential set. The stored value will be retrieved and used by SecureLogin for any future instances of the application. These variables are used to store information such as usernames and passwords.
?	Use the question mark to define the use of a runtime variable. The values of these variables are not stored in the directory; they are reset each time SecureLogin is started. Alternatively, with the use of the `Local` command, these variables are reset each time the application definition is started. These variables are used for temporary information, such as counting, data processing, and date information. The question mark is also used to identify some system runtime variables. For example, `?SysUser` and `?SysPassword`.

Symbol	Description

%	Use the percentage sign to have SecureLogin retrieve the value of a directory attribute of the user object. The attributes available vary depending on the directory in use, and the setup of the directory.
	
	Examples of the attributes you can use are FCN and Surname. Type %FCN or Type %Surname.
	
	NOTE: The attribute name defined here needs to be in the exact case and syntax as the attribute name in the directory. Also, Quotes are required around the variable if the attribute name contains a space. For example,
	
	`Set ?text "%Login Time"`
	
	or
	
	`Messagebox "%Given Name"`
	
	For more information, see “Directory Attribute Variables” on page 41.

****	Use the backslash with the `Type` and `SendKey` commands to specify the use of a special function. The backslash is used along with values to perform the simulation of the pressed keys on the keyboard. Examples of frequently used functions are provided in the following list: ◆ **\Alt-F:** Alt+F on the keyboard in Windows and Web applications. ◆ **\D:** Delete key in a Windows and Web applications. Not applicable to terminal emulators. ◆ **\N:** Enter key in a Windows and Web applications. Not applicable to terminal emulators. ◆ **\T:** Tab in Windows and Web applications. ◆ **\-T:** Shift+Tab in Windows and Web applications.
@	Use the same way as the backslash symbol, except its use is limited to HLLAPI-enabled emulators. This symbol is used along with values to perform the simulation of pressed keys on the keyboardwhen communicating with a host in a terminal emulator application. For example, use @E to simulate pressing the Enter key in a terminal emulator application.
-	Use the hyphen as a switch within several commands, such as `If` and `Type`. The hyphen is used along with values to modify the behavior of commands (such as -Raw), or to switch on or off certain functions (such as -YesNo).

4 Application definition variables

Types of Variables

SecureLogin supports the use of four different types of variables:

- Stored
- Runtime
- Directory attribute

Using a variable to change the default platform

NOTE: Specify variables without spaces, for example $Username_Alias. If you use spaces you must enclose the entire variable in quotation marks, for example "$Username Alias".

Each variable defaults to the platform specified in the application definition or the predefined application name. You can use a variable to change the platform.

Example:

If you have applications `A` and `Z`.

1. Application `A` has default credential `A`, and linked credentials `B` and `C`. A credential selection will prompt you to choose A, B or C.
2. Application Z has default credential Z with linked credentials W, X and Y If you have set the platform to Z and then a credential selection will prompt you to choose W, X, Y or Z.

`$password`: This variable will prompt the user for a credential. For application A a credential selection will prompt you to choose A, B or C and for application Z a credential selection will prompt you to choose W, X, Y or Z.

`$password(A)`: This variable will not give any choice and will use the credential from A.

`$password(Z)`: This variable will not give any choice and will use the credential from Z.

Directory Attribute Variables

SecureLogin can read directory attributes from the currently logged on user's object. For example, `%CN` reads the CN attribute from the currently logged in user's object and displays it.

IMPORTANT: SecureLogin can read and display an attribute only if the attribute is defined in the user object. If the attribute variable is not defined, SecureLogin will display an empty attribute.

You can use the percentage symbol (%) variables only when SecureLogin is configured to use a directory and only on single-valued text attributes.

Quotes are required around the variable if the attribute name contains a space. For example:

```
Set ?text "%fullname"
MessageBox "%mail"
```

For more information on application definition elements and symbol usage refer to Chapter 3, “Managing Application Definitions,” on page 25.

Stored Variables

Stored variables are the most common style of variable used in application definitions and Predefined Applications. They are preceded with a dollar symbol ($). Use these variables to store the values used during the login process, such as usernames, passwords and any other details that are required.

This section contains the following information:

Storing the Variables

The values of these variables are stored in the directory under the user object. They are encrypted so that only the user can access them. You can store variables separately for each application definition and predefined application, so the username variable for one application can be different from the username variable for another application. It is, however, possible to set an application to read variables from another application's application definition and predefined application. This is useful for applications that share user accounts or passwords.

For details on how to do this, see “SetPlat” on page 155.

Using Stored Variables

If a stored variable is referenced in an application definition and predefined application, and there is no value stored for that variable (for example, the first time the program is run), SecureLogin prompts the user to enter a value for the variable. This is an automatic process. It is also possible to manually trigger this process to prompt a user to enter new values for particular variables.

For details on how to do this, see “DisplayVariables” on page 82 and “ChangePassword” on page 68.

NOTE: If you want to hide a variable from an administrator by displaying it as asterix (****) instead of clear text, begin the variable name with $Password. For example, the $PasswordPIN variable is protected as described, however, $PIN is not.

Example of stored variables in use:

```
Dialog
Class #32770
Title "Log on"
EndDialog
Type $Username #1001
Type $Password #1002
Click #1
```

Runtime Variables

Runtime variables are generally used for storage of calculations, processing data, and date information. You can also use them for temporary passwords and usernames.

Runtime variables are preceded by the question mark symbol (?). They have two modes:

- Normal runtime variables are reset each time SecureLogin is started.
- Local runtime variables are reset each time the application definition and predefined application is started.

Runtime variables are Normal by default. For details on how to switch a runtime variable to Local mode, see “Local” on page 106.

Using Runtime Variables

Runtime variables are not stored in the directory or the SecureLogin cache; they are used straight from the computer's memory. For this reason, it is important not to use runtime variables for the storage of usernames, passwords, or other details SecureLogin will need to access in the future.

If runtime variables are used for such details, the user is prompted to enter them each time the application definition or predefined application is run, or each time SecureLogin is restarted. Users are not prompted for `?variables` that have no value. These variables are given the value `<NOTSET>`.

Example of a Runtime Variable

```
Dialog
Class #32770
Title "ERROR"
EndDialog
Local?ErrorCount
Increment?ErrorCount
If?ErrorCount Eq "2"
MessageBox "This is the second time you have received this error. Would you like to
reset the application?" -YesNo ?Result
If ?Result Eq "Yes"
       "App.exe"
      Run "C:\App\App.exe"
Else
      Set?ErrorCount "0"
   EndIf
EndIf
```

SecureLogin Supported Variables

SecureLogin reads details from the system and uses them to create variables that you can incorporate into the application definition. These variables are automatically generated as runtime variables and used in the same manner within any application definition.

Variable	Description
`?BrowserType(system)`	Contains Internet Explorer and indicates the browser on which the application definition is running. This variable is only set in a Web application definition.

Variable	Description
`?CurrTime(system)`	Contains the running time in seconds from Jan 1970 to the present. Use this variable to force password changes every X days, or similar. Do not use the application definition to force a password change if you want to continue the application generating the change password event. This is recommended. Use this variable on applications where you cannot set a password expiry at the application back end.
`?DSVariable(system)`	SecureLogin traps the `DataStoreVariableNotAvailable` exception and stores the name of the variable, which resulted the exception, in a built-in variable called `?DSVariable`. This helps in tracing errors that occurs while trying to read a directory attribute.
`?SysContext(system)`	Contains the context within which the current SecureLogin user's directory object exists.
`?SysPassword(system)`	Contains the directory password of the user currently using SecureLogin. This variable is only available if the appropriate options are chosen when installing SecureLogin.
`?SysServer(system)`	Contains the name of the server or the IP address of the server that was entered in the Novell client login panel. **NOTE:** This variable is only available if the Novell client login extension is installed (eDirectory) and is not available if the MS Active Directory or ADAM option has been installed.
`?SysTree(system)`	The name of the directory tree that the SecureLogin is currently using. **NOTE:** The variable ?SysTree will return the Domain name when using Microsoft GINA (Microsoft Active Directory or ADAM)/ Credential Provider and the Tree name or Port Number when using Novell GINA or LDAP installation.
`?SysTSLaunched (system)`	Contains the condition state value when SLLauncher is run. This variable is set to "True" when a script is being executed by SLLauncher. Otherwise it will be "<NOTSET>".
`?SysUser(system)`	The name of the user currently using SecureLogin.
`?sysInstallDir(Tray)`	Location of the tray or any other SecureLogin application.
`?sysTrayHWND`	System tray window handle.
`?sysProductVersion(`*`os`*`)`	Version of the operating system.
`?sysProductVersion(`*`app`*`)`	Version of the application on which the script is running.
`?sysProductVersion(`*`worker`*`)`	Version of the process (worker) running for the application.
`?sysProductVersion(`*`someapp`*`.exe)`	Version of some other application in the NetIQ SecureLogin folder.
`?sysFileVersion(`*`app`*`)`	Version including the patch number or the hotfix number of the application on which the script is running.

Variable	Description
?sysFileVersion(*worker*)	Version including the patch number or the hotfix number of the process (worker) running on the selected application.
?sysPlatform(*os*)	Architecture of the operating system.
?sysPlatform(*worker*)	Architecture of the process (worker) that runs for the selected application.

Application Definition Best Practices

The following are some of the best practice rules to follow when creating an application definition. These rules make reading the application definition easier and also help if you need to make modifications in the future.

Symbols Used

Table 4-1 *Description of Symbols*

Symbol	Description
< >	Angle brackets represent an item. For example, text, variable, or value.
[ ]	Square brackets represent an optional item. If an item is not marked with square brackets, it is a compulsory item.
↵	Indicates a line break

Blank Line Between Sections

NOTE: Always place the title after all other commands in the dialog block.

Leave a blank line between sections, for example, between the dialog block and the rest of the application definition.

Instead of	Use

`# Logon Dialog Box`
`Dialog`
`Class #32770`
`Title "Log on"`
`EndDialog`
`Type $Username #1001`
`Type $Password #1002 Click #1`
	`# Logon Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Log on"`
	`EndDialog`
	
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

Capitalization

Use capitalization where applicable.

Table 4-2 *Capitalization*

Instead of...	Use...
`messagebox "some text" -yesno ?result`	`MessageBox "Some text" -YesNo ?Result.`

Comments

Use comments throughout to explain what each section does and how it does it.

Instead of...	Use...

`Dialog`
`   Class #32770`
`   Title`
`"Log on" EndDialog`
	`# Written by B. Smith 2004, modified C. Silvagni 2006`
	`# Logon Dialog Box`
	`Dialog`
	`   Class #32770`
	`   Title "Log on"`
	`EndDialog`

Indent Section

Indent sections between pairs of commands, for example `Dialog`, `Repeat`, and `If`. Use an indent of three spaces.

Instead of...	Use...

`If -Text "Some text"`
`#Do thisElse`
`#Do This`
`EndIf`
	`If -Text "Some text"`
	`   #Do thisElse`
	`   #Do this`
	`EndIf`

Password Policy Names

Password policy names must represent the program they are used for. Do not use numerical names.

Instead of...	Use...
PasswordPolicy3	GroupwisePasswordPolicy

Quotation Marks

Always use quotation marks around segments of text in commands.

Instead of...	Use...
Type TextOrIf -Text Login	Type "Text"OrIf -Text "Log on"

Regular Expressions

Regular expressions are text patterns normally used for string matching. Regular expressions might contain a mix of plain text and special characters to indicate the kind of matching to be done.

For example, if you are searching for any numeric character, then the regular expression that you use for the search is, “[0-9]”.

The square [ ] brackets indicate that the character that is compared must match any one of the characters enclosed with in the brackets. The dash (-) between the zero (0) and nine (9) indicates that the range is between the number zero and nine.

If you need search for a special character, then you must use the backslash (\) before the special character.

If your regular expression does not match any controls on a particular application screen, SecureLogin will prompt you to check your regular expression and ensure the correct control is selected. Special characters in your regular expression might need to be escaped.

The following table briefly describes the characters that can be used in regular expressions within SecureLogin application definitions, in particular the `RegSplit` command detailed in “RegSplit” on page 136.

Character	Description
\ (Backslash)	The \ is an escape character indicating that the next character must be used as a regular search character and not as a special character. For example, the regular expression “\” matches a single asterisk and the expression “\\” matches a single backslash.
^ (Caret)	The ^ is an anchor. If you use the ^ preceding any character, it searches the beginning character of any string. For example, the expression “A^” matches an “A” only at the beginning of the string.
[^ (Square bracket and Caret)	The ^ immediately following [, is used to exclude the characters within the square brackets from matching the target string. For example, the expression “[^0-9]” specifies that the target character must not be a numeral.
$ (Dollar sign)	The $ is an anchor. The $ matches the end of the string. For example, the expression “abc$” matches the substring “abc” only if it is at the end of the string.
| (Vertical bar or pipe)	The | allows the character on either side of the vertical bar (or pipe) to match the target string. For example, the expression “a|b” matches a as well as b.
. (Period or full stop)	The . matches any character.
* (Asterisk)	The * indicates that the character to the left of the asterisk in the expression must match at least zero or more times.
+ (Plus sign)	The + indicates that the character to the left of the plus symbol in the expression must match at least once.

Character	Description
? (Question mark)	The ? indicates that the character to the left of the question mark must match at least zero or more than once.
( ) (Parentheses)	The ( ) enclosing a set of characters affects the order of pattern evaluation and also serves as a tagged expression that can be used when replacing the matched substring with another expression.
[ ] (Square brackets)	The [ ] enclosing a set of characters indicates that any of the enclosed characters might match the target character.

Capture Groups

If you are using the regular expressions to extract information rather than just match the text, use capture groups. You can use a captur egroup when using regular expressions to select credentials to be used based on a particular option from a comman dialog box. For example, the name or IP address of a particular server to which you want to connect. In such a scenario, SecureLogin uses the capture group to make a unique name for a credential set and allows useres to have different credentials for different servers.

For example, if a message indicating `Welcome Kerry to the Corporate server` is displayed, then Kerry is the name of the user and Corporate is the name of the server. If you want to match just the text, *Welcome .+ to the .+ server*. If you want to use the server name as the name of the credential set, so that you can create other credential sets for other servers, add a capture group to the same regular expression and get *Welcome .+ to the (.+) server*.

For more general information on regular expressions and usage refer the Boost Web site. (http://www.boost.org/doc/libs/1_33_1/libs/regex/doc/syntax_perl.html)

NetIQ uses the Boost regular expression library (in Perl) when developing SecureLogin. While other reference sites provide detailed and comprehensive information on regular expressions, only the expressions listed in the tables are supported by NetIQ.

Switches

Switches are placed directly after the command, for example, `Type -Raw, If -Text`.

Table 4-3 *Switches*

Instead of...	Use...
`Type $Username -Raw`	`Type -Raw $Username`

Variables

All variable names start with a capital letter.

Table 4-4 *Variables*

Instead of...	Use...
`Type $username`	`Type $Username`

Writing Subroutine Sections

Write subroutine sections at the bottom of the application definition and not partway through.

The name of the subroutine should describe its function. Do not use a numeric name. The name should follow the capitalization rule.

Wherever possible, use the `Include` command to create generic application definitions for frequently used elements, for example password change procedures. For common processes within the same application definition, use subroutines.

5 Support for Dynamic Controls

SecureLogin provides a flexible single sign-on solution by allowing user to choose what to do when an application generates a message. The application definitions of SecureLogin provides full control on the single sign-on environment. You can use application definition to interact with controls present in a window and specify desired input. Controls are the UI elements that require user interaction, these include text fields, radio button, check boxes etc. SecureLogin needs to identify the controls correctly before performing single sign-on for any window. SecureLogin identifies the controls using the control matching. To match the controls, in JavaSSO and DotNetSSO, SecureLogin navigates through the window using Z-order and assigns control IDs in the increasing order that allows writing/ generating of the single sign-on scripts. In WinSSO, Z-order or the existing control IDs are used to match the controls. When the window is opened next time, the application definition matches the controls using the same Z-order or control IDs and verifies the controls. When a control is successfully matched, SecureLogin provides single sign-on using scripts.

Single sign-on might fail when a certain control does not appear at the same place in the window because it breaks the order that SecureLogin identified initially. It can be caused by hidden elements in the window or addition of a new control. Single sign-on fails the control matching and prevents script execution. The `MatchElement` and the `MatchOption` commands of SecureLogin are used to address this issue. These commands provide the following capabilities:

- Using single Dialog/EndDialog matching script to provide single sign-on to windows that changes order of controls. It uses the class, name, value or type attributes to identify the dynamic controls.
- Provides better control match when using WinSSO, JavaSSO and DotnetSSO workers.
- `MatchElement` and `MatchOption` are the dialog specifier commands that are used with other commands to interact with new attributes and achieve desired single sign-on scripts. For example, `ReadInput` command can be used to define an action in the script after reading an input from user.

Make a note of the following points when you use the `MatchElement` command:

1. The command selectors must be specified at the beginning of the command. For example, `MatchElement #<id>`, here `#` is the selector. The `MatchElement` command supports following selectors:

Selector	Example	Example Description
`#ID`	`MatchElement #login #10`	Matching element using ID.
`.Class`	`MatchElement #login .10`	Matching element using class.
`:nth-of-type()`	`MatchElement #login :nth-of-type(10)`	Matching the order.
`Value`	`MatchElement #login value=10`	Matching the element value.
Visible	`MatchElement #login [value='Login'] [visible=true]`	Matching the element visibility.
(space)	`MatchElement #ID <parent> <child>`	Matching the child element of a parent element. You can apply any match on both parent and child elements.
`-optional`	`MatchElement #combo ComboBox -optional`	Allows optional matching of elements. It allows script execution even when the control does not match.

2. The class matching, order matching and ID matching can be specified in any order. For example, the following two commands are in different order but provides same result.

   ```
   MatchElement #login EditBox#1001:nth-of-type(2).TEdit
   MatchElement #login EditBox.TEdit#1001:nth-of-type(2)
   ```

3. You must specify type matching while using `:nth-of-type`. For example:

   ```
   MatchElement #username EditBox:nth-of-type(2)
   ```

4. You must specify exact matching criteria. For example, in the command mentioned below, it will match the first button available.

   ```
   MatchElement #button Button
   ```

5. You must use `Value` and `Visible` at the end of the command.
6. The class names can contain special characters. For example, #, . (dot) or (space) etc. In such scenarios, use \ to escape these special character, see the following example.

   ```
   MatchElement #password
   EditBox#textPassword.WindowsForms10\.EDIT\.app\.0\.259f9d2
   ```

7. Use `MatchElement` with the following commands to achieve a more flexible single sign-on solution.
   a. `SelectOption`
   b. `ReadInput`
   c. `MatchOption`
   d. `Click`
   e. `TextInput`

For more information on `MatchElement`, see MatchElement in NetIQ SecureLogin Application Definition Guide.

6 Command Reference

This section contains the following information:

Command Reference Conventions

This section consists of descriptions and examples of the commands that make up SecureLogin application definitions.

An index and summary is also included as Chapter 2, “Command Quick Reference,” on page 13.

Command Information

The information for each of the commands includes:

Use With values

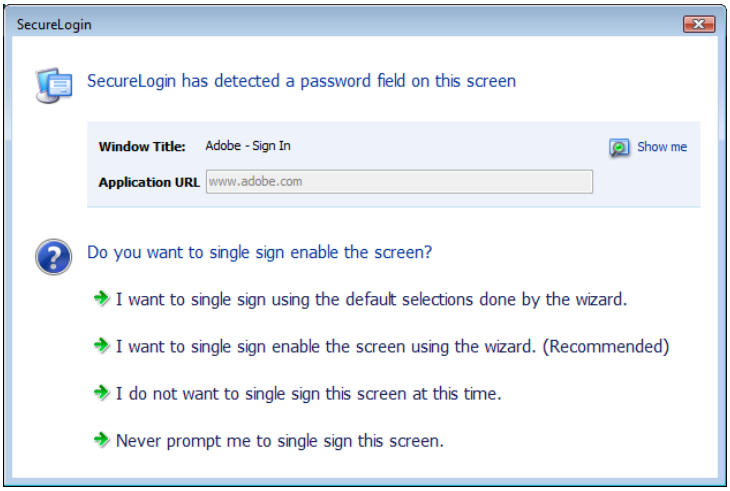
Command	Description
Java	Use as part of a Java* application definition.
Startup	Use as part of a startup.
Terminal Launcher	Use as part of a terminal launcher application definition.
Advanced Web	Use as part of a manually created Web site or Internet application definition. Not compatible with the Web Wizard application definition language. **NOTE:** A predefined **Web** application and an **Advanced Web** application definition are the same.
Web Wizard	Use as part of application definitions created automatically by the Web Wizard. Web Wizard application definitions can be kept in their original XML format or converted to an ASCII script for advanced editing.
Windows	Use as part of a Windows application definition.

Type Values

Command	Description
Action	Performs an action, for example, the Type command types information into a field.
Dialog specifiers	Defines dialog boxes, for example, the Parent and Class commands.
Flow control commands	Directs SecureLogin to a specific location in the application definition, for example, Repeat and EndScript commands.
Variable manipulators	Modifies variables, such as the Add and Subtract commands.

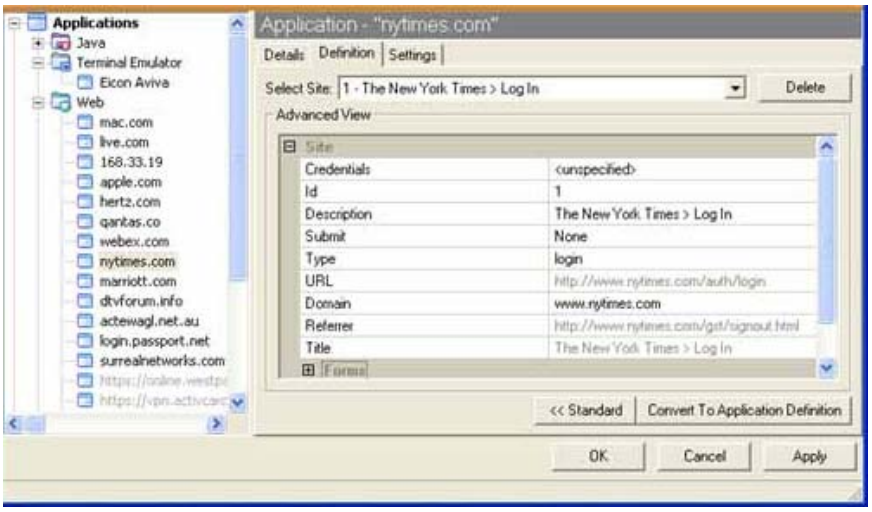
Web Wizard Application Definition Conventions

The SecureLogin advanced Web Wizard makes it easier for users to enable single sign-on Web sites and capture a user's Web-based login details. When the user accesses a Web page from the browser, SecureLogin automatically launches the Web Wizard.



The Web Wizard captures the user's login details and adds them to the user's Web application definitions.

When managing user's Web log in credentials, the **Definition** tab of the **Advanced Setting** page allows administrators to customize site and user credential details. Also available under the **Definitions** tab is an Advanced function that provides more functionality with their associated values and the option to convert the user's login credentials to an application definition.



For more details on how to manage application definitions, see Chapter 3, “Managing Application Definitions,” on page 25.

Site Matching

In SecureLogin version 6.0 and later, Web commands are added to allow much finer control of site matching. Detailed information of the loaded Web site can be matched and used to execute blocks of scripting commands.

The technique used to specify constraints upon a site match are similar to those constraints used in windows scripting.

Instead of `Dialog/EndDialog` commands, equivalent `Site/EndSite` commands have been created and can now be used.

Within these Site blocks, Match commands can be used to filter a given site. If one of the specified match commands fails to match, then the site block will fail to match as a whole. For details of the `Site/EndSite` block command, see “Site/Endsite” on page 160.

Form/Field/Option matching

When matching a specific form, field or other match option it is often the case that multiple items will match the selection criteria. In these cases, the first item on the Web site which matches is considered to be the match.

To access the other fields which also need to be matched, subsequent match commands may be added with the same selection criteria.

NOTE: Matched items may only be matched once, and

Each ID must be unique and cannot have been used previously.

For example:

```
MatchField #1:1 -type "password"
MatchField #1:2 -type "password"
```

will match a site with two password fields. The first is given the ID '#1:1' the second is given the ID '#1:2'

Form/Field/Option ID’s

When matching a site, match methods are used to give specific fields, forms and options their own unique ID.

Once the site has been successfully matched, the given ID is used in input commands to specify particular items.

The actual ID's are denoted with a # followed by 1, 2 or 3 numbers, each separated by a colon – for instance, "#1:3:2".

Auditing

For auditing, use either the `AuditEvent` command built into SecureLogin or the `LogEvent` command from the Windows Resource Kit. Refer the *SecureLogin Administration Guide*.

For details, see “AuditEvent” on page 64

One-Time Passwords

The use of multiple passwords places a high maintenance overhead on large enterprises. Users are routinely required to use and manage multiple passwords, which can result in a significant cost, particularly with regard to calls to the help desk to reset forgotten passwords, or to ensure that all passwords are provisioned when a new user starts or are deleted when an existing user leaves the organization.

One of the main benefits of implementing one-time password systems is that it is impossible for a password to be captured on the wire and replayed to the server. This is particularly important if a system does not encrypt the password went it is sent to the server, as is the case with many legacy mainframe systems.

One-time passwords also offer advantages in terms of disaster recovery because the encryption key is used to generate the one-time password rarely changes. System restoration, which might be to a system version that is hours or many months old, can be achieved without consideration of restoring users' passwords or notifying staff of new passwords.

SecureLogin provides a secure, robust and scalable infrastructure by integrating ActivCard* one-time password authentication functionality.

For details of the `GenerateOTP` command, see “GenerateOTP” on page 87

Commands

AAVerify

This section introduces the commands that you can use in a script, and the usage of the command.

Use with	Startup, Terminal Launcher, Web, or Windows
SecureLogin version	3.5 or later (arguments added in version 3.0)
Type	Action
Usage	AAVerify [-Method **<Defined method to use>**] [-User **<Username>**] [-Tree **<Tree name>**] [?Result]
Arguments	Method The name of the advanced authentication method you wish to use. If not specified, AAVerify uses the method that was chosen during initial authentication to the directory. **NOTE:** You can specify multiple methods. User The name of the user you wish to use for the `AAVerify` command. If not specified, AAVerify reauthenticates the currently logged on user. Tree The name of the tree the user is in. You must use this with the User argument. [?Result] A variable name (preferably a temporary variable) that receives the result of the AAVerify. Set this variable to true for success or false for failure. ?AAVerifyReturnCode A variable that will get set with the error code that is generated from the AAVerify re-authentication process (if any).

Description	Use AAVerify with SecureLogin re-authentication, Novell Modular Authentication Service (NMAS), or Novell Lightweight Directory Access Protocol (LDAP) to verify the user. It is typically used before the application user name and password are retrieved and entered into the login box. This provides application re-authentication using a strong login method. For example, a user might be forced to enter their smart card and PIN before the application will log on using single sign-on, even though the application natively knows nothing about smartcards and PINs. If the verification succeeds, the [?Result] is set to true, otherwise it is set to false. These additions are for SecureLogin and NMAS. SecureLogin re-authentication may use one of the following methods: ◆ Password – your directory password. ◆ Smart card – if the smart card option has been selected during installation. ◆ Logon method – SecureLogin prompts for the same credentials as were used to log on for the current session. You can specify more than one method argument. In this case the user is allowed to re-authenticate with any of the specified methods. For example, you could use the command to request authentication using a fingerprint reader or smart card. NMAS or Novell LDAP-specific The method should be the name of the sequence as defined in the environment. If `AAVerify` is called with no arguments, then the currently logged on user is re-authenticated using the login method that they used for their current session. **NOTE:** When the `AAVerify` command is added to an application definition, it only increases the security of the target application if it is not possible to alter the application definition. If the application definition could be modified or overridden, then the `AAVerify` command could be removed and there would be no additional security. For this reason it is imperative that application definition access be restricted through directory access controls and SecureLogin's preferences, so that only a small, trusted group of administrators can modify, add and override application definitions.
Syntax examples	AAVerify AAVerify -Method "Enhanced Password" ?Result AAVerify -Method "Enhanced Password"-User "BSmith" - Tree "Production" ?Result

Example 1 Windows application definition

This example detects the login dialog box, but before SecureLogin enters the user's credentials, it prompts the user to provide their re-authentication credentials. The credentials are not submited until the re-authentication has succeeded.

```
# Logon Dialog Box
Dialog
   Title "Log on"
EndDialog
AAVerify -Method "Enhanced Password" ?Result
If ?Result Eq "True"
   Type $Username #1001
   Click #1
Else
   Click #2
   MessageBox "Authentication failed. Please verify your
smart card is inserted and your PIN is correct."
EndIf
```

Example 2 Windows application definition

The following example shows the use of exception handling with the `OnExceptions` command.

Refer to "OnException/ClearException" on page 121 for further details and examples of OnException usage.

```
Dialog
   Title "Log on"
EndDialog

OnException AAVerifyCancelled Call
CancelSimpleLoginDialogCancelled
OnException AAVerifyFailed Call
CancelSimpleLoginDialogFailed

AAVerify -method "smartcard"
Type $Username #1001
Click #1

Sub CancelSimpleLoginDialogCancelled
   Click #2
   EndScript
EndSub

Sub CancelSimpleLoginDialogFailed
   Click #2
   MessageBox "Your re-authentication failed. Log on
cancelled"
   EndScript
EndSub
```

Example 3	Windows application definition

The following example shows how to re-authenticate against the user's login method.

```
Dialog
   Title "Log on"
EndDialog

OnException AAVerifyFailed Call AAVerifyFailed
OnException AAVerifyCancelled Call AAVerifyCancelled

If ?isPin Eq "true"
   AAVerify -method "smartcard" ?result
Else
   AAVerify -method "password" ?result
EndIf

ClearException AAVerifyFailed
ClearException AAVerifyCancelled

Type $username
Type \n
Type $password
Type \n

Sub AAVerifyFailed
   Click #2
   MessageBox "Re-authentication failed."
   EndScript
EndSub

Sub AAVerifyCancelled
   Click #2
   EndScript
EndSub
```

Add

Used with	Startup, Terminal Launcher, Web, or Windows
SecureLogin version	3.0 or later
Type	Variable manipulator
Usage	Add **<Variable1> <Variable2>** [?Result]

Arguments	**<Variable1>** The first argument, the number to which the second argument will be added. This argument will also contain the result of the addition equation if the optional [?Result] argument is not passed in. If used without the [?Result] argument, <Variable1> must be a SecureLogin variable. Otherwise, <Variable1> can be any numeric value. **<Variable2>** The second argument, the number added to the first argument in the equation. <Variable2> can be a SecureLogin variable or numeric value. [?Result] Optional, the sum or result of the equation.
Description	Adds one number to another. The numbers can be written into the application definition or they can be variables. The result can be output to another variable or to one of the original numbers.

Syntax examples	`Add 1 2 ?Result`
	`Add ?LoginAttempts ?LoginFailures`
	`Add ?LoginAttempts ?LoginFailures ?Result`
	`Add ?LoginAttempts 3`
	`Add ?LoginAttempts 3 ?Result`

Example	Windows application definition
	
	This example reads the values of control IDs 103 and 104 into variables. From there they are added, and the result is typed into control ID 1
	
	`ReadText #103 ?Number1`
	`ReadText #104 ?Number2`
	`Add ?Number1 ?Number2 ?Result`
	`Type ?Result #1`

Attribute

Use with	Advanced Web application definition
SecureLogin version	3.5 or later
Type	Specifier
Usage	Attribute **<Attribute Name> <Attribute Name>**
Arguments	**< Attribute Name>** Name of the HTML attribute to discover. **< Attribute Value>** The value the above HTML attribute must contain for the condition to be true.

Description	Use the Attribute specifier in conjunction with the `Tag/EndTag` command to specify which HTML attributes and attribute values must exist for that particular HTML tag. For more information, see “Tag/EndTag” on page 172.

Example	This example finds the form that has an attribute of Name with a value of Logon.
	
	`Tag "Form"`
	`Attribute "Name" "Logon"`
	`EndTag`

AuditEvent

Use with	Startup, Terminal Launcher, Java, Web, or Windows application definitions to send an audit event to the Windows Event Log.
SecureLogin version	6.0 or later
Type	Specifier
Usage	AuditEvent [**<message>**]
Arguments	**<message>** The variable or text string passed to the Windows Event Log. **NOTE:** The functionality to send the contents of `$variable` or `?variable` to the Windows Event Log is only supported in SecureLogin 6.1SP1 or later
Description	Use `AuditEvent` to log SecureLogin events to the Windows Event Log. If the `ChangePassword` command is used to generate a `$password` variable, then a log entry is sent to the Windows Event Log.

Example	If the Audit platform agent is not present on the workstation nothing will be logged.
	
	`AuditEvent "message"`
	
	The parameter “message” is passed to the Windows Event Log.
	
	`AuditEvent $message`
	
	The parameter $message variable is passed to the Windows Event Log.

BeginSplashScreen/EndSplashScreen

Use with	Terminal Launcher (Generic and Advanced Generic only)
SecureLogin version	3.0.4 or later
Type	Action

Usage	BeginSplashScreen EndSplashScreen
Arguments	None
Description	Use to display splash screen across the whole Terminal Emulator window. This is used to mask any flickering caused by SecureLogin scraping the screen for text. A `Delay` command at the start of the application definition ensures the emulator window is in place before the splash screen is displayed.

Example	Terminal Launcher application definition
	
	This example launches the emulator and the SecureLogin waits 2 seconds for it to connect. The splash screen is displayed to cover the flickering, the login field is detected, the user name is entered, then the splash screen disappears.
	
	`Delay 2000`
	`BeginSplashScreen`
	`WaitForText "Login:"`
	`Type $Username`
	`EndSplashScreen`
	`Type @E`

BooleanInput

Use with	Advanced application definitions created using the Web Wizard, WinSSO, JavaSSO and .NetSSO workers.
SecureLogin version	3.5.x or later
Type	Action
Usage	BooleanInput #FormID:FieldID check "check"
Arguments	#FormID:FieldID The ID that was given to the matched field in the Site block using `MatchField` command. The FormID and FieldID must be unsigned integers. check "check" "check" is a Boolean value indicating a set or unset state for the specified field.
Description	Used inside a Site block to set the state of a Boolean field (either a check box or radio button).

Example	In this example the value of field #1:3 is being checked by the application definition.
	
	`# === Logon Application Definition #2 ==`
	`# === Google Initial Logon ====`
	`#=====================================`
	`Site Login -userid "Google Logon" -initial`
	`MatchDoimain "www.google.com"`
	`MatchField #1:1 -name "Email" -type "text"`
	`MatchField #1:2 -name "Passwd" -type "password"`
	`MatchField #1:3 -name "Cookie" -type "check"`
	`EndSite`
	`SetPrompt "Enter your user credentials"`
	`TextInput #1:1 -value "$Username"`
	`TextInput #1:2 -value "$Password"`
	`FocusInput#1:2 -focus "true"`
	`BooleanInput #1:3 -check "false"`
	`PressInput`
	`Endscript`

Break

Use with	Startup, Terminal Launcher, Web, or Windows
SecureLogin version	3.5 or later
Type	Action
Usage	Break
Arguments	None
Description	Use Break within the Repeat/EndRepeat commands to break out of a repeat loop.

Example 1	Windows application definition
	
	This example reads the screen and the content is searched for the words ‘log on’. If log on is found, the Repeat loop is broken and the application definition continues. If log on is not found, the application definition will check again.
	
	`Dialog`
	`  Class #32770`
	`  Title "Log on"`
	`EndDialog`
	`Repeat`
	`  ReadText  #301 "?Text"`
	`  If ?Text Eq "Log on"`
	`  Break`
	`  EndIf`
	`Delay 100`
	`EndRepeat`

Example 2	Terminal application definition

This example reads the terminal emulator screen and the content is searched for a successful log on (in this case the application main menu appears). Once the user has logged on, the Repeat loop is broken and the application definition continues. If the log on is not successful, the application definition will check again. Terminal emulators use repeat loops for error handling and to break out of the loop as appropriate.

```
# Initial System Login
WaitForText "ogin:"
Type $Username
Type @E
WaitForText "assword:
"Type $Password
Type @E
Delay 500
# Repeat loop for error handling
Repeat
Check to see if password has expired
If -Text "EMS: The password has expired."
   ChangePassword $Password
   Type $Password
   Type @E
   Type $Password
   Type @E
EndIf
#User has an invalid Username and / or Password stored.
   If -Text "Log on Failed"      DisplayVariables "The
username and / or password stored by SecureLogin is
invalid. Please verify your credentials and try again. IT
x453."
   Type $Username
   Type @E
   Delay 500
   WaitForText "assword:"
   Type $Password
   Type @E
   Delay 500
   EndIf#

Account is locked for some reason, possibly inactive.
If -Text "Account Locked" MessageBox "Your account has
been locked, possibly due to inactivity for 40 days.
Please contact the administrator on x453." EndIf # Main
Menu, user has logged on #successfully. If -Text
"Application Selection" Break
EndIf
Delay100
EndRepeat
```

Call

Use with	Startup, Terminal Launcher, Web, or Windows
SecureLogin version	3.5 or later
Type	Flow control
Usage	Call **<SubRoutine>**

Arguments	**<SubRoutine>** The name of the subroutine called. This must be identical to the name given in the `Sub` command.
Description	Use the `Call` command to call and run a subroutine. When a subroutine is called, the application definition begins executing from the first line of the subroutine. When it is completed, the application definition resumes executing from the command immediately following the `Call` command. Subroutines are useful when you would otherwise have to repeat the same lines of application definition over again.

Example	Terminal application definition
	
	This example looks for the word Username, if it is found on the screen the subroutine Log on is launched. If Wrong Password is found, the subroutine WrongPassword is launched.
	
	`Repeat`
	`If -Text "Username"`
	`  Call "Login"`
	`EndIf`
	`If -Text "Wrong Password"`
	`  Call "WrongPassword"`
	`EndIf`
	`EndRepeat`
	`#==Login Subroutine==`
	`Sub Login`
	`  Type $Username`
	`  Type @E`
	`  Type $Password`
	`  Type @E`
	`EndSub`
	`#==Wrong Password Subroutine==`
	`Sub WrongPassword`
	`  DisplayVariables "The password entered is incorrect. Please check your password and click OK to try again. IT x4532."`
	`  $Password`
	`  Call Login`
	`EndSub`

ChangePassword

Use with	Startup, Terminal Launcher, Web, or Windows
SecureLogin version	3.5 or later
Type	Action
Usage	ChangePassword **<Variable>** [**<Text>**] "Random"

Arguments	**<Variable>** A normal or runtime variable in which the password is stored. **[Text]** The text you want displayed in the change password dialog box. **[Random]** Random will invoke the random password generator.
Description	Use ChangePassword to change a single variable in scenarios where password expiry is an issue. Set the <Variable> to the new password. The flag for this command is Random. If Random is: ◆ Set, the new password is generated automatically in compliance with the variable's password policy. ◆ Not set, a dialog box prompts the user to enter a new password. The new password is tried against any variable password policies that are in place. See also “RestrictVariable” on page 142.

Syntax examples	`ChangePassword $NewPassword`
	`ChangePassword ?NewPassword "Please enter a new password"`
	`ChangePassword ?NewPassword Random`

Example	Windows application definition
	
	This example detects the change password event. The application requires the current user name and password, and then a new password and confirmation of the new password. The application definition creates a backup of the old password in case the password change fails (which is detected by the message that is displayed), and then generates and enters a new password.
	
	`# Change Password Dialog`
	`BoxDialog`
	`Class #32770`
	`Title "Change Password"`
	`EndDialog`
	`Set $PasswordBackup $Password`
	`Type $Password #1015`
	`ChangePassword $Password Random`
	`Type $Password #1005`
	`Type $Password #1006`
	`Click #1#`
	`Change Password Failed Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Change Password Failed"`
	`EndDialog`
	`# Set the password back as the password change failed`
	`$Password $PasswordBackup`
	`MessageBox "The change password process failed. Please retry the password change at your next log on. IT x453."`

Class

Use with	Startup, Windows
SecureLogin version	3.5 or later
Type	Dialog specifier
Usage	Class **<Window-Class>**
Arguments	**<Window-Class>** A string specifying the window class that this statement will match.
Description	When a window is created, it is based on a template known as a window class. The `Class` command checks to see if the class of the newly created window matches its <Window-Class> argument. If the window: ◆ Matches the <Window-Class> argument, the application definition continues to the next line. ◆ Does not match the <Window-Class> argument, execution continues at the next dialog statement. **NOTE:** Use the Window Finder tool to determine the window class.

Example	Windows application definition
	
	This example checks the dialog box generated by the application to determine if the Window Class is #32770. If true and its title is log on, that section of the application definition will execute. If false, the application definition will check the next Dialog block.
	
	`# Logon Dialog Box`
	`Dialog`
	`Class "#32770"`
	`Title "Log on"`
	`EndDialog`
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

ClearPlat

For each dialog block in an application definition, the chosen user ID is reset and you must select it again. Select it again by using a `SetPlat` command or by having the user select again from a list.

When an application first presents a login screen, SecureLogin directs the user to select an appropriate user ID from a list. SecureLogin enters the selected user ID's credentials into the application and submits them.

Resolving issue of re-entering user ID details

If the login fails due to incorrect credentials, SecureLogin prompts the user to change the credentials. SecureLogin does not retain user ID details and prompts the user to re-enter them. However, this could result in changing the wrong credentials if the user selects the incorrect user ID.

To resolve this issue, use the SetPlat, ReLoadPlat and ClearPlat commands. ReloadPlat sets the current user ID to the one which was last chosen (for the given application) or leaves the user ID unset if a user ID has not been selected previously. ClearPlat resets the last chosen user ID.

See also "ReLoadPlat" on page 138 and "SetPlat" on page 155.

Use with	Startup, Terminal Launcher, Web, or Windows
SecureLogin version	3.6.0 or later
Type	Action
Usage	There are three main places where code needs to be added to use the `ClearPlat` command: **Application startup** When an application first starts up, use ClearPlat to clear the previously chosen platform. (Do this in a Windows application by adding an extra dialog statement for the main window.) **Change Credentials Canceled** Call ClearPlat if the user decides not to modify the chosen platform's credentials, thus giving them a chance to choose a different platform next time. **Successful Logon** Call ClearPlat to allow the user to log on again with a different platform at a later stage.
Arguments	None
Description	Use to reset the last chosen platform, causing subsequent calls to ReLoadPlat to do nothing.

Example — Windows application definition

```
#== BeginSection: Application startup ====
Dialog
Class "#32770"
Title "Password Test Application"
EndDialog
ClearPlat
# == EndSection: Application startup====
# ==== BeginSection: Log on ====
Dialog
Class "#32770"
Ctrl #1001
Title "Log on"
EndDialog
ReLoadPlat
SetPrompt "Username =====>"
Type $Username #1001
SetPrompt "Password =====>"
Type $Password #1002
SetPrompt "Domain =====>
"Type $Domain #1003
Click #1
# ==== EndSection: Log on ====

## ====BeginSection: Log on successful ====
Dialog
Class "#32770"
Title "Log on successful"
EndDialog
ClearPlat
```

Example (continued)

```
Click #2
# ==== EndSection: Log on successful ====

# ==== BeginSection: Log on failure ====
Dialog
Class "#32770"
Title "Log on failure"
EndDialog
Click #2
ReLoadPlat
OnException ChangePasswordCancelled Call
ChangeCancelled
ChangePassword $password
ClearException ChangePasswordCancelled
Type -raw \Alt+F
Type -raw L
# ==== EndSection: Log on Failure ====
# ==== BeginSection: Change Credentials Cancelled ====
Sub ChangeCancelled
ClearPlat
EndScript
EndSub
# ==== EndSection: Change Credentials
Canceled ===
```

ClearSite

Use with — Web Wizard

SecureLogin version	6.0 or later
Type	Action
Usage	ClearSite "SiteName"
Arguments	"SiteName" The name of the site to clear, as specified in the matching Site/ EndSite block that will be reset to initial.
Description	Used to clear the 'matched' status for a given site. This allows initial sites to match again and causes recent and subsequent sites to fail to match. The `ClearSite` command needs to have the complete URL specified in the line before the `ClearSite` command.

Examples	In this example? the user is redirected to the Google home page and any previous user information is cleared.
	
	`GotoURL "http://www.google.com"`
	
	`ClearSite Login`
	
	In this example, the `ClearSite` command is used with as part of conditional statement and if a particular condition is true the user information is cleared.
	
	`Site "Login" –subsequent`
	
	`MatchURL "here.now.com"`
	
	`endsite`
	
	`MessageBox "Would you like to login again?" -yesno ?Continue`
	
	`If ?Continue eq "Yes"`
	
	`  Call LoginSub`
	
	`Else`
	
	`  ClearSite Login`
	
	`EndIf`

Click

Use with	Web, WinSSO, JavaSSO and .NetSSO workers.
SecureLogin version	3.5 or later
Type	Action

Windows usage	Usage One: Click **<#Ctrl-ID>** [-Raw] [-Right] Usage Two: Click **<#Ctrl-ID >** [-Raw [-x **< X Co-ordinate >** -y **<Y Co-ordinate >**]] Usage Three: Click [-order] **<#Order-ID>**
Web usage	Click **<#Number>**
Arguments	**<#Ctrl-ID>** The ID number of the control to be pressed. [-order] If the control ID's are not constant, utilize the -order argument to instruct SecureLogin to type into a control based on the creation order and not the tab order. For more information on the -order argument usage, see “Example 4” on page 179. <#Order-ID> For Windows application definitions, this parameter specifies which control based on the creation order in which to type the text. **[-Raw]** -Raw eliminates the mouse and sends a direct click. [-Right] -Right, used only with the -Raw flag, will send a right mouse click. **<X Co-ordinate>** X represents the horizontal co-ordinate relative to the client area of the application (not the screen). **<Y Co-ordinate>** Y represents the vertical coordinate relative to the client area of the application (not the screen). **<#Number>** The pound/hash symbol followed by the sequential number/control ID of the button to be pressed. **Web specific** The number of the button is determined by the Web page layout. See the “DumpPage” on page 85. **Windows specific** This is the control ID. Use the Windows Finder tool to discover the control ID. **Java specific** The index to use is put in an example application definition created by the Java wizard.

Description	When used with Windows applications, the `Click` command sends a click instruction to the specified <#Ctrl-ID>. **NOTE:** If the button to be clicked does not have a control ID, the Type "\N" command will often click the default button in a Windows application. You can set the –Raw flag if the button or control does not respond to the `Click` command. The –Raw flag causes SecureLogin to emulate the mouse and send a direct click message to the control. Using the -Right flag with the -Raw flag sends a right-click to the control. Setting the <#Ctrl-ID> to 0 (zero) sends the click instruction to the window on which the application definition is running. If -Raw is specified, then you can set the X coordinate and the Y coordinates. These coordinates are relative to the client area of the application, not the screen. **NOTE:** The borders of Windows Vista windows are substantially wider than those of Windows XP windows. Consequently, if your application definition will be used on both operating systems, you should use coordinates towards the top left of a Vista button or the bottom right of an XP button to ensure the same button is clicked in both operating systems. When used with Web application definitions, the `Click` command takes a single argument, which is the sequential number on the page of the button to be pressed. Click #3 will click the third button on the page. Keep in mind that, due to Web page layout and design, the sequential order of the buttons may not be obvious, and that you may have to use the `DumpPage` command to discover the field layout (see “DumpPage” on page 85).

Syntax examples	`Click #1`
	`Click #1 -Raw -Right`
	`Click -X 12 -Y 24`
	`Click -order #1`

Example 1	Windows application definition
	
	This example detects the login dialog box, the user name and password are entered, and button number 1 (in this case the logon button) is clicked.
	
	`# Logon Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Log on"`
	`EndDialog`
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

Example 2	Web application definition
	
	This example enters the user name and password, and then the logon button is clicked.
	
	`Type $Username`
	`Type $Password Password`
	`Click #1`

Example 3	Windows application definition
	
	This example uses the Java application, so there is no control ID. Instead, the `Click` command is told to click a particular place on the window.
	
	`# Logon Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Log on"`
	`End Dialog`
	`Type $Username`
	`Type $Password`
	`Click -X 12 -Y 24`

Example 4	Windows application definition
	
	This example shows the use of the **-order** switch and demonstrates a possible 'order' of the parameter.
	
	`# Logon Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Log on"`
	`EndDialog`
	`Type $Username #1001`
	`Type #Password #1002`
	`Click -order #1`

ClickInput

Use with	Advanced Web application definitions created using the Web Wizard.
SecureLogin version	3.5.x or later
Type	Action
Usage	`ClickInput #FormID:FieldID`
Arguments	**<#FormID:FieldID>** The ID that is specified in the matched field in the Site block using MatchField command. The FormID and FieldID must be unsigned integers.
Description	Use this command to simulate pressing the Enter key.
Syntax examples	`ClickInput #1:3`

Example	This example uses the ClickInput command that simulates pressing the Sign In button after the username and password fields are filled for the www.google.com Web site.
	
	`# === Logon Application Definition #2 ==`
	`# === Google Initial Logon ====`
	`#=====================================`
	`Site Login -userid “Google Log On” -initial`
	`MatchForm #1 -name "log on"`
	`MatchDomain “www.google.com”`
	`MatchField #1:1 -name "Email" -type "text"`
	`MatchField #1:2 -name “Passwd” -type “password”`
	`MatchField #1:3 -name "Cookie" -type "check"`
	`EndSite`
	`SetPrompt "Enter your user credentials"`
	`TextInput #1:1 -value “$Username”`
	`TextInput #1:2 -value "$Password"`
	`FocusInput#1:2 -focus “true”`
	`BooleanInput #1:3 -check "false"`
	`ClickInput #1:4`
	`Endscript`

ConvertTime

Use with	Startup, Terminal Launcher, Web, or Windows
SecureLogin version	3.0.4 or later
Type	Variable manipulator
Usage	ConvertTime **<Time> <String Time>**
Arguments	**<String Time>** The output variable.
Description	Use to convert a numeric time value, for example, ?CurrTime(system), into a legible format and store it in <String Time>.

Example	Windows application definition
	
	This example converts the time to a readable format and displays it in a dialog box.
	
	`# Logon Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Log on"`
	`End Dialog`
	`ConvertTime ?CurrTime(system) ?Time`
	`MessageBox ?Time`

Ctrl

Use with	Startup, Windows, Java

SecureLogin version	3.5 or later
Type	Dialog specifier
Usage	Ctrl **<#Ctrl-ID>** [<Regular Expression>]
Arguments	**<#Ctrl-ID>**
	The ID number of the control to check.
	[<RegEx>]
	The regular expression.
Description	Use the `Ctrl` command to determine if a window or its children (any descendants) contains the control expressed in the <#Ctrl-ID> argument. The control ID number is a constant that is established at the time a program is compiled.
	Third party software control ID numbers may not be consistent from one version to the next. Use the Window Finder tool to determine the control ID.
	Using the [<RegEx>] argument adds a further check that allows the application definition to skip to the next command. If the text on the specified <#Ctrl-ID> does not conform to the [<RegEx>], the application definition will skip to the next dialog statement as though the <#Ctrl-ID> did not exist.
Syntax examples	`Ctrl #1` `Ctrl #1 "OK"`
Example	Windows application definition
	This example tests the dialog box to see if it contains the correct control IDs with the correct values. If any of the control IDs are missing, or the text does not match, the application definition passes on to the next dialog block.

```
# Logon Dialog Box
Dialog
Ctrl #1 "OK"
Ctrl #2 "Cancel"
Ctrl #3 "Help"
Title "Log on"
EndDialog

Type $Username
Type "\T"
Click #1
```

DebugPrint

Use with	All
SecureLogin version	6.0 or later
Type	Action
Usage	DebugPrint **<Data>**

Arguments	**<Data>** The text displayed to the user. Data can be several strings, variables, or a combination of both..
Description	Use the `DebugPrint` command to display the text specified in the <Data> variable on a Debug console. The command can take any number of text arguments, including variables (for example, DebugPrint "The user " $Username " has just logged onto the system").
Syntax examples	`DebugPrint "Caught the login dialog"` `DebugPrint "Setting platform to " ?Platform`

Example	Windows application definition
	
	This example displays the the text specified in the ?ServerName variable on the Debug console.
	
	`# Logon Dialog`
	`Dialog`
	`Class "#32770"`
	`Title "Log on"`
	`EndDialog`
	`ReadText #1003 ?ServerText`
	`RegSplit "Server: (.*)" ?ServerText ?ServerName`
	`DebugPrint "Setting the platform to " ?ServerName`
	`SetPlat ?ServerName`
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

Decrement

Use with	All
SecureLogin version	3.5 or later
Type	Variable manipulator
Usage	Decrement <Variable>
Arguments	**<Variable>** The name of the variable to decrease in value.
Description	Use the `Decrement` command to from a specified variable. For example, you can use decrement to count the number of passes a particular application definition has made. Once the number of instances is equal to the specified number, you can instruct the application definition to run another task or end the application definition. This is useful when configuring an application whose login panel is similar to other windows within the application, or to easily control the number of attempts a user can have to access an application. Also see “Increment” on page 103

Syntax examples	`Decrement ?RunCount`

Example	Windows application definition
	
	Each time the application definition is run, a variable is decremented. This example counts the number of times the dialog box is dis-played. If the dialog box is displayed more than three times, the application is closed. If the login is successful, the count is reset.
	
	`# Logon Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Log on"`
	`EndDialog`
	
	`Decrement ?RunCount`
	`If ?RunCount Gt "3"`
	`MessageBox "Log on has been attempted too many times.  The application will be closed."`
	`KillApp "app.exe"`
	`Else`
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`
	`EndIf`
	
	`# Logon Successful Message`
	`Dialog`
	`Ctrl #1`
	`Title "Logon Successful"`
	`EndDialog`
	
	`Set ?RunCount "0"`

Delay

Use with	All
SecureLogin version	3.5 or later
Type	Action
Usage	Delay **<Time Period>**
Arguments	**<Time Period>** A period of time, expressed in milliseconds (1/1000 of a second), during which application definition execution is paused.
Description	Use the `Delay` command to delay the execution of the application definition for the time specified in the <Time Period> argument. The time specified in the <Time Period> argument is noted in milliseconds (for example, Delay 5000 creates a 5-second pause). You can use the `Delay` command to accommodate an introduction screen or another custom feature.

Example	Windows application definition
	
	This example detects the login box, then the application definition waits half a second before acting upon it to make sure that the box is complete.
	
	`# Logon Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Log on"`
	`EndDialog`
	
	`Delay 500`
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

Dialog/EndDialog

Use with	Java, Windows
SecureLogin version	3.5 or later
Type	Dialog specifier
Usage	Dialog/EndDialog
Arguments	None
Description	Use the Dialog/EndDialog command to identify the beginning and end of a dialog specification block respectively. You can use these commands to construct a dialog specification block, which consists of a series of dialog specification statements (for example Ctrl, Title, and so on). When a dialog block is executed, each of the dialog specification statements is executed in sequence. If any statement within the dialog block is not found, the entire dialog block is considered false, and the application definition execution proceeds to the next dialog block, if any. You need to specify as much information in the dialog block to make the dialog box (for example, Log on, Change Password, and so on) unique. The portion of the application definition that follows the `EndDialog` command is called the application definition body. Another dialog block, or the end of the application definition, terminates the application definition body.

Example	Windows application definition
	
	This example tests the dialog box in order to determine its identity. If it is determined to be the login box, the application definition will parse the Type and Click commands to complete the login process.
	
	`# Logon Dialog Box`
	`Dialog`
	`Ctrl #1 "OK"`
	`Title "Log on"`
	`Parent`
	`Title "Application 1"`
	`EndParent`
	`EndDialog`
	
	`Type $Username #1001`
	`Type $Password #1002`
	
	`Click #1`

DisplayVariables

Use with	All
SecureLogin version	3.5 or later
Type	Action
Usage	DisplayVariables [**<User Prompt>**] [**<Variable>** [**<Variable>**] …]
Arguments	**[<User Prompt>]** Optional, customized text displayed in the Enter SecureLogin Variables dialog box. This message can be defined in a runtime variable (see example 2). **[<Variables>]** The name of the variables for which you want the user prompted. If not specified, SecureLogin will prompt for all variables that are used by the application definition.

Description	Use the `DisplayVariables` command to display a dialog box that lists the user's stored variables (for example, $Username and $Password) for the current application. **About editing variables** The user can edit the variables from this dialog box. For example, if the login process is unsuccessful due to an incorrect user name or password, the `DisplayVariables` command prompts the user to edit the stored user name or password values. The login process proceeds as normal from that point. You can also specify a particular variable to display. If the <Variables> parameter is specified, DisplayVariables prompts only for the variables specified. Enter the replacement text in quotation marks after the `DisplayVariables` command. This replaces the default prompt text in the Enter SecureLogin Variables dialog box. If there are no variables stored for the user, the first time SecureLogin attempts to single sign-on to the application, the prompt will not be customized. Once there are variables stored for the user, the prompt will be customized when the application definition is run.The `SetPrompt` command can also be used to customize the prompt text in the dialog box. **NOTE:** You can use the `OnException EnterVariablesCancelled` command to prevent a user from canceling the DisplayVariables prompt.

Syntax examples	`DisplayVariables`
	`DisplayVariables "Please enter your details"`
	`DisplayVariables "Please enter a new password" $Password`
	`DisplayVariables "Please enter your username and password" $Username $Password`
	`DisplayVariables "" $Username $Password`

Example 1	Windows application definition
	
	This example detects the Wrong Password dialog box, and SecureLogin prompts the user to enter a new user name and password. Once specified, SecureLogin enters them into the dialog box, and the user clicks **OK**.
	
	`# Wrong Password`
	`Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Wrong Password"`
	`EndDialog`
	`DisplayVariables "Enter a new username and password"?$Username $Password`
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

Example 2	Windows application definition
	
	This examples passes the message in as a variable.
	
	`Dialog`
	`Class "Notepad"`
	`Title "Untitled - Notepad"`
	`EndDialog`
	`Set ?Vars "\$Username"`
	`Set ?Msg "This is a DisplayVariables message"`
	`DisplayVariables ?Vars`
	`DisplayVariables ?Msg $Password`
	`DisplayVariables "testing" ?Vars`
	`DisplayVariables "testing" $Password $Username`

Divide

Use with	Startup, Terminal Launcher, Web, or Windows
SecureLogin version	3.0 or later
Type	Variable manipulator
Usage	Divide **<Variable1> <Variable2>** [?Result]
Arguments	**<Variable1>** The dividend, the first argument, the number that is divided by the second argument. Also this argument contains the result if the optional [?Result] argument is not passed in. If used without the [?Result] argument, <Variable1> must be a SecureLogin variable, either?Variable1 or $Variable1. Otherwise <Variable1> can be any numeric value. **<Variable2>** The divisor, the second argument, the number by which the first argument is divided. <Variable2> can be a SecureLogin variable or a numeric value. [?Result]Optional, the quotient, the result of the equation.
Description	Use to divide one number by another. The numbers can be written into the application definition or they can be variables. The result can be output to another variable or to one of the original numbers. **NOTE:** This is an integer arithmetic that is 5/2, not 2.5.

Syntax examples	`Divide "1" "2" ?Result`
	`Divide ?LoginAttempts ?LoginFailures`
	`Divide ?LoginAttempts ?LoginFailures ?Result`
	`Divide ?LoginAttempts "3"`
	`Divide ?LoginAttempts "3" ?Result`

Example	Windows application definition
	
	This example read the values of control IDs 103 and 104 into variables. From there they are divided,and typed into control ID 1.
	
	`ReadText #103 ?Number1`
	`ReadText #104 ?Number2`
	`Divide ?Number1 ?Number2 ?Result`
	`Type ?Result #1`

DumpPage

Use with	Advanced Web application definition
SecureLogin version	3.5 or later
Type	Action
Usage	DumpPage **<Variable>**
Arguments	**<Variable>** The string variable to receive the page information.
Description	Use the `DumpPage` command to provide information about the current Web page. Use for debugging Web page application definitions.
Example	`DumpPage ?dump` `MessageBox ?dump`

EndScript

Use with	Startup, Terminal Launcher, Web, or Windows
SecureLogin version	3.5 or later
Type	Action
Usage	EndScript
Arguments	None
Description	Use the `EndScript` command to immediately terminate execution of the application definition.

Example	Windows application definition
	
	This example detects the login box, then SecureLogin enters the user name and password, and the user clicks OK. If the Incorrect Password message is detected, SecureLogin displays a message that the password was incorrect, and terminates the application definition.
	
	`Dialog`
	`Title "Logon Failure"`
	`Ctrl #1`
	`EndDialog`
	
	`ReadText #65535 ?ErrorMsg`
	`If "Incorrect Password" -In ?ErrorMsg MessageBox "You have entered an incorrect password"`
	`EndScript`
	`EndIf`

Event/Event Specifiers

Use with	Windows
SecureLogin version	3.5 or later
Type	Dialog specifier
Usage	Event <Event Specifier>
Arguments	**<Event Specifier>** The application event to monitor. This corresponds to a Windows event, which usually begins with WM_. For example, WM_COPYDATA, WM_GETOBJECT, WM_GETTEXT For detailed information on Windows events, see the Microsoft Developer network Web site. (http://msdn.microsoft.com). Microsoft's Spy++, or similar Windows message spy tools, are also useful for trapping event names in specific windows. Information on Spy ++ is also available on the MSDN Web site.
Description	Application definitions generally execute at the point when an application window is created. This corresponds to the WM_CREATE message that is received from an application window at start up. By adding the Event specifier to a dialog block, you can override this behavior, such that an application definition only executes when (and only when) the specified message is generated. If no Event specifier is given, it is equivalent to Event WM_CREATE. You can only apply the Event specifier within a Dialog and EndDialog statement block. Only one Event may be specified per Dialog block. If there is a requirement to monitor for multiple events, each must be specified within their own Dialog block. For more information, see MSDN or other documentation on the Win32 messaging system.

Syntax examples	`Dialog`
	`Class "someclass"`
	`Event WM_ACTIVATE`
	`EndDialog`
	`MessageBox "Caught the WM_ACTIVATE message"`

FocusInput

Use with	Startup, Terminal Launcher, Web or Windows and advanced application definitions created using the Web Wizard, WinSSO, JavaSSO and .NetSSO workers.
SecureLogin version	3.5.x or later
Type	Action
Usage	FocusInput #FormID:FieldID
Arguments	#FormID:FieldID The ID that was given to the matched field in the Site block using `MatchField` command. The FormID and FieldID must be unsigned integers.
Description	Used to focus on an input field.

Example	In this example the value of field #1:2 is being checked by the application definition.
	
	`# === Logon Application Definition #2 ==`
	`# === Google Initial Logon ====`
	`#=======================================`
	`Site Login -userid "Google Logon" -initial`
	`MatchDoimain "www.google.com"`
	`MatchField #1:1 -name "Email" -type "text"`
	`MatchField #1:2 -name "Passwd" -type "password"`
	`MatchField #1:3 -name "Cookie" -type "check"`
	`EndSite`
	`SetPrompt "Enter your user credentials"`
	`FocusInput #1:1`
	`TextInput #1:1 -value "$Username"`
	`FocusInput #1:2`
	`TextInput #1:2 -value "$Password"`
	`FocusInput #1:3`
	`BooleanInput #1:3 -check "false"`
	`Endscript`

GenerateOTP

GenerateOTP command supports two types of usage:

- “AISC Usage” on page 88
- “HOTP Usage” on page 89

AISC Usage

Use with	Startup, Terminal Launcher, Web, or Windows
SecureLogin version	3.5.0 or later
Type	Action
Usage	GenerateOTP -mode <string>-challenge <string>
Arguments	**<result>** A variable that receives the value of the one-time password (OTP) that is generated. **-mode** Specifies the type of OTP that is dynamically generated. The default value for mode is set to AISC-SKI for smartcard OTP. Setting this to AISC-SKI makes SecureLogin use algorithm to generate an OTP based on the user's smart card. This setting is deprecated and can be removed. **-challenge** When the OTP generated is based on a challenge/response or asynchronous mode, the challenge needs to be passed to the `GenerateOTP` command as an argument, normally by means of a script that reads the challenge from the screen.
Description	OTP is an authentication method specifically designed to avoid the security exposures inherit in traditional fixed and static passwords. OTPs rely upon a predefined relationship between the user and the authenticating server. The encryption key is shared between the user's token generator and the server, with each performing the pseudo-random code calculation at user logon. If the codes match, the user is authenticated. The `GenerateOTP` command incorporates OTP token generation functionality embedded in smartcard technology. The soft tokens can be generated in synchronous and asynchronous modes which now allows soft tokens to be loaded onto mobile devices such as PDAs or be sent to cell phones as SMS text messages. **Synchronous mode:** Synchronous authentication of timeplus-event authentication replaces static alphanumeric passwords with a pseudo-random code that is dynamically generated at configured time intervals, generally around once a minute. The pseudo-random code is based on a shared encryption key and the current time. **Asynchronous mode:** Asynchronous authentication or challenge response authorization replaces static alphanumeric passwords with a pseudo-random code that is dynamically generated based on a shared encryption key, the current time and a challenge/response combination. In Asynchronous mode the challenge must be passed to the `GenerateOTP` command as an argument. The application definition asynchronous example shows a typical command structure to enable OTP for use with smart card technology.

Example	In SecureLogin version 6.0, the `GenerateOTP` command was enhanced to integrate with smart cards.
	
	In Synchronous mode the `GenerateOTP` command requires the administrator to pass the -mode variable, AISC-SKI, to the command.
	
	In this instance AISC-SKI is the smart card and SKI is the name of the applet used on the smart card.
	
	An example application definition enabling synchronous OTP encryption key distribution for use with smart cards is as follows:
	
	`Dialog`
	`Title "Test App"`
	`EndDialog`
	`GenerateOTP -mode "AISC-SKI" ?OtpResult`
	`Type ?OtpResult #14`
	
	In Asynchronous mode the challenge must be passed to the `GenerateOTP` command as an argument. This requires a script that reads the challenge variable from the screen.
	
	An example application definition enabling asynchronous OTP encryption key distribution for use with smartcards is as follows:
	
	`Dialog`
	`Title "Test App"`
	`EndDialog`
	`ReadText #12 ?tmp`
	`GenerateOTP -mode "AISC-SKI" -challenge ?tmp ?Otp`
	`Type ?Otp #14`
	
	It is assumed that a call without a challenge passed in is synchronous. The -mode parameter, instead of being passed in via the script, can also be created as a single sign-on variable in the script platform.
	
	If the -mode parameter is not passed in as a parameter to the `GenerateOTP` command, SecureLogin will check for a variable named mode. Values passed into the command via the script will override values defined as variables. This is for future integration with SecureLogin for Mobiles.
	
	NOTE: It is assumed that the `acomx.dll` is present on the machine and in the path. If not, then additional code may be required to specify the location of this library file.
	
	The smartcard is assumed to be in the card reader at OTP generation time and a single card reader is also assumed.
	
	If the user's smart card has not been authenticated the user will be prompted to enter a PIN to unlock the card. This is required only once as the PIN is normally cached.

HOTP Usage

Use with	Startup, Terminal Launcher, Web, or Windows
SecureLogin version	7.0 SP2 or later

Type	Action
Usage	`GenerateOTP -METHOD=<XMethod> -MODE=<XMode> ?OTPResult`
Arguments	**-METHOD** Defines the method or algorithm to generate the OTP. You can use the following value: ◆ OATH-HOTP **-MODE** Defines the mode of OTP generation and usage - software, hardware or smart token. You can use any one of the following values: ◆ SOFT ◆ HARD ◆ AI-SC(Smart Card) **?OTPResult** A variable that receives the value of the one-time password (OTP) that is generated.
Description	SecureLogin will enable using wizards to configure applications supporting OTP based authentication. Using wizards, users can configure OTP algorithm specific parameters with the options available to choose from using the wizard.

Example	`# place your application definition here.`
	`Set ?OTPCredential "<NOTSET>"`
	`GenerateOTP -method "OATH_HOTP" -mode "SOFT" ?OTPCredential`
	`If ?OTPCredential Eq "<NOTSET>"`
	`DisplayVariables "Please specify all information." $OTPSecretKey $OTPCounter $OTPDigit`
	`GenerateOTP -method "OATH_HOTP" -mode "SOFT" ?OTPCredential`
	`EndIf`
	`MessageBox "OTP Generation Success"`
	`MessageBox ?OTPCredential`

GetCheckBoxState

Use with	Advanced Web, Windows
SecureLogin version	3.5 or later
Type	Action
Usage	GetCheckBoxState **<#Item Number> <Variable>**

Arguments	**<Item Number>** The ID of the check box. **<Variable>** The target variable for the status of the specified check box. Value returned is Checked or Unchecked. Partially selected tristate check boxes will be returned as Unchecked. The variable can be a question mark (?) or a dollar sign ($) variable.
Description	Use the `GetCheckBoxState` command to return the current state of the specified check boxes.

Example	`GetCheckBoxState #25 ?state1`
	`GetCheckBoxState #26 ?state2`
	`MessageBox ?state1`
	`MessageBox ?state2`

GetCommandLine

Use with	Startup, Windows
SecureLogin version	3.0.4 or later
Type	Action
Usage	GetCommandLine**<Variable>**
Arguments	**<Variable>** This variable defines where to store the captured command line.
Description	Use the `GetCommandLine` command to capture the full command line of the program that is loaded and save it to the specified variable. **NOTE:** You can use the GetCommandLine to detect and differentiate backend systems and databases for use with multiple logons in the SAP application.

Example	Windows application definition
	
	This example reads the command line of the application, and then tests the line to see if it is Notepad.exe. If it is, Notepad is closed. If it is not, the application definition ends.
	
	`GetCommandLine ?Text`
	`If ?Text Eq "\"C:\Windows\System32\notepad.exe\""`
	`  KillApp Notepad.exe`
	`EndIf`

GetEnv

Use with	All
SecureLogin version	3.5 or later

Type	Action
Usage	GetEnv **<EnvVar> <Variable>**
Arguments	**<EnvVar>** This is the environment variable name you wish to retrieve. **<Variable>** This variable defines where to store the retrieved environment variable data.
Description	Use the `GetEnv` command to read the value of an environment variable and saves it in the specified variable.

Example	Windows application definition
	
	`GetEnv "SESSIONNAME" ?SessionName`
	`If ?SessionName eq "console"`
	`MessageBox "Running from Citrix Server Console"`
	`EndIf`

GetHandle

Use with	Windows
SecureLogin version	6.1.0 or later
Type	Action
Usage	GetHandle **<Variable>**
Arguments	**<Variable>** This variable defines where to store the captured handle.
Description	Use GetHandle to capture the unique handle of the window that the Windows application definition script is activated on. GetHandle is used to retrieve the handle so that the value is passed to TLaunch.exe to inform the terminal launcher what window to interact with, or to pass the value to any other application.

Example 1	Windows application definition
	
	`GetHandle ?winHandle`
	`MessageBox ?winHandle`

Example 2	Windows application definition
	
	`GetReg "HKLM\Software\Microsoft\Windows\CurrentVersion\App Paths\SLProto.exe\Path" ?SLLocation`
	`If ?SLLocation eq "<NOTSET>"`
	`EndScript`
	`EndIf`
	
	`GetHandle ?PuttyHWND`
	`Strcat ?TLaunch ?SLLocation "tlaunch.exe"`
	`Strcat ?TLaunchHWND "/hwnd" ?PuttyHWND`
	`Run ?TLaunch "/auto" "/ePutty" "/l" "/pPutty - Detection and Login" "/t" "/q" "/s" ?TLaunchHWND`

GetIni

Use with	Windows, Web, Terminal Emulator, Java
SecureLogin version	3.5 or later
Type	Action
Usage	GetIni **<ini file> <section> <key> <variable>**
Arguments	**<Ini File>** This is the file name from which you wish to read the section or key. **<Section>** Name of the section that contains the key name. **<Key>** Name of the key to read. **<Variable>** This variable defines where to store the retrieved data from the `ini` file.
Description	Use the `GetIni` command to read data from INI file.
Example	Windows application definition `GetIni "C:\Program Files\Lotus\Notes\Notes.ini" "Notes" "KeyFileName" ?NotesDefaultIDFileSetPlat`

GetMD5

Use with	Windows
SecureLogin version	6.0 or later
Type	Action

Usage	GetMD5 **<Variable>**
Arguments	**<Variable>** Variable to store the returned MD5 hash value.
Description	Use the `GetMD5` command to generate an MD5 hash value of the current process the script is running for. `GetMD5` will only work with Win32 scripts. Message-Digest algorithm 5 (MD5) is employed in SecureLogin and can be used to check the integrity of files against a known hash value. MD5 hash values are widely used in software to provide assurance that a particular file has not been altered. The administrator can compare a published MD5 sum with the checksum of another file to recognize corrupt or incomplete files, particularly for large executable files.

Example	In a Windows application definition, the MD5 hash value is stored in the variable that was passed as the argument to the command. The variable can either be a temporary or stored variable type.
	
	`GetMD5 ?tmp`
	
	or
	
	`GetMD5 $hash_value`
	
	The MD5 hash value would normally be obtained with the Window Finder tool on a window from the application. This MD5 value will then be put into a script and compared against the results of the `GetMD5` command. If the MD5 hash values do not match, the executable file may have been altered.

GetReg

Use with	All
SecureLogin version	3.5 or later
Type	Action
Usage	GetReg **<RegEntry> <Variable> [<platform>]**

Arguments	**<RegEntry>** This is the registry entry to read. **<Variable>** This variable defines where to store the retrieved environment variable data. **<platform>** This optional variable reads the worker registries. The following are the platform inputs: ◆ **<x64>** It reads 64 bit registries even when the worker is 32-bit. It ignores 32-bit windows. ◆ **<x32>** It reads 32-bit registries even when the worker is 64-bit. ◆ **<os>** It reads the registries based on the operating system. It reads 32-bit registries on 32-bit Windows and 64-bit registries on 64-bit windows.

Description	Use the `GetReg` command to read data from the registry and save it in the specified variable.
	
	The following is format for the registry entry input: `HIVE\KEY\Value`
	
	Valid hives are:
	
	`"HKCR", "HKEY_CLASSES_ROOT", "HKCC "HKEY_CURRENT_CONFIG", "HKCU"`
	`"HKEY_CURRENT_USER "HKLM "HKEY_LOCAL_MACHINE "HKU "HKEY_USERS`

Example	Windows application definition
	
	`GetReg "HKLM\Software\ABCCorp\ProductID" ?ProductID`
	`If ?ProductID noteq "xxxxxxxxxx"`
	`  #Not corporate desktop`
	`  EndScript`
	`EndIf`

GetDirectoryStatus

NOTE: The status of the primary datastore connection can be found in the SecureLogin About box

Use with	All
SecureLogin version	7.0 or later
Type	Variable manipulators
Usage	GetDirectoryStatus **<?StatusVariable>**

Arguments	**<?StatusVariable>** The target variable to which the value of the primary status is copied. The value returned is either online or offline.
Description	Use the `GetDirectoryStatus` command to find out whether SecureLogin can connect to the directory or not. The status is online if the network is up, SecureLogin connects to the directory, and the user is working in online mode. The status is offline if the network is down, the network is up but the directory is unavailable, or the user chose to work in offline mode.

Example	`GetDirectoryStatus ?status`
	`If ?status eq "online"`
	`#online instructions`
	`EndIf`
	`If ?status eq "offline"`
	`#offline instructions`
	`EndIf`

GetSessionName

Use with	Terminal Emulator
SecureLogin version	3.5 or later
Type	Action
Usage	GetSessionName **<?Variable>**
Arguments	**<Variable>** The target variable that the session name is copied into.
Description	Use the `GetSessionName` command to find the current HLLAPI session name that is used to connect and returns it to the specified variable. This command is only valid for tlaunch emulator definitions with the type HLLAPI.
Example	Terminal Emulator application definition `GetSessionName ?Session_name`

GetText

Use with	Web, Terminal Launcher
SecureLogin version	3.0 or later
Type	Action
Usage	GetText **<Variable>**

Arguments	**<Variable>** This variable defines where to store the captured text.
Description	Use the `GetText` command to get all of the text from the screen and save it to the specified variable. It is used in large Web application definitions that contain several If -Text statements. In Netscape, each If -Text statement screen scan to find the specified text caused the screen to flicker. However, using GetText (for example If ?Text -in ?FromGetText), the application definition can contain multiple If -Text commands with only one scan of the screen.

Example	Web application definition
	
	This example copies the text content of the Web page to the ?Text variable. SecureLogin tests for the presence of the word ‘Logon’. If Logon exists, SecureLogin enters the credentials and submits them automatically.
	
	`GetText ?Text`
	`If "Log on" -In ?Text`
	`Type $Username`
	`Type $Password Password`
	`EndIf`

GetURL

Use with	Web
SecureLogin version	3.0 or later
Type	Action
Usage	GetURL **<Variable>**
Arguments	**<Variable>** This variable defines where to store the captured URL.
Description	Use the `GetURL` command to capture the URL of the site that is loaded and save it to the specified variable.

Example	Web application definition
	
	This example copies the URL of the Web site to the ?URL variable and tests the URL to see if it matches text being searched for. If it does, SecureLogin pops up a message box and redirects the user to the intranet.
	
	`GetURL ?URL`
	`If "Log off" -In ?URL`
	`MessageBox "You have chosen to log off the application.`
	`You will now be redirected to the intranet home page."`
	`GoToURL "http://Intranet"`
	`EndIf`

GoToURL

Use with	Web
SecureLogin version	3.5 or later
Type	Action
Usage	GoToURL **<URL> [<-frame>]**
Arguments	**<URL>**
	The URL to which the browser will navigate.
	<-frame>
	Opens the URL in the frame which started the application definition.
Description	Use the `GoToURL` command to make the browser navigate to the specified URL. By default the command opens the new Web page in the main window, rather than the frame that started the application definition.
	When using the -frame option on a framed Web page, the URL redirect occurs only in the current frame rather than the parent window.
	You must include http://.
Example	Web application definition
	This example detects an incorrect password message, displays a message box informing the user, and then browses to the NetIQ Web site.

```
If -Text "Incorrect Password"
  MessageBox "You have entered an incorrect password"
  GoToURL "http://www.NetIQ.com"
EndIf
```

Highlight

Use with	Web
SecureLogin version	3.5 or later
Type	Action
Description	Use the `Highlight` command to set the focus of the Web page on a field.
	The command is useful for pages that do not have any control selected after loading or for any fields that change the behavior after gaining focus.
	This command is functionally equivalent to the SetFocus command in Windows scripts.

Example	Web application definition

```
If –Text "Logon"
  Highlight #1
  Type $Username #1
  Highlight #2
  Type $Password #2
  Type “\N”
EndIf
```

If/Else/EndIf

Use with	Startup, Terminal Launcher, Web, or Windows
SecureLogin version	3.5 or later
Type	Flow control

Usage 1	`If` **`<Value1>`** `<Gt|Lt>` **`<Value2>`**
	`#Do This`
	`[Else]`
	`#Do This`
	`EndIf`

Usage 2	`If` **`<Value1>`** `<Eq|NotEQ >` **`<Value2>`** `[-I|-S]`
	`#Do This`
	`[Else]`
	`#Do This`
	`EndIf`

Usage 3	`If` **`<Value1>`** `<-In|-NotIn>` **`<Value2>`** `[-I|-S]`
	`#Do This`
	`[Else]`
	`#Do This`
	`EndIf`

Usage 4	`If -Text [-Frame]` **`<Text>`**
	`#Do This`
	`[Else]`
	`#Do This`
	`EndIf`

Usage 5	`If -Exist|-NotExist` **`<Variable>`**
	`#Do This`
	`[Else]`
	`#Do This`
	`EndIf`

Arguments	**<Value1>** The left side of the expression for evaluation. **<Value2>** The right side of the expression for evaluation. **<Text>** The text for which you are searching.

Description

Use the `If` command to establish a block to execute if the expression supplied is true. The `Else` command works inside an If block. The `Else` command is executed if the operator in the If block is false. Use the `EndIf` command to terminate the If block.

Text comparison operators supported The text comparison operators supported by the If command are:

- Eq: True if the left side is equal to the right side.
- NotEQ: True if the left side is not equal to the right side.
- -In: True if the left side is a substring of the right side.
- -NotIn: True if the left side is not a substring of the right side.
- -SiteDeparted: Checks if the current document is still active or not.

When using these text comparison operators, you may optionally specify whether the comparison is to take into account the case of the strings being compared. If -I is specified, the comparison is case insensitive. If -S is specified, then the comparison is case sensitive. By default the Eq and NotEQ operators are not case sensitive, while the -In and -NotIn operators are case sensitive.

An operator is also supplied to directly query the application for a particular string:-Text: Evaluates to true if the specified text is found in the application windows of the application. For Internet Explorer application definitions, you can supply an optional -Frame argument, which restricts the command to look for the specified text in the current frame.

Numerical comparison operators supported Two numerical comparison operators are supported by the `If` command, Gt and Lt. The command evaluates to true if the left side is greater than or less than (respectively) the right side. This is a numerical comparison, so the left and right sides must be numbers.

An operator is supplied to check for the existence of a stored variable:

- -Exists: True if the specified variable exists.
- -NotExist: True if the specified variable does not exist.

Syntax examples

```
If $Number NotEQ "1"
MessageBox "NotEQ 1"
Else
MessageBox "Eq 1"
EndScript
EndIf

If ?Value1 Gt ?Value2
If -Text "Log on"
If -Exists $RunBefore
If "Log on" -In ?Text
```

Example 1 Web application definition

This example tests for an incorrect password. If it is found, an incorrect password message box is displayed. If the error message is not found, SecureLogin logs in as normal.

```
If -Text "Incorrect Password"
DisplayVariables "You have an incorrect password. Please
verify it and retry log on."
EndScript
Else
Type $Username
Type $Password Password
EndIf
```

Example 2 Windows application definition

Each time the application definition is run, a variable is incremented. This example counts the number of times the dialog box is displayed. If it is displayed more than three times, the application is closed. If the log on is successful, the count is reset.

```
# Logon Dialog Box
Dialog
   Class #32770
   Title "Log on"
EndDialog

ReadText #1001 ?Username

If –Exists $Username
Else
   Set $Username ?Username
EndIf
Increment ?RunCount
If ?RunCount Gt "3"
MessageBox "Log on has been attempted too many times.
The application will be closed."
KillApp "app.exe"

Else

   Type $Username #1001

   Type $Password #1002

   Click #1

EndIf

# Logon Successful Dialog Box
Dialog
   Ctrl #1
   Title "Log on successful"
EndDialog

Set ?RunCount "0"
```

Example 3	Web application definition
	
	This example copies the text content of the Web page to ?WebText. The variable is then tested to see if 'Log on' is present. If it is, SecureLogin performs the login process. If it is not present, the application definition is terminated.
	
	`GetText ?WebText`
	`If "Log on" –In ?WebText`
	`  Type $Username`
	`  Type $Password Password`
	`Else`
	`  EndScript`
	`EndIf`

Example 4	Startup
	
	This example tests, upon SecureLogin loading, to see if SecureLogin has been run by the user. If it has not, SecureLogin sets the variable so that the message is only displayed once, and then displays a welcome message along with the option for further details on SecureLogin.
	
	`If –NotExist $LoadedBefore`
	`  EndScript`
	`Else`
	`MessageBox –YesNo ?Result "Welcome to SecureLogin Single Sign-On, a new password management tool that will save you the hassle of remembering your passwords. Would you like more details on how to use SecureLogin and what it can do for you?"`
	`Set $LoadedBefore "Yes"`
	`If ?Result Eq "Yes"`
	`GoToURL "http://www.NetIQ.com/securelogin.htm"`
	`EndIf`
	`EndIf`

Include

Use with	All
SecureLogin version	3.0 or later
Type	Flow control
Usage	Include **<Platform-Name>**
Arguments	**<Platform-Name>** The name of the application definition to include.
Description	Use the `Include` command to share commonly-used application definition commands by multiple applications. The application definition identified by <Platform-Name> is included at execution time into the calling application definition. The application definition included with the `Include` command must comprise commands supported by the calling application.

Example	Windows application definition
	
	This example detects the login dialog, the Notepad.exe application definition is executed, and then the user's credentials are entered.
	
	`# Logon Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Log on"`
	`EndDialog`
	
	`Include "Notepad.exe"`
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

Increment

Use with	All
SecureLogin version	3.5 or later
Type	Variable manipulator
Usage	Increment **<Variable>**
Arguments	**<Variable>** The name of the variable to increase in value.
Description	Use the `Increment` command to add to a specified variable. For example, you can use increment to count the number of passes a particular application definition has made. Once the number of instances is equal to the specified number, you can instruct the application definition to run another task or end the application definition. This is useful when configuring an application whose login panel is similar to other windows within the application, or to easily control the number of attempts a user can have to access an application. Also see “Decrement” on page 79
Syntax examples	Increment ?RunCount

Example	Windows application definition
	
	Each time the application definition is run, a variable is incremented. This example counts the number of times the dialog box is displayed. If the dialog box is displayed more than three times, the application is closed. If the log on is successful, the count is reset.
	
	`# Logon Dialog Box`
	`Dialog`
	`  Class #32770`
	`  Title "Log on"`
	`EndDialog`
	
	`Increment ?RunCount`
	`If ?RunCount Gt "3"`
	`  MessageBox "Log on has been attempted too many times. The application will be closed."`
	`  KillApp "app.exe"`
	`Else`
	`  Type $Username #1001`
	`  Type $Password #1002`
	`  Click #1`
	`EndIf`
	
	`# Logon Successful Message`
	`Dialog`
	`  Ctrl #1`
	`  Title "Log on successful"`
	`EndDialog`
	
	`Set ?RunCount "0"`

KillApp

Use with	All
SecureLogin version	3.5 or later
Type	Action
Usage	KillApp **<Process-Name>** KillApp **<-Title>**
Arguments	**<Process-Name>** The name of the process to terminate. **-title “Application title”** The title of the process to terminate.
Description	Use the `KillApp` command to terminate an application.

Example 1 Windows application definition

Each time the application definition is run, a variable is incremented. This example counts the number of times the dialog box is displayed. If the dialog box is displayed more than three times, the application is closed. If the log on is successful, the count is reset.

```
# Logon Dialog Box
Dialog
   Title "Log on"
   Class #32770
EndDialog

Increment ?RunCount

If ?RunCount Gt "3"
   MessageBox "Log on has been attempted too many times.
The  application will be closed."
   KillApp "app.exe"

Else
   Type $Username #1001
   Type $Password #1002
   Click #1
EndIf

# Logon Successful Message
Dialog
   Title "Log on successful"
   Ctrl #1
EndDialog

Set ?RunCount "0"
```

Example 2 Windows application definition

Same application definition as used in Example 1, however, the KillApp process is specified by title.

```
Dialog
   Title "Login Simple"
   Class #32770
EndDialog

Increment ?RunCount

If ?RunCount Gt "3"
   MessageBox "Log on has been attempted too many times.
The application will be closed."
   KillApp -title "Login Simple"
Else
   Type $Username #1001
   Type $Password #1002
   Click #1
EndIf

# Logon Successful Message
Dialog
   Title "Login - Simple Successful"
   Ctrl #1
EndDialog

Set ?RunCount "0"
```

Local

Use with	All
SecureLogin version	3.5 or later
Type	Variable manipulator
Usage	Local **<?Variable>**
Arguments	**<?Variable>** The runtime variable to declare as local.
Description	Use the `Local` command to declare that a runtime variable will only exist for the lifetime of the application definition. Local runtime variables are used in the same way as normal runtime variables and are still written as ?Variable. Declare local runtime variables as local by using the `Local` command, followed by the variable name. When runtime variables are declared local, you cannot set them back again. You can declare a runtime variable local at any time in an application definition. Using local runtime variables increases the performance of SecureLogin, although only slightly. Local runtime variables are used to run application definitions multiple times without storing the runtime variables between each run of the application definition. Local runtime variables are also used to prevent runtime variables from overwriting each other, which could happen if two instances of an application definition are running at the same time. For example, use the `Local` command if two instances of Terminal Launcher are running, each instance running the same application definition but attached to different emulator sessions.

Example	Windows application definition
	
	This example declares a variable as local, and then uses it to count the number of times a dialog box is displayed. If the dialog box is displayed too many times, SecureLogin will alert the user, then close the application.
	
	`# Invalid Logon Message`
	`Dialog`
	`  Class #32770`
	`  Title "Logon Failure"`
	`EndDialog`
	
	`Local ?RunCount`
	`Increment ?RunCount`
	`If ?RunCount Gt "5"`
	`  MessageBox "Closing application"`
	`  KillApp "PasswordText.exe"`
	`EndIf`
	`Type $Username`
	`Type $Password`

MatchDomain

Use with	Advanced application definitions created using the Web Wizard.
SecureLogin version	3.5.x or later
Type	Action
Usage	MatchDomain "Domain"
Arguments	Domain The domain name or address to be matched.
Description	Use MatchDomain inside a Site block to filter a site based on its domain. If the domain doesn't match, the Site block fails to match. The domain matched is a normally a low level domain name such as www.yahoo.com and not www.yahoo.com/mymail/login

Example	In this example the site www.google.com is being matched by the application definition.
	
	`# === Logon Application Definition #2 ==`
	`# === Google Initial Logon ====`
	`#=====================================`
	`Site "Login" -userid "Google Log On" -initial`
	`  MatchDomain "www.google.com"`
	`  MatchField #1:1 -name "Email" -type "text"`
	`  MatchField #1:2 -name "Passwd" -type "password"`
	`  MatchField #1:3 -name "Cookie" -type "check"`
	`EndSite`
	
	`SetPrompt "Enter your user credentials"`
	`TextInput #1:1 -value "$Username"`
	`TextInput #1:2 -value "$Password"`
	`FocusInput#1:2 -focus "true"`
	`BooleanInput #1:3 -check "false"`
	`PressInput`
	`Endscript`

MatchElement

Use with	WinSSO, WebSSO, JavaSSO and .NetSSO workers.
SecureLogin version	8.7 or later 8.6 or later for WebSSO
Type	Action
Usage	`MatchElement #<ElementID> <Selector>`

Arguments	**#<ElementID>** The element ID is assigned to the matched control. **<Selector>** It is a combination of attributes that identifies an element of a form uniquely.The matching is based on this selector. WebSSO supports full CSS selector. It also supports the text matching. WinSSO, JavaSSO, and NetSSO are limited to the following. `<Type>#<id>.<class>:nth(<order>)[value=<value>][visible=<true|false>]`
Description	Use `MatchElement` to match the dynamic controls in a window.

Example	`Dialog`
	`Title "ACMsample"`
	`MatchElement #username EditBox`
	`EndDialog`
	
	In this example, `MatchElement` matches the type of the control.
	
	`Dialog`
	`Title "ACMsample"`
	`MatchElement #password #1000`
	`EndDialog`
	
	In this example, `MatchElement` matches the ID of the control.
	
	`Dialog`
	`Title "ACMsample"`
	`MatchElement #submit .Edit#101`
	`EndDialog`
	
	In this example, `MatchElement` matches the class and ID of the control.
	
	`Dialog`
	`Title "ACMsample"`
	`MatchElement #buttonvalue Button[value=ok]`
	`EndDialog`
	
	In this example, `MatchElement` matches the value of the control.

MatchField

Use with	Advanced application definitions created using the Web Wizard.
SecureLogin version	3.5.x or later
Type	Action
Usage	MatchField #FormID:FieldID [-optional] [-name "name"] [-type "type"] [-value "value"] [-defaultValue "defaultValue"] [-id “ID”]

Arguments	#FormID:FieldID The ID to be given to the matched option within the field, building from the #FormID of the associated form. The FormID and FieldID must be unsigned integers. The combined #FormID:FieldID must be unique within the site block. -optional Specifies that matching this field is not required to successfully match the parent form. -name "name" Match against the field name. -type "type" Match against the field type. Type can be one of the following: ◆ Button ◆ Checkbox ◆ Image ◆ Hidden ◆ Password ◆ Radio ◆ Reset ◆ Submit ◆ Text ◆ TextArea ◆ Select-multiple ◆ Select-one -value "value" Match against the field value. -defaultValue "defaultValue" Match against the field’s default value. -id “ID” Match against the field ID.
Description	Use MatchField to filter a form based on the presence of a particular field. If the field fails to match and it is not specified as optional, then the parent form will fail to match.

Example 1	This example would locate the Web page fields Email, Password and Cookie within the Web site www.google.com .com matched by the application definition.
	
	`# === Logon Application Definition #2 ==`
	`# === Google Initial Logon ====`
	`#=====================================`
	`Site Login -userid "Google Log On" -initial`
	`  MatchForm #1 -name "log on"`
	`  MatchDomain "www.google.com"`
	`  MatchField #1:1 -name "Email" -type "text"`
	`  MatchField #1:2 -name "Passwd" -type "password"`
	`  MatchField #1:3 -name "Cookie" -type "check"`
	`  MatchField #1:4 -name "SAVEOPTION" -type "checkbox" -value "YES"`
	`  MatchField #1:5 -name "Submit2" -type "submit"`
	`EndSite`
	
	`SetPrompt "Enter your user credentials"`
	`TextInput #1:1 -value "$Username"`
	`TextInput #1:2 -value "$Password"`
	`FocusInput#1:2 -focus "true"`
	`BooleanInput #1:3 -check "false"`
	`BooleanInput #1:4 -check "false"`
	`PressInput`
	`Endscript`

Example 2	In this example, the username and password fields are matched using field IDs. Also, the login-submit button is matched using field ID.
	
	`Site "Form" -initial`
	`  MatchForm #0 -name ""`
	`  MatchField #0:1 -name "" -id "username" -type "email"`
	`  MatchField #0:2 -name "" -id "password" -type "password"`
	`  MatchField #0:3 -name "" -id "login-submit" -type "submit"`
	`EndSite`

MatchForm

Use with	Advanced application definitions created using the Web Wizard.
SecureLogin version	3.5.x or later
Type	Action
Usage	MatchForm #FormID [-optional] [-name "name"] [-action "action"] [-method "method"] [-target "target"] MatchForm #FormID [-optional] [-noform]

Arguments	#FormID The ID to be given to a matching form. The ID must be an unsigned integer prefixed with # and unique within the site block. -optional Specifies that matching this form is not required to successfully match site. -noform Specifies the form id, which is required to define the MatchField command. This argument is used to match the elements that are not defined in the form tag. -name "name" Specifies the form name to match against. The form name is an optional value given to a form by the creator of the Web site. -action "action" Specifies the form action to match against. The URL to which the form content is sent for processing. -method "method" Specifies the form method to match against. The method or how to send the form data to the server. -target "target" Specifies the form target to match against. The window or frame at which to the form targets its contents.
Description	Use MatchForm to filter a site based on the presence of a particular form. If the form fails to match and it is not specified as optional, then the site will fail to match. Use MatchForm with -noform to match fields that use the elements that are not defined in the form tag. This argument is required to get a fom id for the MatchField command.

Example	◆ In this example the form named ‘log on’ within the Web site www.google.com .com is being matched by the application definition.
	
	`# === Logon Application Definition #2 ==`
	`# === Google Initial Logon ====`
	`#=====================================`
	`Site Login -userid "Google Log On" -initial`
	`  MatchForm #1 -name "log on"`
	`  MatchDomain "www.google.com"`
	`  MatchField #1:1 -name "Email" -type "text"`
	`  MatchField #1:2 -name "Passwd" -type "password"`
	`  MatchField #1:3 -name "Cookie" -type "check"`
	`EndSite`
	
	`SetPrompt "Enter your user credentials"`
	`TextInput #1:1 -value "$Username"`
	`TextInput #1:2 -value "$Password"`
	`FocusInput#1:2 -focus "true"`
	`BooleanInput #1:3 -check "false"`
	`PressInput`
	`Endscript`
	
	◆ The name, method, target and action of forms can match the null values by using an empty quote (“”). For example, the form name may be a “null”, the form can be matched without the name
	
	`MatchForm #1 -name ""`
	
	◆ The MatchField command requires a MatchForm id and if the input and other fields are without the form tag then use `-noform`
	
	`MatchForm #1 -noform`
	
	◆ When multiple forms are present, each form can be matched within a single site and is considered as single form.
	
	`Site "complex"`
	`  MatchForm #1 -noform`
	`  MatchForm #2 -name "user details"`
	`  MatchForm #3 -name "submission options"`
	`EndSite`

MatchOption

Use with	Advanced Web application definitions created using the Web Wizard, WinSSO, JavaSSO and .NetSSO workers. **IMPORTANT:** Use MatchOption with MatchElement when using for WinSSO, JavaSSO and .NetSSO workers.
SecureLogin version	3.5.x or later for Web. 8.7 for WinSSO, JavaSSO and .NetSSO workers.
Type	Action
Usage for Web	MatchOption #FormID:FieldID:OptionID [-optional] [-text "text"] [-value "value"]

Usage for WinSSO, JavaSSO and .NetSSO workers	MatchOption #ElementID [-index “index”] [-text "text"]
Arguments for Web	#FormID:FieldID:OptionID The ID to be given to the matched option within the field, building from the #FormID:FieldID of the associated selection field. The FormID, FieldID and OptionIDs must be unsigned integers. The combined #FormID:FieldID:OptionID must be unique within the site block. -optional Specifies that matching this option is not required to successfully match the parent field. -text "text" Specifies the text string for this particular option. **NOTE:** The text is what is displayed to the user. -value "value" Specifies the value for this particular option. **NOTE:** The value is what is passed to the server when a form is submitted.
Arguments for WinSSO, JavaSSO and .NetSSO workers	#ElementID The ID to be given to the matched option within the field, building from the #ElementD of the associated selection field. The element ID must be unique within the site block. -text "text" Specifies the text string for this particular option. **NOTE:** The text is what is displayed to the user. -index "index" Specifies the order for this particular option. **NOTE:** The index starts from 0.
Description	Use the `MatchOption` command to filter a field based on the presence of a particular option. An option is an item within a specific combo box or list box. If the option is not found, and it is not specified as optional, then the parent field will also fail to match.

Example for Web	In this example the form named ‘log on’ within the secure Web site www.lotto.com is being matched by the application definition.
	
	`# === Logon Application Definition #4 ==`
	`# === Lotto User Initial Logon ====`
	`#=====================================`
	`Site Login -userid "Member Log In" -initial`
	`  MatchForm #1 -name "log in"`
	`  MatchDomain "https://site10.Lotto.com"`
	`  MatchField #1:1 -name "Member ID" -type "text"`
	`  MatchField #1:2 -name "Passwd" -type "password"`
	`  MatchOption #1:3 -text "Secure" -value "Secure"`
	`EndSite`
	
	`SetPrompt "Enter your user credentials"`
	`TextInput #1:1 -value "$Username"`
	`TextInput #1:2 -value "$Password"`
	`FocusInput #1:3`
	`BooleanInput #1:3 -check "true"`
	`PressInput`
	`Endscript`

Example for WinSSO, JavaSSO and .NetSSO workers	In this example, the `MatchOption` command matches fourth element of the index.
	
	`Dialog`
	`MatchElement #1:5 #1034`
	`MatchOption #1:5:3 -index 4`
	`EndDialog`

MatchReferer

Use with	Advanced Web application definitions created using the Web Wizard.
SecureLogin version	3.5.x or later
Type	Action
Usage	MatchReferer "Referer"
Arguments	MatchReferer Used inside a site block, MatchReferer is used to filter a site based on a referer. If the site referer does not match, the site block fails to match. "Referer" The site referer which is to be matched. If PageA.htm includes a link to PageB.htm, then the referer is "PageA.htm".
Description	Use MatchReferer inside a Site/EndSite block to match or filter a site based on a referer.

Example	In this example the refering HTML page www.lotto.com/index.html is being matched by the application definition.
	
	`# === Logon Application Definition #5 ==`
	`# === Lotto User Initial Logon ====`
	`#=====================================`
	`Site "Login" -userid "Member Log On"?-initial`
	`MatchForm #1 -name "log on"?`
	`MatchReferer "www.lotto.com/index.html"?`
	`MatchDomain "https://site10.lotto.com"?`
	`MatchField #1:1 -name "Member ID"?-type "text"?`
	`MatchField #1:2 -name "Passwd"?-type "password"?`
	`MatchOption #1:3 -name "Secure"?-type "text"?`
	`EndSite`
	
	`SetPrompt "Enter your user credentials"?`
	`TextInput #1:1 -value "$Username"?`
	`TextInput #1:2 -value "$Password"?`
	`FocusInput #1:2 -focus "true"?`
	`BooleanInput #1:3 -check "true"?`
	`PressInput`
	`Endscript`

MatchRegex

Use with	All
SecureLogin version	7.0 or later
Type	Action
Usage	MatchRegex **<RegEx> <Input-String>**
Arguments	**<RegEx>** The regular expression **<Input-String>** The string to match against.
Description	Use the `MatchRegex` command to test whether a regular expression matches against the specified string or not. Can be used inside a Site–EndSite or Dialog–EndDialog block for example. For more information regarding regular expressions see the Boost C++ Libraries Web site (http://www.boost.org/doc/libs/1_33_1/libs/regex/doc/syntax_perl.html)

Example	This example matches against any Web page on the www.google.com domain that has a text box, a password box and text somewhere on the page that matches against the “Welcome \w+ to Gmail” regular expression (“Welcome Nick to Gmail” for example).
	
	`Site "Gmail: Email from Google"`
	`  MatchForm #1`
	`  MatchDomain "www.google.com"`
	`  MatchField #1:10 -type "text"`
	`  MatchField #1:11 -type "password"`
	`  GetText ?PageText`
	`  MatchRegex "Welcome \w+ to Gmail" ?PageText`
	`EndSite`
	`MessageBox "Matched"`

MatchTitle

Use with	Advanced Web application definitions created using the Web Wizard. **NOTE:** `-regex` parameter is not supported in SecureLogin versions prior to 7.0.
SecureLogin version	3.5 or later
Type	Action
Usage	MatchTitle [-regex] "Title"
Arguments	-regex Indicates that the Title argument is a regular expression. "Title" The site title which is to be matched. For more information regarding regular expressions see the Boost C++ Libraries Web site (http://www.boost.org/doc/libs/1_33_1/libs/regex/doc/syntax_perl.html)
Description	Use `MatchTitle` inside a `Site` block to match or filter a site based on a HTML page title.

Example	In this example the HTML page with the title 'The New York Times > Log In' within the Web site www.nytimes.com is matched by the application definition.
	
	`# ==========================================`
	`# Login Script #1 - The New York Times > Log In`
	`# ==========================================`
	`# === Initial Login ===`
	`Site "Login" -userid "nytimes.com #1" -initial`
	`MatchURL "http://www.nytimes.com/auth/login"`
	`MatchDomain "www.nytimes.com"`
	`MatchTitle "The New York Times > Log In"`
	`MatchForm #1 -name "login"`
	`MatchField #1:1 -name "USERID" -type "text"`
	`MatchField #1:2 -name "PASSWORD" -type "password"`
	`MatchField #1:3 -name "SAVEOPTION" -type "checkbox" -value "YES"`
	`MatchField #1:4 -name "Submit2" -type "submit"`
	`EndSite`
	
	If the title needing to be matched is "The New York Times > Log In", then you could use -regex and match against only a portion of the title.
	
	`MatchTitle -regex "Times > Log In"`

MatchURL

Use with	Advanced Web application definitions created using the Web Wizard. **NOTE:** `-regex` parameter is not supported in SecureLogin versions prior to 7.0.
SecureLogin version	3.5 or later
Type	Action
Usage	MatchURL [-regex] "URL"
Arguments	-regex You may also use regular expressions to match part of a URL, such as the domain only. "URL" The site URL which is to be matched. This need not be the URL listed in the navigation field of the Web browser as the given page may not have been loaded from there. For more information regarding regular expressions see the Boost C++ Libraries Web site. (http://www.boost.org/doc/libs/1_33_1/libs/regex/doc/syntax_perl.html) :
Description	Use MatchURL inside a Site block to match or filter a HTML page within a site based on its URL. The URL can be a complex Web address or a secure Web site.

Example	In this example the URL “https://www.nytimes.com/auth/login” is matched.
	
	`# === Initial Login ===`
	`Site "Login" -userid "nytimes.com #1" -initial`
	`MatchURL "https://www.nytimes.com/auth/login"`
	`MatchDomain "www.nytimes.com"`
	`MatchTitle "The New York Times > Log In"`
	`MatchForm #1 -name "login"`
	`MatchField #1:1 -name "USERID" -type "text"`
	`MatchField #1:2 -name "PASSWORD" -type "password"`
	`MatchField #1:3 -name "SAVEOPTION" -type "checkbox" -value "YES"`
	`MatchField #1:4 -name "Submit2" -type "submit"`
	`EndSite`
	
	If the URL to match is “http://www.nytimes.com/auth/login?URI=http://”, then you can match a portion of the URL with the -regex parameter.
	
	`MatchURL -regex "nytimes.com"`

MessageBox

Use with	Startup, Terminal Emulator, Web, or Windows
SecureLogin version	3.5 or later
Type	Action
Usage	MessageBox **<Data> [-Background] [-DefaultNo] [-YesNo <?Variable>] [-YesNoCancel <?Variable>]**

Arguments	**<-YesNo>** The -YesNo flag allows the user to select Yes or No within the message box, rather than being limited to an OK button only. **<-YesNoCancel>** The -YesNoCancel flag allows the user to select Yes, No, or Cancel when a message box is displayed. **<?Variable>** This runtime variable is required with the -YesNo / -YesNoCancel flag to store the result of the user action. **<-Background>** When specified, this parameter allows the user to open an application and work in that application, without having to respond to the MessageBox. If this parameter is not used, the MessageBox remains the active window. In Web applications, you must respond to the MessageBox before you can continue with any other work. **<-DefaultNo>** This optional parameter is used only with the -YesNoCancel flags. When the -DefaultNo parameter is set, the No button has the default focus rather than the Yes button. **<Data>** The text displayed to the user. <Data> can be several strings, variables, or a combination of both.
Description	Use the `MessageBox` command to display a dialog box that contains the text specified in the <Data> variable. The application definition is suspended until the user reacts to this message. The MessageBox can take any number of text arguments, including variables (for example, MessageBox "The user " $Username " has just logged onto the system"). You can set the -YesNo flag when calling a MessageBox. If the -YesNo flag is set, the MessageBox prompts the user with a box that has a Yes and a No button, rather than an OK button. Place a runtime <?Variable>, immediately after the flag, to capture the MessageBox result.

Syntax examples	`MessageBox "Application definition completed successfully"`
	
	`MessageBox "Do you wish to continue?"  -YesNo ?Result`
	
	`MessageBox "Do you wish to continue?" -YesNoCancel ?Result -Background -DefaultNo`

Example 1 — Windows application definition

This example detects the change password dialog box. A message box is displayed prompting the user whether or not they would like to change their password, and to inform them it was successful.

```
# Change Password Dialog Box
Dialog
   Class #32770
   Title "Change Password"
EndDialog

MessageBox -YesNo ?Result "Your password has expired,
would you like to change it now?"
If ?Result Eq "Yes"
   Type $Username #1015
   Type $Password #1004
   ChangePassword $Password Random
   Type $Password #1005
   Type $Password #1006
   Click #1
   MessageBox "Password changed successfully"
Else
   Click #2
   MessageBox "You chose not to change your password"
EndIf
```

Example 2 — Terminal Emulator test application definition

Use message boxes when troubleshooting application definitions. This example displays a message box before each step in the application definition to allow the writer to see where the application definition execution is failing.

```
MessageBox "Beginning wait for logon prompt"
WaitForText "ogin:"
MessageBox "Logon detected, now entering user name"
Type $Username
MessageBox "User name entered, now simulating Enter"
Type @E
MessageBox "Enter has been simulated, now waiting for
password"?
WaitForText "assword:"
MessageBox "Password detected, now entering password"
Type $Password
MessageBox "Password entered, now simulating Enter"
Type @E
MessageBox "Sequence completed, the user should now be
logged on"
```

Multiply

Use with	All
SecureLogin version	3.0 or later
Type	Variable manipulator
Usage	Multiply **<Variable1> <Variable2>** [?Result]
	NOTE: You must use integer arithmetic.

Arguments	**<Variable1>** The first argument is the number multiplied by the second argument. **<Variable2>** The second argument is the number by which the first number is multiplied. [?Result] Optional, the result of the equation.
Description	This fails to list the exception for RegSplitFailed. Use to multiply one number by another. You can write the numbers into the application definition or use variables. The results can be output to another variable or to one of the original numbers.
Syntax examples	Multiply "1" "2" ?Result Multiply ?LoginAttempts ?LoginFailures Multiply ?LoginAttempts ?LoginFailures ?Result Multiply ?LoginAttempts "3" Multiply ?LoginAttempts "3" ?Result

Example	Windows application definition
	
	This example reads the values of control IDs 103 and 104 into variables. From there they are multiplied, and typed into control ID 1.
	
	`ReadText #103 ?Number1`
	`ReadText #104 ?Number2`
	`Multiply ?Number1 ?Number2 ?Result`
	`Type ?Result #1`

OnException/ClearException

Use with	All
SecureLogin version	3.0.4 or later
Type	Flow control
Usage	OnException **<Exception Name>** Call **<SubRoutine>** ClearException **<Exception Name>**

Arguments	**<Exception Name>** The name of the exception on which you wish to act. The following exceptions are supported: ◆ **AAVerifyCancelled:** When a user cancels the re-authentication process (support will depend on the Advanced Authentication product being used). ◆ **AAVerifyFailed:** When the AAVerify re-authentication command fails. ◆ **ChangePasswordCancelled:** When a user cancels on the **Change Password** dialog. ◆ **EnterVariablesCancelled:** When a user cancels the automatic variable prompt box or the display variables prompt box. ◆ **GenerateOTPCancelled:** When a user cancels the GenerateOTP dialog. ◆ **GenerateOTPFailed:** When the `GenerateOTP` command fails. ◆ **PickListCancelled:** When a user cancels the pick list choice dialog. ◆ **RunFailed:** When the program specified by the `Run` command fails to launch. ◆ **SelectLoginCancelled:** When a user cancels the dialog box listing the login credential set. **<SubRoutine>** The name of the subroutine you want to run when the exception condition is true.
Description	Use the `OnException` command to detect when certain conditions are met. Currently, this is when Cancel is clicked on either of two dialog boxes. When the condition is met, a subroutine is run. Use the `ClearException` command to reset the exceptions value.
Syntax examples	`OnException ChangePasswordCancelled Call Display Error` `ClearException ChangePasswordCancelled`

Example 1 Windows application definition

In this example the login failed because the user has invalid credentials stored. This provides the user with an opportunity to verify their user name and password, but what happens if the user clicks Cancel? If the user clicks Cancel, the exception is executed and forces the user to enter their credentials.

```
# Logon Failed Dialog Box
Dialog
   Class #32770
   Title "Log on failed"
EndDialog
OnException EnterVariablesCancelled Call
VariablesCancelled
DisplayVariables "Please verify your user name and
password and try again.  IT x4532"
ClearException EnterVariablesCancelled
Type $Username #1001
Type $Password #1002
Click #1

Sub VariablesCancelled
   OnException EnterVariablesCancelled Call
VariablesCancelled
   DisplayVariables "You cannot cancel this verification
dialog box.  Please verify your user name and password
when prompted and click OK to try again."
   ClearException EnterVariablesCancelled
EndSub
```

Example 2 Windows application definition

This example prompts the user to change their password. The user is restricted from clicking cancel and is forced to enter a new password.

```
# Change Password Dialog Box
Dialog
   Class #32770
   Title "Change Password"?
EndDialog

Type $Username #1005
Type $Password #1006
OnException ChangePasswordCancelled Call ForceChangePwd
ChangePassword $Password "Please enter a new password for
the Human Resources application. IT x4532"?
Type $Password #1007
Type $Password #1008
ClearException ChangePasswordCancelled

Sub ForceChangePwd
   OnException ChangePasswordCancelled Call
ForceChangePwd
   ChangePassword $Password "You must enter a new
password and cannot Cancel. IT x4532"?
   ClearException ChangePasswordCancelled
EndSub
```

Example 3 Windows application definition

This example demonstrates the OnException usage of AAVerifyCancelled and AAVerifyFailed.

```
#
# Login - Simple
#
Dialog
   Title "Login - Simple"
   Class "#32770"
   Ctrl #1001
   Ctrl #1002
   Ctrl #1 "&Login"
   Ctrl #2 "Cancel"
   Ctrl #1027 "Username:"
   Ctrl #1028 "Password:"
   Ctrl #1009
EndDialog
   OnException AAVerifyCancelled Call
CancelSimpleLoginDialogCancelled
   OnException AAVerifyFailed Call
CancelSimpleLoginDialogFailed
   AAVerify -method "smartcard"
   Type $Username #1001
   Type $Password #1002
   Click #1
#
# Cancel the Simple Login Window - AAVerify cancelled
#
Sub CancelSimpleLoginDialogCancelled
   Click #2
   EndScript
EndSub
#
# Cancel the Simple Login Window - AAVerify failed
#
Sub CancelSimpleLoginDialogFailed
   Click #2
   MessageBox "Your re-authentication failed.  Login
canceled"
   EndScript
EndSub
```

Example 4 Windows application definition

This example demonstrates the OnException usage of GenerateOTPCancelled and GenerateOTPFailed.

```
#
# Login - Simple
#
Dialog
   Title "Login - Simple"
   Class "#32770"
   Ctrl #1001
   Ctrl #1002
   Ctrl #1 "&Login"
   Ctrl #2 "Cancel"
   Ctrl #1027 "Username:"
   Ctrl #1028 "Password:"
   Ctrl #1009
EndDialog
   OnException GenerateOTPCancelled Call
CancelSimpleLoginDialogCancelled
   OnException GenerateOTPFailed Call
CancelSimpleLoginDialogFailed
   GenerateOTP -mode "AISC-SKI" ?OtpResult
   Type $Username #1001
   Type ?OtpResult #1002
   Click #1
#
# Cancel the Simple Login Window - GenerateOTP cancelled
#
Sub CancelSimpleLoginDialogCancelled
   Click #2
   EndScript
EndSub
#
# Cancel the Simple Login Window - GenerateOTP failed
#
Sub CancelSimpleLoginDialogFailed
   Click #2
   MessageBox "Your generation of your password failed.
Login cancelled"
   EndScript
EndSub
```

Example 5 Windows application definition

This example demonstrates the OnException usage of `SelectLoginCancelled`. In the following example, two credential sets defined, one credential set is the default credentials created for the application and the other is a linked credential set. When the application is executed SecureLogin will prompt the user to select the credential set to use for this logon session. The following steps will link another credential set to an existing application definition.

1. In the notification area, right-click the SecureLogin  icon, then select **New Login**. The Add New Login Wizard Welcome page is displayed.
2. Select the application for which you want to add another login. In this example, Notepad.
3. Click **Next**.
4. In the **Description** field, specify a descriptive name for the login. For example, Talk.
5. Click **Finish**.
6. NSL prompts you to enter values for $Username and $Password for the newly created credential set. If you type in the name of an existing credential set, you are not prompted. But there is no selection box displayed to select an existing credential set. You must know and type the credential set name the same as you would for using the `setplat` command.
7. Start the application.

   The `<name of the application; in this example, notepad.exe>` login selection dialog box is displayed.
8. Select the required login credential set, then click **OK**.

   SecureLogin enters the credentials, and you are automatically logged on to the application.

```
## BeginSection: "Login Form"
Dialog
   Title "Untitled - Notepad"
EndDialog

OnException SelectLoginCancelled Call CannotCancel
SetPlat Login1
Type $username #1001
Type $password #1002

Sub CannotCancel
   Messagebox "You cannot cancel selecting to use a
credential set, closing application."
   #Send ALT F4 to close application
   type \ALT \|115
   EndScript
EndSub
```

Parent/EndParent

Use with Windows

SecureLogin version	3.5 or later
Type	Dialog specifier
Usage	Parent EndParent
Arguments	None
Description	Use the `EndParent` command to begin a parent block in which the statements act upon a window's parent. The commands that follow the `Parent` command function identically to commands used in a dialog block; if they equate to false, then the application definition ends. For example, the command Title in a parent block returns false if the title of the parent does not match the one specified in the command. However, if a command in a parent block returns a false result, the execution does not skip to the next parent block, as it would in a dialog block. Instead, the parent block proceeds to the next dialog block or the application definition terminates if no further dialog blocks exists. The `Parent` command is particularly useful in applications where the dialog box (for example, a login dialog box) is the child of an open window, typically in the background. If you are unable to single sign-on to an application after enabling it with the wizard, you typically need to specify parent blocks. You can also use the `Parent` command to execute commands on a dialog's parent. For example, it is possible to get an application definition to click a button on the parent window. An example is shown below. Use the `EndParent` command to terminate a parent block and set the subject of the application definition back to the original window. You can nest the `Parent` command, thereby allowing the parent block to act on the parent of the parent. **NOTE:** If you use the wizard or try to enable an application and it does not seem to work, try using the `Parent` command. It is able to handle windows that are within windows.

Example 1	Windows application definition
	
	This example specifies the dialog box that is used for log on. In this case, the parent of the login box has a class of "Centura:MDIFrame".
	
	`# Logon Dialog Box`
	`Dialog`
	`  Class "Centura:Dialog"`
	`  Ctrl #4098`
	`  Ctrl #4100`
	`  Title "Log on"`
	`  Parent`
	`  Class "Centura:MDIFrame"`
	`  EndParent`
	`EndDialog`
	
	`Type $Username #4098`
	`Type $Password #4100`
	`Click #4101`

Example 2	Windows application definition
	This example is used to click a button on the login window's parent.

```
# Logon Dialog Box
Dialog
   Class #32770
   Title "Log on"
EndDialog

Type $Username #1001
Type $Password #1002
Parent
   Click #1
EndParent
```

PickListAdd

Use with	All
SecureLogin version	3.5 or later (see note under Description below)
Type	Action
Usage	PickListAdd **<Display-Text> [<Return-Value>]**
Arguments	**<Display-Text>**
	The text displayed in the pick list for the specified option.
	<Return-Value>
	The value returned from the pick list. If not specified, the return value is the display text.
Description	Use the `PickListAdd` command to allow users with multiple accounts for a particular system to choose the account to which they will log on.
	You can also use `PickListAdd` command to choose from multiple sessions on one mainframe account. Use the `PickList` to build a list of databases, phone numbers, or any list from which a user can choose. You can then set variables or take action accordingly.
	`PickListAdd` is always used with the `PickListDisplay` and is typically also used in conjunction with the `SetPlat` command.
	NOTE: Change in usage from SecureLogin 6.1 on. Setting variables after adding them to the list no longer results in the new value appearing in the list. For example:

```
PickListAdd ?Y
Set ?Y "Text"
PickListDisplay ...
```

will display the value <not set>

Java or Windows application definition

In this example, the user has to pick which of three accounts to use. They pick which account they want to use, and SecureLogin switches to that set of credentials using the `SetPlat` command.

```
# Logon Dialog Box
Dialog
   Class #32770
   Title "Log on"
EndDialog

PickListAdd "Account One" "One"
PickListAdd "Account Two" "Two"
PickListAdd "Account Three" "Three"
PickListDisplay ?Account "Please select the account you
wish to use"-NoEdit
SetPlat ?Account
Type $Username #1001
Type $Password #1002
Click #1
## End Logon Dialog Box ##
```

Example 2 — Any application definition

In this example, the application should execute and when SecureLogin runs it should display the numbers 0 - 9.

```
Set ?Count "0"
Repeat 10
   PickListAdd ?Count
   Increment ?Count
EndRepeat
PickListDisplay ?Count "Please select your option " -
NoEdit
```

Example 3 — Java or Windows application definition

In this example, SecureLogin reads the possible values for the **Other** drop down box. It then prompts the user to select the desired item and types in the username, password, and selected item.

```
###Logon
PickListAdd #3
PickListDisplay ?Database "Select your database" -NoEdit
SetPlat ?Database
Type #1 $Username
Type #2 $Password
Select ?Database #3
###End logon##
```

PickListDisplay

Use with	All
SecureLogin version	3.5 or later
Type	Action
Usage	PickListDisplay **<?Variable> <Display-Text> [-NoEdit]**

Arguments	**<?Variable>** The output variable for the selected option. **<Display-Text**> The description text for the pick list box. **-NoEdi**t The -NoEdit flag disables the addition of extra variables by the user.
Description	Use the `PickListDisplay` command to display the pick list entries built by previous calls to `PickListAdd`. The `PickListDisplay` command returns the result in a <?Variable> sent to the command. If the desired entry is not among the displayed entries, the user can enter their own data into an edit field at the bottom of the pick list. Set the -NoEdit flag to turn this feature off.
Syntax examples	`PickListDisplay ?Choice "Please select the account you wish to use"` `PickListDisplay ?Choice "Please select the account you wish to use" -NoEdit`

Example	Windows example
	
	In this example, the user has three accounts to this application and wants to pick which one to use. They pick which account they want to use and SecureLogin uses the `SetPlat` command to switch to that set of credentials.
	
	`# Logon dialog box`
	`Dialog`
	`  Class #32770`
	`  Title "Log on"`
	`EndDialog`
	
	`PickListAdd "Account one" "One"`
	`PickListAdd "Account two" "Two"`
	`PickListAdd "Account three" "Three"`
	`PickListDisplay ?Account "Please select the account you wish to use" -NoEdit`
	`SetPlat ?Account`
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

PositionCharacter

Use with	Password Policy application definitions
SecureLogin version	3.5 or later
Type	Action
Usage	POSITIONCHARACTER [NUMERAL] [UPPERCASE] [LOWERCASE] [PUNCTUATION] <Position>, [<Position>].

Arguments	**[NUMERAL]** The character at <Position> must be a numeral. [**UPPERCASE]** The character at <Position> must be an uppercase character. **[LOWERCASE]** The character at <Position> must be a lowercase character. **[PUNCTUATION]** The character at <Position> must be a punctuation character. **<Position>** The character position in the password.
Description	Use this command in a password policy application definition to enforce that a certain character in the password is a numeral, uppercase, lowercase, or a punctuation character. You can specify multiple positions.
Example	The password is not valid unless the first, sixth, and seventh characters are uppercase. `POSITIONCHARACTER UPPERCASE 1,6,7`

PressInput

Use with	Advanced Web application definitions created using the Web Wizard.
SecureLogin Version	3.5.x or later
Type	Action
Usage	PressInput [#FormID:FieldID]
Arguments	#FormID:FieldID The ID that was given to the matched field in the Site block using `MatchField` command. The FormID and FieldID must be unsigned integers.
Description	Simulates a keyboard enter event focusing a given field beforehand.

Example	This example the `PressInput` command within the application definition is the equivalent of clicking the Sign In button on the www.google.com Web site.
	
	`# === Logon Application Definition #2 ==`
	`# === Google Initial Logon ====`
	`#=====================================`
	`Site Login -userid "Google Log On" -initial`
	`  MatchForm #1 -name "log on"`
	`  MatchDomain "www.google.com"`
	`  MatchField #1:1 -name "Email" -type "text"`
	`  MatchField #1:2 -name "Passwd" -type "password"`
	`  MatchField #1:3 -name "Cookie" -type "check"`
	`  MatchField #1:4 -name "Submit" -type "submit"`
	`EndSite`
	
	`SetPrompt "Enter your user credentials"`
	`TextInput #1:1 -value "$Username"`
	`TextInput #1:2 -value "$Password"`
	`FocusInput #1:3`
	`BooleanInput #1:3 -check "false"`
	`FocusInput #1:4`
	`PressInput #1:4`
	`Endscript`

ReadInput

Use with	Web application, WinSSO, JavaSSO and .NetSSO workers
SecureLogin version	8.5 or later
Type	Action
Web application usage	ReadInput **#<form id>:<field id> -checked ? <writable variable name>**
Arguments	**<#Ctrl-ID>** The control ID number of the input to read. **<?Variable>** The variable that receives the text that is read. It can read input given in check box, radio button, submit button and text field.
Description	In a web application, the `ReadInput` command reads the text from any given `<#Ctrl-ID>` (check box, radio button, submit button and text field). For this command to function correctly, the `<#Ctrl-ID>` must be valid.
Syntax examples	`ReadInput #1:3 -checked ?check`

Example 1	Web application
	
	`Site "test"`
	`MatchForm #1 -name ""`
	`MatchField #1:3 –type "text"`
	`EndSite`
	
	`ReadInput #1:3 -checked ?fieldValue`

ReadText

Use with	Terminal Emulator, Windows. **NOTE:** For terminal emulator application definitions, this command is only supported when used with `tlaunch.exe` emulator definitions with the type "HLLAPI". This includes HLLAPI, WinHLLAPI, and HLLAPI16 definitions. This command will not function with other emulator definition types.
SecureLogin version	3.5 or later
Type	Action
Windows Usage	ReadText **<#Ctrl-ID> <?Variable>** ReadText [-order] **<#Order-ID>**
Terminal Launcher Usage	ReadText **<?Variable> <Character-Number> <Row-Number> <Column-Number>**
Arguments	**<#Ctrl-ID>** The control ID number of the text to read. [-order] If the control ID's are not constant, utilize the -order argument to instruct SecureLogin to type into a control based on the creation order and not the tab order. For more information on the -order argument usage, see “Example 4” on page 179. <#Order-ID> For Windows application definitions, this parameter specifies which control based on the creation order in which to type the text. **<?Variable>** The variable that receives the text that is read. **<Character-Number>** The number of characters to read. **<Row-Number>** The horizontal position number of the first character to read (for example, row). **<Column-Number>** The vertical position number of the first character to read (for example, column).

Description	The `ReadText` command can be used in both Windows and Terminal Emulator application definitions. While the usage and arguments for the use of ReadText with Windows and Terminal Launcher are different, the results of each command are the same. **Windows application definition** In a Windows application definition, the `ReadText` command reads the text from any given <#Ctrl-ID>, and sends it to the specified variable. For this command to function correctly, the <#Ctrl-ID> must be valid. **Terminal Launcher application definition** In a Terminal Launcher application definition, the `ReadText` command reads a specified number of characters, starting at the <Row-Number>, and sends those characters to the specified <Variable>. The `ReadText` command will not work with Generic or Advanced Generic emulators, it only works with HLLAPI and some DDE emulators. For Generic or Advanced Generic emulators, use the If -Text or Gettext commands. For more information, see “If/Else/EndIf” on page 99 and “GetText” on page 96.
Syntax examples	`ReadText #301 ?Text` `ReadText ?Text 10 4 6`

Example 1	Terminal Emulator application definition
	
	`# Read 10 characters starting at row 4 column 6.`
	`ReadText ?result 10 4 6`

Example 2	Windows script `ReadText #1004 ?result`

Example 3 Windows application definition

The same title and class appear in the error message dialog box when a user fails to log on.

This example distinguishes between errors and provides users with more specific information, rather than a general message stating their username and password is incorrect, or the account is locked. In this case, the example reads the error message, clicks OK, and prompts the user with a customized message.

```
# Logon Failed Message
Dialog
   Class #32770
   Title "Log on failed"
EndDialog

ReadText #65535 ?ErrorMsg
Click #1
If "Invalid Username" -In ?ErrorMsg
   DisplayVariables "Please verify your Username and try
again." $Username
   Type $Username #1001
   Type $Password #1002
   Click #1
EndIf
If "Invalid Password" -In ?ErrorMsg
   DisplayVariables "Please verify your Password and try
again." $Password
   Type $Username #1001
   Type $Password #1002
   Click #1
EndIf
If "Account locked" -In ?ErrorMsg
   MessageBox "Your account is locked. Please contact the
IT help- desk on x4532."
   EndScript
EndIf
```

Example 4 Windows application definition

This example reads the text from a control ID and sets the database variable so the user is not prompted to set the variable.

```
# Logon Dialog Box
Dialog
   Class #32770
   Title "Log on"
EndDialog

ReadText #15 ?Database
If -Exists $Database
Else
   Set $Database ?Database
EndIf
Type $Username #1001
Type $Password #1002
Type $Database #1003
Click #1
```

Example 5	Terminal Emulator application definition
	
	This example reads a message in a terminal emulator and displays a message in a user friendly format.
	
	`ReadText ?Message 30 24 2`
	`MessageBox ?Message`

Example 6	Windows application definition
	
	This example reads the text from a control defined by its creation order and sets the database variable so the user is not prompted to set the variable.
	
	`# Logon Dialog Box`
	`Dialog`
	`  Class #32770`
	`  Title "Log on"`
	`EndDialog`
	
	`ReadText -order #5 ?Database`
	`If -Exists $Database`
	`Else`
	`  Set $Database ?Database`
	`EndIf`
	`Type $Username #1001`
	`Type $Password #1002`
	`Type $Database #1003`
	`Click #1`

RegSplit

Use with	All
SecureLogin version	3.5 or later
Type	Action
Usage	RegSplit **<RegEx> <Input-String> [<Output-String1> [<Output-String2>]...]**
Arguments	**<RegEx>** The regular expression. **<Input-String>** The string that to split. **<Output-String1>** The first sub-expression. **<Output-String2>** The second sub-expression.

Description	Use the `RegSplit` command to split a string using a regular expression. <Output-String1> and <Output-String2> contain the first and second sub-expressions respectively. When using regular expressions with the `RegSplit` command, ensure that any regular expressions comply with the syntax rules detailed under “Regular Expressions” on page 47. For more information regarding regular expressions see: www.boost.org/doc/libs/1_33_1/libs/regex/doc/syntax_perl.html (http://www.boost.org/doc/libs/1_33_1/libs/regex/doc/syntax_perl.html) For information regarding Microsoft regular expression usage, search the Microsoft MSDN Library at: http://msdn2.microsoft.com/en-us/library/default.aspx (http://msdn2.microsoft.com/en-us/library/default.aspx)

Example 1	Windows application definition
	
	This example copies text from control ID 301 to the ?Text variable. The `RegSplit` command is then used to strip the user name details out of the text that was read. The platform is set to that user name, and the correct password is entered by SecureLogin.
	
	`# Logon Dialog Box`
	`Dialog`
	`  Class #32770`
	`  Title "Log on"`
	`EndDialog`
	
	`ReadText #65535 ?Text`
	`RegSplit "Please enter the password for (.*) account"`
	`?Text ?UserSetPlat ?User`
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

Example 2 — How to handle regsplit exception with OnException

```
# Logon Dialog Box
Dialog
   Title "Untitled - Notepad"
EndDialog

Set ?Url "Oneabc"
Type ?Url
Type \n
# Create exception handler
OnException RegSplitFailed Call RegSplitError
# Provide suspicious regular expression, note the "\)"
below
RegSplit "(.*\)abc(.*)" ?Url ?Lhs ?Rhs
StrCat ?Url ?Lhs ", " ?Rhs
MessageBox ?Url
ClearException RegSplitFailed

Sub RegSplitError
   # print out RegSplitReturnCode
   Type "RegSplitError: "
   Type ?RegSplitReturnCode
   Type \n
   EndScript
EndSub
```

Example 3 — Open text example

```
Set ?InputString "This is a long string with a few
components in it"
RegSplit "This(.*)a long(.*)with(.*)components(.*)"
?InputString ?First ?Second ?Third ?Fourth
Type "First value is " ?First
Type \n
Type "Second value is " ?Second
Type \n
Type "Third value is " ?Third
Type \n
Type "Fourth value is " ?Fourth
#?First = "is", ?Second = "string", ?Third = "a few",
?Fourth = "in it"
```

ReLoadPlat

Use with	Startup, Terminal Emulator, Web, or Windows
SecureLogin version	3.5 or later
Type	Action
Usage	ReloadPlat
Arguments	None

Description	When an application first presents a login screen, SecureLogin displays a message prompting the user to select an appropriate platform from a list. Once selected, SecureLogin enters the chosen platform's credentials into the application and submits them. If log on fails due to incorrect credentials, SecureLogin prompts the user to change their credentials. SecureLogin does not retain the platform details and prompts the user to re-enter the information. This could result in the user changing the wrong credentials if they select the incorrect platform. The `SetPlat`, `ReLoadPlat` and `ClearPlat` commands resolve this issue. `ReloadPlat` sets the current platform to the one which was last chosen (for the given application) or, if a platform was not previously selected, the command will leave it unset. Use the `ReLoadPlat` command at: ◆ Log on. Before the user first logs onto the application, call `ReLoadPlat`. This prevents the user from having to reselect a platform after a failed log on. ◆ Failed log on. Call `ReLoadPlat` to reselect the platform that contained the incorrect credentials. This gives the user an opportunity to change the credentials using a `ChangePassword` or a `DisplayVariables` command. See also “SetPlat” on page 155 and “ClearPlat” on page 70.

Example Windows application definition

```
# ==== BeginSection: Application startup ====
Dialog
   Class "#32770"
   Title "Password Test Application"
EndDialog

ClearPlat
# ==== EndSection: Application startup ====

# ==== BeginSection: Log on ====
Dialog
   Class "#32770"
   Title "Log on"
   Ctrl #1001
EndDialog

ReLoadPlat
SetPrompt "Username =====>"
Type $Username #1001
SetPrompt "Password =====>"
Type $Password #1002
SetPrompt "Domain =====>"
Type $Domain #1003
Click #1
# ==== EndSection: Log on ====

## ==== BeginSection: Log on successful ====
Dialog
   Class "#32770
   "Title "Log on successful"
EndDialog

ClearPlat
Click #2
# ==== EndSection: Log on successful ====

# ==== BeginSection: Log on failure ====
Dialog
   Class "#32770"
   Title "Log on failure"
EndDialog

Click #2
ReLoadPlatOnException ChangePasswordCancelled Call
Change-Cancelled
ChangePassword $password
ClearException ChangePasswordCancelled
Type -raw \Alt F
Type -raw L
# ==== EndSection: Log on failure ====

# ==== BeginSection: Change credentials cancelled ====
Sub ChangeCancelled
   ClearPlat
   EndScript
EndSub
# ==== EndSection: Change credentials Cancelled ===
```

Repeat/EndRepeat

Use with	All
SecureLogin version	3.5 or later
Type	Action
Usage	Repeat **[Loop#]** EndRepeat
Arguments	**[Loop#]** The number of times the repeat application definition block is repeated. If not specified, the repeat continues indefinitely unless broken by other commands.
Description	Use the `Repeat` command to establish an application definition block similar to the `If` command. The repeat block is terminated by an `EndRepeat` command. Alternatively, you can use the `Break` or `EndScript` commands to break out of the loop.
Syntax examples	`Repeat` `Repeat 3`

Example	Terminal application definition

This example uses the `Repeat` command to watch the screen for the messages and responds accordingly. You can use the `Break` command to jump to the next command following the EndRepeat.

```
# Initial System Log on
WaitForText "login:"
Type $Username
Type @E
WaitForText "password:"
Type $Password
Type @E
Delay 500
#Repeat loop for error handling
Repeat
   #Check to see if password has expired
   If -Text "EMS: The password has expired."
      ChangePassword
      #Password
      Type $Password
      Type @E
      Type $Password
      Type @E
   EndIf
   #User has an invalid Username and / or # Password
stored.
   If -Text "Log on Failed"
      DisplayVariables "The username and / or password
stored by SecureLogin is invalid. Please verify your
credentials and try again.  IT x453."
      Type $Username
      Type @E
      Delay 500
      WaitForText "password:"
      Type $Password
      Type @E
      Delay 500
   EndIf
   # Account is locked for some reason, possibly
inactive.
   If -Text "Account Locked"
      MessageBox "Your account has been locked, possibly
due to inactivity for 40 days. Please contact the
administrator on x453."
   EndIf
   # Main Menu, user has logged on successfully.
   If -Text "Application Selection"
      Break
   EndIf
   Delay 100
EndRepeatDelay 100
EndRepeat
```

RestrictVariable

Use with	All
SecureLogin version	3.5 or later
Type	Action

Usage	RestrictVariable **<Variable-Name> <Password-Policy>**
Arguments	**<Variable-Name>** The name of the variable to restrict. **<Password-Policy>** The name of the policy to enforce on the variable.
Description	Use the `RestrictVariable` command to monitor a variable and enforce a specified password policy on the variable. Use the `RestrictVariable` command to monitor a variable and enforce a specified password policy. There are two instances when the password policy is enforced. 1. On application startup if credential data is not defined 2. On ChangePassword command On application startup SecureLogin will prompt the user for credential information if the values do not exist. If an empty credential is restricted with the `RestrictVariable` command, SecureLogon will require the user to provide a valid entry before the script continues. Users could cancel out of the prompt for new credentials. Hence, it is a normal practice to monitor this activity with the `OnException` command. When the `ChangePassword` command is used, the user is forced to enter a password that complies to the selected password policy set with the `RestrictVariable` command. ChangePassword can also be canceled by the user and should be monitored with the `OnException` command. When restricting variables to policies, if you are making a tighter policy than is already in place, and you restrict a variable that does not match the policy today, then the user cannot save it the first time. This is because when SecureLogin detects there is no saved credential, a user who has a password of 6 characters today cannot save it if the policy restricts the $Password variable to 8 characters and 2 numbers. “Example 2” on page 145 works around this by restricting a new password variable (?NewPwd), instead of restricting the $Password variable. The user can store their existing password when SecureLogin prompts for the credentials first time, and enforces the stronger password policy when the password expires in x days. You can restrict any variable using a password policy, not just a $Password. You can also use RestrictVariable to make sure other variables are entered in the correct format. For example, you can enforce that $Username is always lowercase or $Database is 6 characters and no numbers.

Example 1 Windows application definition

This example uses the application definition to restrict the $Password variable to the Finance password policy. The user's password must match the policy when they first save their credentials. When the password requires changing, the application definition generates a new password randomly based on that policy (no user intervention is required).

```
# Set the password to use the Finance password policy
RestrictVariable $Password FinancePwdPolicy

#Logon Dialog Box
Dialog
   Class #32770
   Title "Log on"
EndDialog

Type $Username #1001
Type $Password #1002

#Change Password Dialog Box
Dialog
   Class #32770
   Title "Change Password"?
EndDialog

Type $Username #1015
Type $Password #1004
ChangePassword $Password Random
Type $Password #1005
Type $Password #1006
Click #1
```

Example 2	Windows application definition

This example uses the application definition to restrict the ?NewPwd variable to the Finance password policy. When the application starts for the first time and prompts the user to enter their credentials, then their current password ($Password) is saved and used.

When the password expires, the password policy is enforced on any new password. This is a way to enforce tougher password policies (than are currently in place) when you cannot guarantee all existing passwords meet the new policy.

```
# Set the password to use the Finance password policy
RestrictVariable ?NewPwd FinancePwdPolicy

# Logon Dialog Box
Dialog
   Class #32770
   Title "Log on"
EndDialog

Type $Username #1001
Type $Password #1002
Click #1

# Change Password Dialog Box
Dialog
   Class #32770
   Title "Change Password"
EndDialog

Type $Username #1015
Type $Password #1004
ChangePassword "Please enter a new password." ?NewPwd
Type ?NewPwd #1005
Type ?NewPwd #1006
Set $Password ?NewPwd
Click #1
```

Run

Use with	Startup, Terminal Launcher, Web, or Windows
SecureLogin version	3.5 or later
Type	Action
Usage	Run **<Command> [<Arg1> [<Arg2>] ...]**
Arguments	**<Command>**
	The full path of the program to execute.
	<Arg1> <Arg2>
	An optional list of arguments and switches for the command.

Description	Use the Run command to launch the program specified in <Command> with the specified optional [<Arg1> [<Arg2>] …] arguments. The application definition does not wait for the launched program to complete.

Example	Startup application definition
	
	This example prompts the user to start the Finance System.
	
	If they click:
	
	◆ **Yes**, the Run command is used to start the application with the necessary switches.
	◆ **No**, a message box is displayed, and the application is not started.
	
	`MessageBox "Would you like to connect to the Finance System?" -YesNo ?Result`
	`If ?Result Eq "Yes"`
	`  Run "C:\Program Files\HRS\Finance.exe"  "/DB:HRS" "/Debug"`
	`Else`
	`  MessageBox "You have chosen not to run the Finance System.  Please do so manually."`
	`  EndScript`
	`EndIf`

RunEX

Use with	Startup, Terminal Launcher, Web, or Windows
SecureLogin version	7.0.3 or later
Type	Action
Usage	RunEx [-show <state>] [-position <coord>] [-size <dimensions>] -cmd "executablepath" [<Arg1>... <ArgN>]

Arguments	**-show <state>** `state` is a variable or value that represents a window state. **NOTE:** If the state cannot be evaluated to a valid state, then the default value is `NORMAL`. **-position <coord>** `coord` is a variable or value that represents screen position in pixels from the top left of the window. **-size <dimensions>** `dimensions` is a variable or value that represents width in pixels. **<Arg1>....<ArgN>** An optional list of arguments and switches for the command **-cmd <command>** Command is the full path of the program to execute. Note that the full path is only necessary if the application cannot be located in the systems path environment variable.
Description	RunEX executes a function in the hidden mode based on the options the user provides.

Example	For Example:
	
	`# Run cmd.exe maximized`
	`RunEx -show maximize -cmd cmd.exe`
	`# Run cmd.exe minimized`
	`RunEx -show 6 -cmd cmd.exe`
	`# Run cmd.exe hidden using a variable`
	`Set ?show hide`
	`RunEx -show ?show -cmd cmd.exe`
	`# Run cmd.exe at position 50, 50`
	`RunEx -position 50,50 -cmd cmd.exe`
	`RunEx -pos 50,50 -cmd cmd.exe`
	`# Run cmd.exe at position 50, 50 using a variable`
	`Set ?pos 50,50`
	`RunEx -pos ?pos -cmd cmd.exe`
	`# Run cmd.exe in a window sized to 800,900`
	`RunEx -size 800,900 -cmd cmd.exe`
	`# Run cmd.exe in a window sized to 400,500 using a variable`
	`Set ?size 400,500`
	`RunEx -size ?size -cmd cmd.exe`

Select

Use with	Java, Advanced Web, Windows
SecureLogin version	6.1 or later
Type	Action
Usage	Select **<Text of Item to select> [<#Ctrl-ID>]**

Arguments	**<Text of Item to select>** The text item that you want SecureLogin to select in the list box. **<#Ctrl-ID>** When multiple list boxes are found, this specifies which list box to address.
Description	Use the `Select` command to select entries from a combo or list style control.

Examples	This example picks an item from the session combo or list control:
	
	`Select ?session #1`
	
	This example selects a tab within another tab control. When one tab control is contained within another, the tab selection order is irrelevant.
	
	`Select "Quick Connect" #70`
	`Select "Connection" #69`
	
	This example selects a cell from within a table
	
	`Select "[0,0]" #1`
	`If –text "User" #1`
	`  Select "[0,1]" #1`
	`  Type $Username #1`
	`Endif`

SelectListBoxItem

Use with	Advanced Web application definitions
SecureLogin version	3.5 or later
Type	Action
Usage	SelectListBoxItem **<Item text of selection> <#Ctrl-ID> [-multiselect]**
Arguments	**<Item text of selection>** The text item that you want SecureLogin to select in the list box. it can be a variable or a string. **<#Ctrl-ID>** This argument is required and represents the control ID of the list box. **<-multiselect>** Used to select multiple list box entries by using a subsequent `SelectListBoxItem` command.
Description	Use the `SelectListBoxItem` command to select entries from a list box. For instructions on determining <#Ctrl-IDs>, see “DumpPage” on page 85.

Example	`If "ERROR: The credentials supplied were invalid. Please try again." -In ?Text`
	`  SelectListBoxItem "Find Context" #1`
	`  Type ?SysUser #1`
	`  Type $Password #2`
	`  MessageBox "If logon continues to fail, please logout of the computer and back in, retry, and report it to your SecureLogin administrator."`
	`  EndScript`
	`EndIf`

SelectOption

Use with	Advanced Web application definitions, WinSSO, JavaSSO and .NetSSO workers.
SecureLogin version	3.5.x or later
Type	Action
Usage	**SelectOption #FormID:FieldID:OptionID -select <true|false>** or **SelectOption #FormID:FieldID –clear**
Arguments	#FormID:FieldID:OptionID The ID that was given to the matched option in the Site block using the `MatchOption` command. The FormID, FieldID, and OptionID must be unsigned integers. -select "select" Selects or deselects a specific option. "select" is a Boolean value, either "true" or "false". -clear Deselects all options for the given control.
Description	Use the `SelectOption` command to select or deselect options within a list box or combo dialog box.

Example	This example clears the selection in the option list and selects option 2 only.
	`SelectOption #1:3 -clear`
	`SelectOption #1:3:2 –select true`

SendEvent

Use with	All
SecureLogin Version	7.0

Type	Action
Usage	SendEvent <Windows Handle> <Event Specifier>
Arguments	<Windows Handle> A valid windows handle. This should be a local variable with the handle initialised via a call to `GetHandle`. Alternatively, it is possible to broadcast the event by using the Windows constant HWND_BROADCAST. <Event Specifier> See “Event/Event Specifiers” on page 86 for the applicable conditions. In addition, a new custom single sign-on event can be used, SSO_NOTIFY.
Description	Use the `SendEvent` command with constants: ◆ HWND_BROADCAST to send an event to all windows ◆ SSO_NOTIFY to send a custom single sign-on event

Example 1	Send WM_SETFOCUS using a captured handle
	
	`Event WM_SETFOCUS`
	`....`
	`GetHandle ?handle`
	`SendEvent ?handle WM_SETFOCUS`

Example 2	Broadcast the custom SSO_NOTIFY event
	
	`Event SSO_NOTIFY`
	`....`
	`SendEvent HWND_BROADCAST SSO_NOTIFY`

SendKey

Use with	Terminal Emulator
SecureLogin Version	3.5 or later
Type	Action
Usage	SendKey **<Text>**
Arguments	**<Text>** The text typed into the emulator screen.

Description	Use the SendKey command to work only with Generic and Advanced Generic emulators. You can use the SendKey command in the same manner as the Type command. Generally, the Type command is the preferred command to use. The Type command places the text into the clipboard, and then pastes it into the emulator screen. The SendKey command enters the text directly into the emulator screen. **Using the Type Command:** Variables do not work with the SendKey command. If you want to use variables, use the Type command. The Type command has many special functions, and some you can use with the SendKey command. For more information, see "Type" on page 175 and Chapter 8, "Reference Commands and Keys," on page 193.

Example	Terminal Emulator application definition
	
	The example sends the username and password to the terminal emulator.
	
	`#Send User Name`
	`SendKey "DJones"`
	`SendKey "\N"`
	`#Send Password`
	`SendKey "Hu7%f"`
	`SendKey "\N"`

Set

Use with	All
SecureLogin version	3.5 or later
Type	Action
Usage	Set <Variable> <Data>
Arguments	**<Variable>** The variable to which the data is being assigned. **<Data>** The text or variable to read from and assign to the specified variable, for example: `Set ?Message "\?Username"`
Descriptions	Use the Set command to copy the value of <Data> into <Variable>. The <Data> can be any text or another variable, whereas the <Variable> must be either a ?Variable or $Variable.

Example 1 Windows application definition

This example uses the application definition to set a ?RunCount variable to count the number of times the application is run.

```
# Logon Dialog Box
Dialog
   Class #32770
   Title "Log on"
EndDialog

If ?RunCount Eq <NOTSET>
   Set ?RunCount "1"
Else
   Increment ?RunCount
EndIf

Type $Username #1001
Type $Password #1002
Click #1
```

Example 2 Windows application definition

This example uses the application definition to set the ?NewPwd to the stored $Password variable.

```
# Change Password Dialog Box
Dialog
   Class #32770
   Title "Change Password"
EndDialog

Type $Username #1015
Type $Password #1004
ChangePassword ?NewPwd Random
Type ?NewPwd #1005
Type ?NewPwd #1006
Set $Password ?NewPwd
Click #1
```

Example 3 Windows application definition

This example uses the application definition to read the value of control ID 15 and sets the $Database variable so the user does not have to set the variable.

```
# Logon Dialog Box
Dialog
   Class #32770
   Title "Log on"
EndDialog

ReadText #15 ?Database
If -Exists $Database
Else
   Set $Database ?Database
EndIf
```

SetCheckBox

Use with Advanced Web, Windows

SecureLogin version	3.5 or later
Type	Action
Usage	SetCheckBox **<Item Number> <Option>**
Arguments	**<Item Number>** The check box in reference to the number of check boxes found. **<Option>** Specifies the status of the check box as Checked or Unchecked.
Description	Use the `SetCheckBox` command to select or clear a check box.

Example	`MessageBox "Scroll down to see the 'Search Language' section with the check boxes then click OK"setcheckbox #1 "checked"`
	`setcheckbox #2 "checked"`
	`setcheckbox #3 "checked"`
	`setcheckbox #4 "checked"`
	`setcheckbox #25 "checked"`
	`setcheckbox #26 "checked"`
	`setcheckbox #27 "checked"`
	`MessageBox "Did it select the first four languages and Norwegian, Polish and Portuguese languages" -yesno ?advweb`
	`if ?advweb eq yes`
	`  Set ?cmd37 "SetCheckBox command worked"`
	`Else`
	`  Set ?cmd37 "SetCheckBox failed"`
	`Endif`
	`SetCheckbox #1 "unchecked"`
	`SetCheckbox #2 "unchecked"`
	`SetCheckbox #3 "unchecked"`
	`SetCheckbox #4 "unchecked"`
	`SetCheckbox #26 "unchecked"`
	`SetCheckbox #27 "unchecked"`
	`MessageBox "Did it clear all languages except Norwegian" -yesno ? advweb2`
	`If ?advweb2 eq yes`
	`   set ?cmd38 "SetCheckBox command worked"`
	`Else`
	`   set ?cmd38 "SetCheckBox failed"`
	`Endif`

SetCursor

Use with	Terminal Emulator application definition (only works with HLLAPI and some DDE Tlaunch emulator definitions)
SecureLogin version	3.5 or later
Type	Action
Usage 1	SetCursor **<Screen-Position>**
Usage 2	SetCursor **<X Coordinate> <Y Coordinate>**

Arguments	**<Screen-Position>** The position on the screen to move the cursor. **<X Coordinate>** The horizontal coordinate. When specified, a row or column conversion is carried out before the cursor is set to the position. **<Y Coordinate>** The vertical coordinates. When specified, a row or column conversion is carried out before the cursor is set to the position.
Description	Use the `SetCursor` command to set the cursor to a specified <Screen-Position> or <X Coordinate> <Y Coordinate>. The position is noted by a number greater than 0 (zero), for example, SetCursor 200. Terminal Launcher displays an error message if the screen position is invalid.

Syntax examples	`SetCursor 200`
	
	`SetCursor 100 500`

Example	Terminal Emulator application definition
	
	This example sets the cursor to the correct position, and then you enter credentials.
	
	`SetCursor 200`
	`Type $Username`
	`Type @E`
	`Type $Password`
	`Type @E`

SetFocus

Use with	Java and Windows
SecureLogin version	3.5 or later
Type	Action
Arguments	**<#Ctrl-ID>** The ID number of the control to which the keyboard focus is directed.
Description	Use the `SetFocus` command to set the `keyboard` focus to a specified control ID. A valid control ID is required for the `SetFocus` command to function correctly.

Example	Windows application definition
	
	This example sets the focus to the username field (#1001). The username is typed and a tab stop is simulated, and then the password is typed and pressing **Enter** is simulated.
	
	`# Logon Dialog Box`
	`Dialog`
	`  Class #32770`
	`  Title "Log on"`
	`EndDialog`
	
	`SetFocus #1001`
	`Type $Username`
	`Type \T`
	`Type $Password`
	`Type \N`

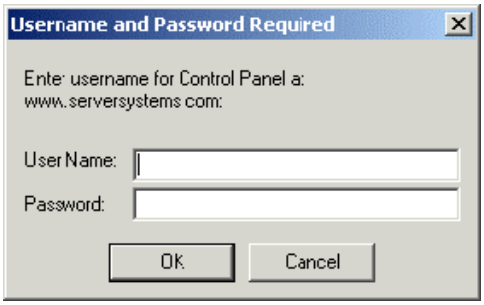
SetPlat

Use with	All
SecureLogin version	3.5 or later
Type	Action
Usage 1	SetPlat **<Application-Name>**
Usage 2	SetPlat **<RegEx> <Variable> <#Ctrl-ID>**
Arguments	**<Application-Name>** Application name from which to read the variables. **<RegEx>** Regular expression to use as application name. **<Variable>** Use a previously set ?Variable, for example, using a PickList (see “PickListAdd” on page 128). **<#Ctrl-ID>** The control ID number of the regular expression. For information regarding regular expressions see: (http://www.boost.org/doc/libs/1_33_1/libs/regex/doc/syntax_perl.html)

Description	By default, variables are stored directly against the platform or application on which you have SecureLogin enabled. For example, if you enable Groupwise.exe, the Groupwise credentials are stored against the Groupwise.exe platform. SetPlat sets the platform or application from which variables are read and saved if you have: ◆ Multiple accounts (for example, your own log on and an admin log on) accessing the same platform or application. ◆ Multiple platforms or applications using a common set of credentials? Other uses of SetPlat include: ◆ Configuring application1 to read it's $Username and $Password from application2. This saves a user from entering the credentials twice and having to remember to update them in both locations when they change, and so on. ◆ Configuring application1, application2, and application3 to read the users credentials from Platform Common. This results in a single store of common credentials which you only need to update once.

Example 1 Windows application definition

Following is a standard dialog box for accessing a password protected site. The dialog box is generated by the browser itself. The details for this window must be specified through a Windows script and not a web script. If the browser is Mozilla Firefox, we must create a Windows application definition for Firefox.exe.



When you specify the Title, Class, User Name, and Password fields for this dialog box in Firefox, they display the same dialog box irrespective of the password protected site. Since the FireFox browser is generating this window, the same dialog box is used with any password protected site and not just the site www.serversystems.com.

However, the previous dialog box always contains the name of the Web site to which to log on. You can use this name as the unique identifier in order to set a new platform and to save the log on credentials.

Using a dialog block with a SetPlat statement The solution is to use a dialog block with a SetPlat statement such as:

```
Dialog
   Ctrl #330
   Ctrl #214
   Ctrl #331
   Ctrl #1
   Ctrl #2
   Title "Username and Password Required"
   SetPlat #331 "Enter username for (.*) at (.*):"
EndDialog
Type $Username #214
Type $Password #330
Click #1
```

The power of this application definition is the line:

```
SetPlat #331 "Enter username for (.*) at (.*):"
```

This reads the line from dialog control ID 331, enters the user name for Control Panel at www.serversystems.com, and applies the regular expression to this text. Regular expressions are a way of manipulating text strings, however, for most purposes a few very basic commands work.

	For information regarding regular expressions see: www.boost.org/doc/libs/1_33_1/libs/regex/doc/syntax_perl.html (http://www.boost.org/doc/libs/1_33_1/libs/regex/doc/syntax_perl.html) When the user has run the application definition, they will see the user name and password saved as www.serversystems.com. The text matched inside the brackets then becomes the symbol application. If a dialog <#Ctrl-ID> is not specified, the symbol application is unconditionally changed to the application specified in <RegEx>. An unconditional `SetPlat` command is only valid if specified before Dialog/EndDialog statements.

Example 2	Windows application definition
	
	This example displays a pick list and sets a new platform so multiple users can log on to the application. In this case, SetPlat creates a new platform called Default User, Global Administrator, or Regional Administrator, and the respective $Username and $Password is saved there.
	
	`# Logon Dialog Box`
	`Dialog`
	`  Class #32770`
	`  Title "Log on"`
	`EndDialog`
	
	`PickListAdd "Default User"`
	`PickListAdd "Global Administrator"`
	`PickListAdd "Regional Administrator"`
	`PickListDisplay ?Choice "Please select the account you wish to use"-NoEdit`
	`SetPlat ?Choice`
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #3`

SetPrompt

Use with	All
SecureLogin version	3.5 or later
Type	Action
Usage	SetPrompt **<Prompt-Text>**
Arguments	**<Prompt-Text>** The customized text prompt displayed in the Enter SecureLogin Variables dialog box.

Description

`SetPrompt` is invoked anytime a user would be prompted for the values in stored variables. For example, a newly created application where user's credentials have not been set, will invoke the SecureLogin Variables dialog box. This box has a standard header text and the fields are represented with the standard User and Password labels. The `SetPrompt` command allows you to customize these values so that the user is prompted with a more precise message. For example, you may need to prompt in the user's native language or you would like to indicate what type of password or restrictions may apply. `SetPrompt` can also be used to customize the same dialog box when displayed with the `DisplayVariables` command. For more information, see "DisplayVariables" on page 82.

NOTE: Positioning of the `SetPrompt` command is crucial. Position it before the first usage of each variable to name that variable, and apply the final Setprompt to the text displayed at the top of the prompt screen.

Example 1

Windows application definition

This example replaces the default text prompt in the Enter SecureLogin Variables dialog box. It places the `SetPrompt` command after the last variable typed.

```
# Logon Dialog Box
Dialog
   Class #32770
   Title "Log on"
EndDialog

Type $Username #1001
Type $Password #1002
SetPrompt "Please enter your user name and password for
accessing the Human Resources system.  These credentials
will be remembered by SecureLogin and you will be
automatically logged on in future. IT Help Desk x4532"
Click #1
```

Example 2

Windows application definition

This example replaces the text prompt next to any variable entry field in the Enter SecureLogin Variables box and places the `SetPrompt` command immediately before the variable in the application definition.

```
Logon Dialog Box
Dialog
   Class #32770
   Title "Log on"
EndDialog

SetPrompt "Enter Username==>"
Type $Username #1001
SetPrompt "Enter Password==>"
Type $Password #1002
SetPrompt "Please enter your user name and password for
accessing the Human Resources system.  These credentials
will be remembered by SecureLogin and you will be
automatically logged on in future. IT Help Desk x4532"
Click #1
```

Site/Endsite

Use with	Advanced Web/Web Wizard Script application definitions created using the Web Wizard.
SecureLogin Version	3.5.x or later
Type	Action
Usage	Site ["Name" [-userid "userid"] [-initial|-subsequent|-recent timeout] [-nonexclusive]]

Arguments	`Site`
	
	The Site/EndSite commands are used to match a particular site given a set of filters. Site/EndSite usage is much the same as the Dialog/EndDialog commands found in the windows scripting commands.
	
	`Name`
	
	Name is a static string used to denote the site being matched. The Name cannot be a variable and the same value can be used by multiple Site commands to specify a match for the same site under differing conditions.
	
	`-userid "userid"`
	
	Specifies the default set of credentials to be used for this site block.
	
	NOTE: "userid" must be a static string.
	
	`-initial`
	
	Specifies that this site block will only match the first time.
	
	`-subsequent`
	
	Specifies that this site block will only match after an initial match has already been made.

Arguments	`-recent timeout`
	
	Specifies that this site block will only match if a previous match was made within the given timeout period.
	
	Timeout is given in milliseconds.
	
	`-nonexclusive`
	
	Specifies that even if this site block matches, other scripts and wizards will not be prevented from running.
	
	`-events create mutate`
	
	Specifies the subset of an event to monitor the webpage and execute the scripts.
	
	NOTE: To ensure backward compatibility, the mutate event is raised only when the following preference is enabled for Web group:
	
	Add application prompts for web pages on mutation. The event monitoring feature is enabled when the **Enable DHTML monitor on web pages** is set to **Yes**.

Description	Site/EndSite begins and ends a site definition, similar to the Dialog/ EndDialog commands used in Windows application definition scripts. There can be multiple site definitions within a single advanced web application script to identify different sites within the same domain. Site/EndSite blocks are used to define all the parameters SecureLogin would expect to find on a Web page to run the application definition. 'Match' commands can be used to filter a given site. If one of the contained match commands fails to match, then the site block fails to match as a whole.

Example 1	This simple example will match against the website www.mybank.com.
	
	`# === My Bank Initial Logon ===`
	`Site "www.mybank.com" -userid "My Logon Credentials" -`
	`initial`
	`EndSite`

Example 2	This simple example will match the Web site www.google.com, match the login form fields and logs on to the user's account using the user's e-mail address, password, and don't remember checkbox unchecked.
	
	`# === Logon Application Definition #2 ==`
	`# === Google Initial Logon ====`
	`#=====================================`
	`Site Login -userid "Google Log On" -initial`
	`  MatchDomain "www.google.com"`
	`  MatchField #1:1 -name "Email" -type "text"`
	`  MatchField #1:2 -name "Passwd" -type "password"`
	`  MatchField #1:3 -name "Cookie" -type "check"`
	`EndSite`
	
	`SetPrompt "Enter your user credentials"`
	`TextInput #1:1 -value "$Username"`
	`TextInput #1:2 -value "$Password"`
	`FocusInput#1:2 -focus "true"`
	`BooleanInput #1:3 -check "false"`
	`PressInput`
	`Endscript`

Example 3	The following site definitions show examples of how the -events argument could be used. Note that the preference, **Add application prompts for web pages on mutation** must be set to **Yes** for SecureLogin to use this argument. For more information see the description for `-events` in the arguments section above.
	
	a) Using Site/endsite without the `-events` option is the same as using the option `-events create`:
	
	`Site Login`
	`endsite`
	
	`Site Login -events create`
	`endsite`
	
	b) To ignore creation event and only handle when the page changes:
	
	`Site Login -events mutate`
	`endsite`
	
	c) To act on either creation or mutation:
	
	`Site Login -events create mutate`
	`endsite`

-SiteDeparted

Use With	Web
Novell SecureLogin version	3.5 or later
Type	Action
Argument	`SiteDeparted` is a conditional variable.
Description	Use the SiteDeparted variable in Web scripts to see if the current document is still active when used as part of an If statement.

Example	The following example checks if the user has navigated away from the current Web site or not.
	
	If the users have navigated away from the Website, it informs the users and exists the script.
	
	`GotoURL "www.google.com"`
	`Delay 1000`
	`If -SiteDeparted`
	`  MessageBox "Script terminated, we have left the web-site"`
	`  EndScript`
	`EndIf`

StrCat

Use with	All
SecureLogin Version	3.5 or later

Type	Action
Usage	StrCat **<Variable> <Input-String1> <Input-String2>**
Arguments	**<Variable>** The variable to which you want a result saved. **<Input-String1>** First data string or variable. **<Input-String2>** Second data string or variable.
Description	Use the `StrCat` command to append the second data string to the first data string. For example, StrCat ?Result "SecureRemote " "$Username". In this case "$Username" is "Tim", and the variable "?Result" now contains the value "SecureRemote Tim".

Example	Windows application definition
	
	This example reads the user name from #1001 into ?Username and uses the `StrCat` command to append the ?Username value with the value of $Password. The resulting string is returned in the ?LoginID variable, which SecureLogin then uses to log on to the system.
	
	`# Logon Dialog Box`
	`Dialog`
	`  Class #32770`
	`  Title "Log on"`
	`EndDialog`
	
	`ReadText #1001 ?Username`
	`StrCat ?LoginID $Username $Password`
	`Type ?LoginID #1002`
	`Click #1`

StrLength

Use with	All
SecureLogin version	3.0.4 or later
Type	Variable manipulator
Usage	StrLength **<Destination> <String>**
Arguments	**<Destination>** The output variable that will contain the results of the string length computation. **<String>** The string whose length you want to measure.
Description	Use the `StrLength` command to count the number of characters in a variable and output that value to the destination variable.

Example	Windows application definition
	
	This example reads the password from #301 and then uses StrLength to count the number of characters. If it is less that 4, an error message is displayed.
	
	`Logon Dialog Box`
	`Dialog`
	`  Class #32770`
	`  Title "Log on"`
	`EndDialog`
	
	`ReadText #301 ?Password`
	`StrLength ?Length ?Password`
	`If ?Length Lt "4"`
	`  MessageBox "Password is too short"`
	`EndIf`

StrLower

Use with	All
SecureLogin Version	3.0.4 or later
Type	Variable manipulator
Usage	StrLower **<Destination> [<Source>]**
Arguments	**<Destination>** The output variable. Also the input variable if no source is specified. **[<Source>]** The input variable. If not specified, SecureLogin reads the destination variable, makes the necessary changes, and writes over the variable.
Description	Use the `StrLower` command to modify a variable so that all the characters are lower case. If only a: ◆ Destination variable is specified, the string is read from the destination, then is stored back to it. ◆ Source variable is specified, the string is read from the source, and the modified value is stored in the destination variable. In this case, the source variable remains unchanged.

Example	Windows application definition
	
	The example reads the user name from #1001 and copies it into ?Username. The `StrLower` command is then used to make sure the user name is all lower case.
	
	`# Logon Dialog Box`
	`Dialog`
	`  Class #32770`
	`  Title "Log on"`
	`EndDialog`
	
	`ReadText #1001 ?Username`
	`StrLower ?LowerCaseUsername ?Username`
	`Type ?LowerCaseUsername #1002`
	`Click #1`

StrLower

Use with	All
SecureLogin Version	3.0.4 or later
Type	Variable manipulator
Usage	StrLower **<Destination> [<Source>]**
Arguments	**<Destination>** The output variable. Also the input variable if no source is specified. **[<Source>]** The input variable. If not specified, SecureLogin reads the destination variable, makes the necessary changes, and writes over the variable.
Description	Use the `StrLower` command to modify a variable so that all the characters are lower case. If only a: ◆ Destination variable is specified, the string is read from the destination, then is stored back to it. ◆ Source variable is specified, the string is read from the source, and the modified value is stored in the destination variable. In this case, the source variable remains unchanged.

Example	Windows application definition
	
	The example reads the user name from #1001 and copies it into ?Username. The `StrLower` command is then used to make sure the user name is all lower case.
	
	`# Logon Dialog Box`
	`Dialog`
	`  Class #32770`
	`  Title "Log on"`
	`EndDialog`
	
	`ReadText #1001 ?Username`
	`StrLower ?LowerCaseUsername ?Username`
	`Type ?LowerCaseUsername #1002`
	`Click #1`

StrReplace

Use with	All
SecureLogin version	8.7 or later
Type	Variable manipulator
Usage	`StrReplace` **<Destination> search replace <Source>**
Arguments	**<Destination>** The output variable. **<Source>** The input variable. **search** It is the existing character or string in the **<Source>** variable that is to be replaced. **replace** It is the new character or string that replaces the existing character or string (**search**) in the **<Source>** variable.
Description	Use the `StrReplace` command to replace a character or string in a source variable with a new character or string. The result is stored in the destination variable.

Example	`Dialog`
	`Title "Untitled - Notepad"`
	`EndDialog`
	`Type $Username`
	`StrReplace ?Us \ \\ $username`
	`Type ?Us`
	`StrReplace ?Us abc Test ?Us2`
	`Type ?Us2`
	`EndScript`
	
	If Username is `abc\n\tdef` then the **<Us>** variable will store `abc\\n\\tdef` and **<Us2>** will store `Test\n\tdef`.

StrUpper

Use with	All
SecureLogin version	3.0.4 or later
Type	Variable manipulator
Usage	StrUpper **<Destination> [<Source>]**
Arguments	**<Destination>** The output variable. Also the input variable if no source is specified. **[<Source>]** The input variable. If not specified, SecureLogin reads the destination variable, makes the necessary changes, and writes over the variable.
Description	Use the `StrUpper` command to modify a variable so that all the characters are upper case. If only a: ◆ Destination variable is specified, the string is read from the destination and is then stored back to it. ◆ Source variable is specified, the string is read from the source, and the modified value is stored in the destination variable. In this case, the source variable remains unchanged.

Example	Windows application definition
	
	This example reads the user name from #1001 and copies it into ?Username. The `StrUpper` command is then used to make sure the user name is all upper case.
	
	`# Logon Dialog Box`
	`Dialog`
	`  Class #32770`
	`  Title "Log on"`
	`EndDialog`
	
	`ReadText #1001 ?Username`
	`StrUpper ?UpperCaseUsername ?Username`
	`Type ?UpperCaseUsername #1002`
	`Click #1`

Sub/EndSub

Use with	Startup, Terminal Emulator, Web, or Windows
SecureLogin version	3.5 or later
Type	Flow control
Usage	Sub **<Name>** EndSub

Arguments	**<Name>** Any name entered to identify the subroutine.
Description	Use the Sub/EndSub commands around a block of lines within an application definition to denote a subroutine. A subroutine defined with Sub/EndSub commands are called through the script command `Call`. For more information on calling subroutines, see “Call” on page 67.

Example	Terminal Emulator application definition
	
	This example checks the emulator screen for the text Log On or Wrong Password. If either is found, the appropriate subroutine is called and run. After the subroutine completes, the script will continue processing the remaining commands in the application definition script.
	
	`If -Text "Log On"`
	`  Call "Login"`
	`EndIf`
	`If -Text "Wrong Password"`
	`  Call "WrongPassword"`
	`EndIf`
	
	`Sub Login`
	`  Type $Username`
	`  Type @E`
	`  Type $Password`
	`  Type @E`
	`EndSub`
	
	`Sub WrongPassword`
	`  DisplayVariables "Enter correct password" $Password`
	`  Call Login`
	`EndSub`

Submit

Use with	The valid Web related application definitions that really apply.
SecureLogin version	3.5 or later
Type	Action
Usage	Submit
Arguments	None

Description	Use the `Submit` command only in Web application definitions and only with Internet Explorer to allow for enhanced control of how and when a form is submitted. The `Submit` command performs a Submit on the form in which the first password field is found. The `Submit` command is ignored if used with FireFox. The function performed by the `Submit` command is automatically performed by Web application definitions by default. For example, the application definition: Type $Username Type $Password Password Types the user name and password and submits the form. **When submits do not occur automatically** However, submits do not occur automatically if any of the following commands are in the application definition: Type \N, Type \T, or Click. Furthermore, an automatic submit does not occur if you type text into a specific text entry field. For example, in the application definition snippet below, the `Submit` command must follow the `Type` command for the application definition to work properly: Type $Username #1001 Submit
Example	Web application definition This example enters the user name and password and then executes a manual Submit. Type $Username #1 Type $Password #2 **Submit**

Substr

Use with	Startup, Terminal Emulator, Advanced Web/Web Wizard Script, or Windows
SecureLogin version	7.0.3 or later
Type	Action
Usage	SubStr **[<var result>] [<var source>] [<var start>] [<var count>]**

Arguments	**<var result>** The <var result> argument is the variable where the sub text is stored. **<var source>** The <var source> argument is the actual string. **<var start>** The <var start> argument is the index number of the sub text. **<var count>** The <var count> argument is the number of characters from the <var start> position.

Example	Windows application definition
	
	This example displays a subtext from the given string.
	
	`Substr ?result abc123ABC!@# 3 6`
	`?result 123ABC`

SubstVar

Use with	Java, Startup, Terminal Emulator, Advanced Web/Web Wizard Script, or Windows
SecureLogin version	6.0 or later
Type	Action
Usage	SubstVar **[<var result>] [varvar]**
Arguments	**<var result>** The <var result> argument is a variable in which the value of the varvar variable is stored. **<varvar>** The <varvar> argument contains a variable of which value changes according to the variable manipulators or other commands that are used in the script.
Description	This command is used to create an array or a group.

Example	This example evaluates the variable that is stored in <varvar> and places the value in <var result>. If the variable that is stored in <varvar> is a runtime variable named password, then the following is equivalent to `Set ?result ?password`.
	
	`SubstVar ?result ?varvar`
	
	Following example can help in accessing password history for first ten passwords:
	
	`Set ?Count 0`
	`Repeat 10`
	`  Strcat ?varvar  password ?Count`
	`  SubstVar ?currPass ?varvar`
	`# obtain the required details from each password`
	`  Increment ?Count`
	`EndRepeat`

Subtract

Use with	Startup, Terminal Emulator, Advanced Web/Web Wizard Script, or Windows
SecureLogin version	3.0 or later
Type	Variable manipulator
Usage	Subtract **<Start-Value> <Subtract-Value> [?Result]**
Arguments	**<Start-Value>** The <Start-Value> argument is the start number from which the second argument is subtracted. This argument contains the result if the optional [?Result] argument is not passed in. If used: ◆ Without the [?Result] argument, then <Start-Value> must be a SecureLogin variable, for example, ?StartValue or $StartValue. ◆ With the [?Result] argument, then <Start-Value> can be a SecureLogin variable or a numeric value. **<Subtract-Value>** The <Subtract-Value> argument is the number subtracted from the first argument. <Subtract-Value> can be a SecureLogin variable or a numeric value. **[?Result]** The result of the equation. This argument is optional but, if used, set to <Start-Value> - <Subtract-Value>. The [?Result] must be a SecureLogin variable, for example, $Result or ?Result.

Description	Use the `Subtract` command to subtract one value from another. Other numeric commands include the Add, Divide, and Multiply. For more information see: ◆ “Add” on page 62 ◆ “Divide” on page 84 ◆ “Multiply” on page 120 **NOTE:** The `Subtract` command correctly subtracts when <Start-Value>, <Subtract-Value> and <Result-Value> are between -2147483648 and +2147483647.

Syntax examples:	`Subtract "1" "2" ?Result`
	`Subtract ?LoginAttempts ?LoginFailures`
	`Subtract ?LoginAttempts ?LoginFailures ?Result`
	`Subtract ?LoginAttempts "3"`
	`Subtract ?LoginAttempts "3" ?Result`

Example	Windows application definition
	
	This example reads the values of control IDs 103 and 104 into variables. From there they are subtracted and typed into control ID 1.
	
	`ReadText #103 ?Number1`
	`ReadText #104 ?Number2`
	`Subtract ?Number1 ?Number2 ?Result`
	`Type ?Result #1`

Tag/EndTag

Use with	Advanced Web application definitions
SecureLogin version	3.5 or later
Type	Tag specifier
Usage	Tag <Form-Name> EndTag
Arguments	<Form-Name> The form name is an optional value given to a form by the creator of the Web site.
Description	Use the Tag/EndTag commands to find HTML tags.

Example	This example finds the form that has an attribute of Name with a value of Log on.
	
	`Tag "Form"`
	`  Attribute "Name" "Log on"`
	`EndTag`

TextInput

Use with	Advanced Web application definitions created using the Web Wizard Script, WinSSO, JavaSSO and .NetSSO workers.
SecureLogin version	3.5.x or later
Type	Action
Usage	TextInput #FormID:FieldID -value "value"
Arguments	#FormID:FieldID The ID that was given to the matched field in the Site block using `MatchField` command. The FormID and FieldID must be unsigned integers. -value "value" The text value to be input.
Description	Use the `TextInput` command after a Site block to input text into a specified field. You can enter text into fields of type password/text/textarea/file.

Example	In this example the text value of the system user name and password are passed to the application definition.
	
	`# === Logon Application Definition #2 ==`
	`# === Google Initial Logon ====`
	`#=====================================`
	`Site Login -userid "Google Log On" -initial`
	`  MatchDomain "www.google.com"`
	`  MatchField #1:1 -name "Email" -type "text"`
	`  MatchField #1:2 -name "Passwd" -type "password"`
	`  MatchField #1:3 -name "Cookie" -type "check"`
	`EndSite`
	
	`SetPrompt "Enter your user credentials"`
	`TextInput #1:1 -value "$Username"`
	`TextInput #1:2 -value "$Password"`
	`FocusInput#1:2 -focus "true"`
	`BooleanInput #1:3 -check "false"`
	`PressInput`
	`Endscript`

Title

Use with	Java, Windows, .NET etc.
SecureLogin version	3.5 or later
Type	Dialog specifier
Usage	Title **<Window-Title>** [-regex “regular expression”]

Arguments	**<Window-Title>** The text to test against the window title. -regex You may also use regular expressions to match part of a URL, such as the domain only. For more information regarding regular expressions see: www.boost.org/doc/libs/1_33_1/libs/regex/doc/syntax_perl.html (http://www.boost.org/doc/libs/1_33_1/libs/regex/doc/syntax_perl.html)
Description	Use the `Title` command to retrieve the title of a window and compare it against the string specified in the <Window-Title> argument. For this block of the application definition to run, the retrieved window title and the <Window-Title> argument must match the text supplied to the `Title` command in the dialog block. Title is one of the main commands to identify a window. However, the `Title` command alone may not be enough – if there is more than one window in a platform (application) with the specified title, the SecureLogin application definition will run every time that window is detected. Make `Title` the first command in the Dialog block to speed the matching process and ensure that all detected controls are also created. However, with some applications, if the text to match is too long, this will slow the detection and creation process. Consequently, if your application definition is unusually slow to execute, try placing the `Title` command after all other commands in the Dialog block. For Windows applications, either Title or Class should be defined in a Dialog block at least once. **Uniquely identifying a window** To uniquely identify a window, the `Title` command is typically used with the Class or Ctrl commands. For more information, see “Class” on page 70 and “Ctrl” on page 77. **NOTE:** Use the Window Finder tool to determine the window title.

Example 1	Windows application definition
	
	This example tests the dialog box to see if it has the correct title. If the title is not correct, the application definition passes on to the next Dialog block.
	
	`# Logon Dialog Box`
	`Dialog`
	`  Class #32770`
	`  Title "Logon"`
	`EndDialog`
	
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

Example 2	Windows application definition
	
	This example uses a regular expression to identify the window title.
	
	`Dialog`
	`  Class "#32770"`
	`  Parent`
	`      Class "#32770"`
	`      Title -regex "training"`
	`   EndParent`
	`   Ctrl #1001`
	`   Ctrl #1002`
	`   Ctrl #1`
	`   Title "Logon - Simple"`
	`EndDialog`

Type

Use with	Java, Terminal Emulator, Web Wizard Script/Advanced Web, or Windows
SecureLogin version	3.5 or later
Type	Action
Terminal usage	Type [-Raw] <Text>
Windows usage	Type <Text> [<#Ctrl-ID>]
	Type [-Raw] <Text>
	Type [-order] <Text> [<#Order-ID>]
	Type [-msg] <Text> [<#Ctrl-ID>]
Web usage	Type <Text> [<#Field-ID>]
	Type <Text> ["password"]
	Type [-Raw] <Text>

Arguments	[-Raw] By default, when typing into a terminal emulator or Windows application, SecureLogin verifies that the window exists before continuing. This verification process is disabled when the -Raw argument is provided. Furthermore, instead of setting the text in the field directly, the -Raw argument simulates actual keystrokes, causing SecureLogin to type into whichever window has focus.The -Raw argument can also be used in a Web application. The -Raw argument attempts to type the text into the window that owns the Web page (Internet Explorer or Firefox) and works the same as -Raw on Windows applications. [-order] If the control ID's are not constant, utilize the -order argument to instruct SecureLogin to type into a control based on the creation order and not the tab order. For more information on the -order argument usage, see “Example 5” on page 179. [-msg] The -msg argument can be used when a Type command is sending the data correctly, but the application is not successfully reading the data. The -msg argument will only work in Windows applications as the argument simulates the keys being pressed (that is, key down, character, key up). The -msg argument sends the data character by character versus sending the text string all at once. This -msg option is often useful for older Windows applications, particularly old versions of Lotus Notes. **<Text>** The text to type into this area. This text can be static text, such as ABC, or any SecureLogin variable, such as $Username. **[<#Ctrl-ID>]** For Windows application definitions, this optional argument specifies the control into which to type the text. Use the Window Finder to extract these control IDs. For more information, see “Windows specific” on page 178. [<#Order-ID>] For Windows application definitions, this parameter specifies which control based on the creation order in which to type the text.

[<#Field-ID>]

For Web application definitions, this optional argument specifies the text field into which to type the text. For more information, see “Web specific” on page 178.

[“password”]

For Web application definitions, this optional argument specifies the field is of type "password". HTML controls with the type "password" are masked so that any values specified will be obscured from view as the user types in the value. For example, typing password into a field set with the type "password" will display "********" on the screen. SecureLogin will look at the HTML code to find the field with this type set. Typically, only password fields are set with this type. Other fields will be of the type "text". If [password] is used, that application's application definition cannot use a <#Field-ID> argument. For more information, see “Web specific” on page 178.

For example the following HTML source shows a username and a password field.

```
<tr>
   <td align="right" width="35%">Username:</td>
   <td align="left"><input name="User.id" id="username"
style="width:198px;" value="" type=text
autocomplete="off" class="text" MAXLENGTH=64></td>
</tr>
<tr>
   <td align="right">Password:</td>
   <td align="left"><input name="User.password"
style="width:198px;" type=password autocomplete="off"
MAXLENGTH=32 class="text" ></td>
</tr>
```

Description

Use the Type command to enter data such as user names and passwords into applications. There are reserved character sequences that are used to type special characters, for example Tab and Enter. If it is not possible to determine control IDs in a Windows application, and the Type command is not working, use the SendKey command instead.

Windows specific In Windows, if the <#Ctrl-ID> argument is:

- Provided, it must be a number that refers to a control ID as identified by the Window Finder Tool. SecureLogin will then send the contents of the <Text> argument directly to the window and to the specific control that matches the <#Ctrl-ID> argument.
- Not specified, SecureLogin will send keystrokes to whichever control has focus. In the Windows environment, the -Raw option is often useful when the Window Finder Tool is unable to determine control IDs for the text entry areas of an application, or these control IDs are changing. If using the -Raw argument, the <#Ctrl-ID> argument is ignored.

Web specific For Web pages, there are two ways to specify which field receives <Text>.

- The first method uses absolute positioning by means of the <#Field-ID> argument. The <#Field-ID> is a number that refers to the location of the field within the HTML form. For example, #1 refers to the first text entry field in the Web form; #2 refers to the second text entry field, and so on.
- The second method uses relative positioning using the password argument. In this method the SecureLogin agent first locates the text field within the HTML form that is a password field, and types <Text> into that field. Other type commands send their <Text> parameters to fields that are relative to the first password field.

  For example, the Type command immediately preceding the Type command that has the [Password] argument is sent to the text field immediately preceding the first password field.

Example 1

Windows application definition

This example shows the use of the Type command in a Windows application definition.

```
# Logon Dialog Box
Dialog
   Class #32770
   Title "Log on"
EndDialog

Type $Username #1001
Type $Password #1002
Type "DB2" #1003
Click #1
```

Example 2 Windows application definition

This example shows the use of the -Raw argument. This argument is not actually required for the application but is used as an example.

```
# Calculator Is Active
Dialog
   Class #SciCalc
   Title "Calculator"
EndDialog

Type -Raw "15"
Type -Raw "+"
Type -Raw "20"
Type -Raw "="
```

Example 3 Windows application definition

This example shows the use of the -msg argument. In this instance the argument is not actually required for this application but is used as an example.

```
# Calculator Is Active
Dialog
   Class #SciCalc
   Title "Calculator"
EndDialog

Type -msg $Password #480
```

Example 4 Windows application definition

The following syntax examples compare and contrast the use of the various Type command arguments.

```
Type #1 "text"
```

Will type text into control with ID of 1

```
Type #1 "text" -order
```

Will type text into the first control drawn in the application dialog window.

```
Type #1 "text" -msg
```

Will type text character by character into the first control with an ID of 1.

```
Type #1 "text" -raw
Type #1 "text" -focus
```

Ignores the unused parameter #1

Example 5 Windows application definition

This example shows the use of the -order argument and demonstrates the possible syntax that can be used.

```
Type -order #1 "some text"
Type #2 "some text" -order
Type "some text" -order #3
```

Example 6	Web application definition
	
	This example shows the use of the HTML type password as an argument to find the appropriate Password field.
	
	`Type $Username`
	`Type $Password Password`
	
	In the application definition above, the SecureLogin agent locates the first password field. The first `Type` command sends $Username to the field immediately before the password field. The second `Type` command sends $Password to the password field. The same application definition could be rewritten using absolute placement as shown below. In the following example, the `Submit` command is also used to automatically submit the page.
	
	`Type $Username #1`
	`Type $Password #2`
	`Submit`

Sending keyboard commands using Type

SecureLogin can send special keyboard keystrokes to Windows and Web-based applications to emulate the user's keyboard entry. The `Type` command passes keystrokes to the window that the application definition is defined for. These special keystrokes include the ability to select menu items, special keys such as Alt for F1, and other special keyboard combinations.

Special key commands

Type	Simulates
\Alt+<key>	Pressing the ALT key plus the desired <key>.
\Shift+<key>	Pressing the SHIFT key plus the desired <key>.
\Ctrl+<key>	Pressing the CTRL key plus the desired <key>.
\LWin+<key>	Pressing the left Windows key plus the desired <key>.
\RWin+<key>	Pressing the right Windows key plus the desired <key>.
\Apps+<key>	Pressing the Application key plus the desired <key>.

Raw key commands

You can also use the `Type` command to send a combination of raw key commands. “Windows Keyboard Functions” on page 193 details the available keyboard sequences you can use with the `Type` command.

Type	Simulates
|<xxx>	The format for sending a raw key command, where <xxx> represents the keyboard code.
|18+65	Pressing the ALT-A keys in sequence.

Type commands used with Terminal Launcher

The use of the `Type` command to send special characters in a Terminal Emulator definition is dependent upon the emulator definition defined in tlaunch. The section below applies to only HLLAPI based emulator definitions for Generic and Advanced Generic emulator definitions, use the `SendKey` command. For more information, see section"SendKey" on page 150. Listed below are the @ keys that you can use with the type command for HLLAPI based emulator definitions. These commands perform specific emulator and mainframe functions. For example, you can send functions such as Enter, Tab, System Request, and Clear.

The example below shows the use of the @ commands in a Terminal Emulator application definition.

TYPE @ command

WAITFORTEXT "Log on:"

Type $username

Type @T

Type $password

Type @E

The "Terminal Emulator Commands" on page 198 details the commands that you can use within a Terminal Emulator application definition.

WaitForFocus

Use with	Windows
SecureLogin version	3.5 or later
Type	Flow control
Usage	WaitForFocus **<#Ctrl-ID> [<Repeat-Loops>]**
Arguments	**<#Ctrl-ID>**
	The ID number of the control with the focus.
	[<Repeat-Loops>]
	The number of repeat-loops that will run.
Description	Use the `WaitForFocus` command to suspend the running of the application definition until the <#Ctrl-ID> has received keyboard focus, or the <Repeat-Loops> expire. The <Repeat-Loops> is an optional value that defines the number of loop cycles to run. The <Repeat-Loops> value defaults to 3000 loops if nothing is set. Once focus is received, the application definition continues.
	Set the value of <Repeat-Loops> to a negative number, for example, WaitForFocus "$1065" "-1", for the loop to never expire.
	NOTE: Do not place WaitForFocus commands within Dialog / EndDialog statements.

Syntax examples	`WaitForFocus #301`
	`WaitForFocus #301 "2000"`
	`WaitForFocus #301 "0"`
	`WaitForFocus #301 "-1"`

Example 1	Windows application definition
	
	This example will look for a window that matches the criteria specified in the Dialog/EndDialog section. Once found, it will wait indefinitely for control 301 to receive focus before it will submit the user's credentials to the application.
	
	`# Logon Dialog Box`
	`Dialog`
	`  Class #32770`
	`  Title "Log on"`
	`EndDialog`
	
	`WaitForFocus #301 "-1"`
	`Type $Username`
	`Type \T`
	`Type $Password`
	`Type \N`

Example 2	This example has the `WaitForFocus` command suspend the running of the application definition until control ID #15 is reached and a message box with "love" should appear.
	
	`## BeginSection: "Logon Window"`
	`Dialog`
	`  Class "Notepad"`
	`  Title "Untitled - Notepad"`
	`EndDialog`
	
	`Setprompt "Optional:"`
	`# Here the correct ID with the loops set to 0`
	`WaitForFocus #15 0`
	`Set ?thu "love\me"`
	`RegSplit "(.*)\\(.*)" ?thu ?Domain ?User`
	`MessageBox ?Domain`
	`## EndSection: "Logon Window"`

WaitForText

Use with	Terminal Emulator
SecureLogin version	3.5 or later
Type	Flow control
Usage	WaitForText <Text>
Arguments	<Text> The text for which the application definition is waiting.

Description	Use the `WaitForText` command to make SecureLogin wait for the specified text to display before continuing. For example, you may make SecureLogin wait for a user name field to display before attempting to type a user name. The text may appear anywhere on the terminal screen and is usually case sensitive (this depends on the Terminal Emulator itself). If the text is written in the wrong case, the terminal launcher will pause and try to find the correct text in the correct case, until the terminal screen times out. Since `WaitForText` will search with the character case specified in <Text>, it is common practice to remove the initial letter from the first word. For example, `WaitForText` "logon" will work regardless of whether the text "logon" or "Logon" is displayed. Also, some terminal emulators will not correctly match text that is hard against the left margin of the window. Again, if you encounter this situation, try to match text without the leading character.

Example	Terminal Launcher application definition
	
	This command instructs SecureLogin to wait for the text ogin: to appear on the emulator screen before entering the user name. It will then wait for password: to display before entering the password.
	
	The WaitForText cuts off the first character because it finds both Password and password, and responds to all password entry points.
	
	`WaitForText "ogin:"`
	`Type $Username`
	`Type @E`
	`WaitForText "assword:"`
	`Type $Password`
	`Type @E`

While/Endwhile

Use with	Startup, Terminal Emulator, Web, or Windows
SecureLogin version	7.0.3 HF1 or later
Type	Flow control

Usage 1	`While` **`<Value1>`** `<Gt|Lt>` **`<Value2>`**
	`#Do This`
	`Endwhile`

Usage 2	`While` **`<Value1>`** `<Eq|NotEQ >` **`<Value2>`** `[-I|-S]`
	`#Do This`
	`Endwhile`

Usage 3	`While` **`<Value1>`** `<-In|-NotIn>` **`<Value2>`** `[-I|-S]`
	`#Do This`
	`Endwhile`

Usage 4	`While -Text [-Frame]` **`<Text>`**
	`#Do This`
	`Endwhile`

Usage 5

```
While -Exist|-NotExist <Variable>
#Do This
Endwhile
```

Arguments

<Value1>

The left side of the expression for evaluation.

<Value2>

The right side of the expression for evaluation.

<Text>

The text for which you are searching.

Description

Condition: It is a boolean expression. If condition is nothing, then SecureLogin considers the condition as False.

While: The condition following the `While` command will run until the condition is True.

Endwhile: Terminates the definition of the `While` block.

Text comparison operators supported The text comparison operators supported by the While command are:

- Eq: True if the left side is equal to the right side.
- NotEQ: True if the left side is not equal to the right side.
- -In: True if the left side is a substring of the right side.
- -NotIn: True if the left side is not a substring of the right side.
- -SiteDeparted: Checks if the current document is still active or not.

When using these text comparison operators, you may optionally specify whether the comparison is to take into account the case of the strings being compared. If -I is specified, the comparison is case insensitive. If -S is specified, then the comparison is case sensitive. By default the Eq and NotEQ operators are not case sensitive, while the -In and -NotIn operators are case sensitive.

An operator is also supplied to directly query the application for a particular string:-Text: Evaluates to true if the specified text is found in the application windows of the application. For Internet Explorer application definitions, you can supply an optional -Frame argument, which restricts the command to look for the specified text in the current frame.

Numerical comparison operators supported Two numerical comparison operators are supported by the `While` command, Gt and Lt. The command evaluates to true if the left side is greater than or less than (respectively) the right side. This is a numerical comparison, so the left and right sides must be numbers.

An operator is supplied to check for the existence of a stored variable:

- -Exists: True if the specified variable exists.
- -NotExist: True if the specified variable does not exist.

NOTE: The following commands are not supported when using Google Chrome:

- `Highlight`
- `Type`
- `Click`
- `Submit`
- `SetCheckbox`
- `Select`
- `DumpPage`.
- `GetCheckBoxState`

7 Testing Application Definitions

Using the SecureLogin Test Application

To allow Administrators and other application definition writers to practice their application definition creation skills, the Password Test application is included in the software package. It is designed to replicate an application logon panel and supports the following processes:

- Initial log in
- Wrong password
- Password change

If you do not have the test application, contact Technical Support.

The following example, application definition for the Password Test application, further explains the SecureLogin application definition principles.

Example Application Definition for the Test Application

The application definition for the PSL Password Test Application (`PasswordTest.exe`) provides an example of a typical Windows application definition, including error handling and changing the password. Remember, the password for this application is hard-coded to single when the application is closed and restarted. This can cause confusion when setting strong password policies and changing passwords. You must also create a password policy called PwdTestPolicy, according to the password policy defined in this application definition. The password policy must require a minimum of 6 characters, but no complex rules, in order to use single as a password.

Here is the sample application definition in its entirety. Following this application definition is the explanation of what each section does.

```
# Set Password Policy
RestrictVariable $Password PwdTestPolicy
# ==== BeginSection: Log on ====
Dialog

   Class "#32770"

   Ctrl #1001

   Title "Log on"

EndDialog

SetPrompt "Username =====>"
Type $Username #1001
SetPrompt "Password =====>"
Type $Password #1002
SetPrompt "Domain =====>"
Type $Domain #1003
```

```
Click #1
SetPrompt "Please enter your user name and password to access Password Test.
SecureLogin will remember and automatically log you on in future.  IT Help Desk
x4532"
# ==== EndSection: Log on ====

# ==== BeginSection: Log on failure ====
Dialog

   Class "#32770"

   Title "Log on failure"

EndDialog

# Read the error message and set it as a temporary variable, then clear it
ReadText #65535 ?ErrorMessage
Click #2

# If log on failed, display the current stored Username and Password and prompt the
user to verify them, then retry log on
If "You have failed to log on." -In ?ErrorMessage
   DisplayVariables "Log on to Password Test failed. The password for this
application must be single when it first starts. IT Help Desk x4532"
# Press Alt>F and L to invoke the logon box so the user doesn't have to.

   Type -Raw "\Alt+F"

   Type -Raw "L"

   Type $Username

   Type $Password

   Type $Domain

EndIf

# ==== EndSection: Log on ====

# ==== Begin Section: Change Password ====
# Change Password Dialog Box
Dialog
Class "#32770"
Title "Change Password"
EndDialog

# Backup password, fill in the old user name and password, then start the change
password routine
Set ?PwdBackup $Password
Type $Username #1015
Type $Password #1004
ChangePassword ?NewPwd "Please enter a new password for this application."
Type ?NewPwd #1005
Type ?NewPwd #1006
Click #1


# Change password successful message
```

```
Dialog

   Class "#32770"

   Ctrl #65535 "You have changed your password successfully."

   Title "Change successful"

EndDialog

# Clear application owned message and accept new password
Click #2
Set $Password ?NewPwd
# ==== End Section: Change Password ====
```

Application Definition Explained

You can use the same application definition to show what function each section performs. `Dialog/EndDialog` blocks define a Windows dialog box. When the dialog box appears, SecureLogin detects that this dialog box is based on the information found within the dialog block. The `Dialog/EndDialog` block must contain enough information for the block to be unique, or the application definition runs when other dialog boxes owned by the same executable with the same information appear.

When SecureLogin detects that all the information between `Dialog` and `EndDialog` is contained in the dialog box on the screen (for example, the application login box, the change password box, or the failed logon box), it runs the application definition commands until it sees the next dialog statement or the end of the application definition, whichever is applicable. The order does not matter in Windows application definitions, because SecureLogin watches for all dialog boxes while the executable is running. Use a logical order for troubleshooting purposes.

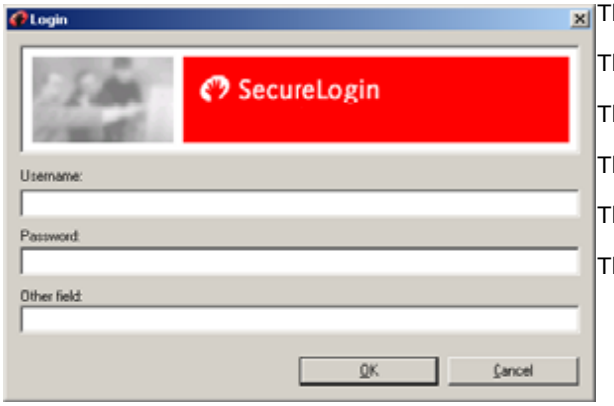
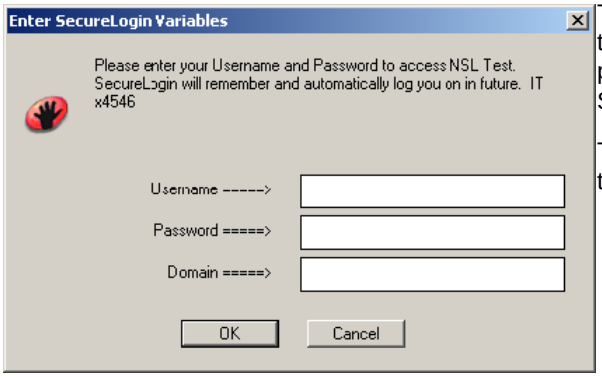
Dialog boxes

The following application definition example shows screen captures of the relevant dialog boxes. You can use the Window Finder tool to gather information about the title of the window, class names, dialog IDs, and so on. Use the wizard to automate the application definition creation.

Application definition section	Comments
`# Set Password PolicyRestrictVariable $Password PwdTestPolicy`	This restricts the $Password variable to comply with the Password Policy "PwdTestPolicy".
`# ==== BeginSection: Log on ====Dialog Class "#32770" Ctrl #1001 Title "Log on"EndDialog`	When `PasswordTest.exe` runs, SecureLogin watches for dialog boxes that appear and match the information defined between the `Dialog/EndDialog` commands. You can specify all values, or a few, as long as the information specified is unique to that dialog box.

Application definition section	Comments

`SetPrompt "Username =====>"`
`"Type $Username #1001`
`SetPrompt "Password =====>"`
`"Type $Password #1002`
`SetPrompt "Domain =====>"`
`Type $Domain #1003`
`Click #1`
`SetPrompt "Please enter your Username and Password to access NSL Test. SecureLogin will remember and automatically log you on in future. IT Helpdesk x4546"`
`# ==== EndSection: Log on ====`
	Type the stored ($) Username variable into #1001, and so on. `SetPrompt` is used to customize the window the user sees when there are no credentials stored.
	
	When the user first runs an application that is newly enabled for single sign-on, SecureLogin prompts for their login credentials, and stores and remembers them for future login attempts.

	The title is Log In. The Class is #32770. The **Username** field is Control ID #1001. The **Password** field is Control ID #1002. The **Other field** is Control ID #1003. The **OK** button is Control ID #1.
	This dialog box is only displayed the first time the application definition is run by a user. It prompts the user to enter credentials for SecureLogin to store. The `SetPrompt` command is used throughout the example application.
	This is the login failure dialog box. The title is Login Failure. The class is #32770. The **OK** button is Control ID #2. The error message is Control ID #65535

Application definition section	Comments
	This is the Change Password dialog box. The **Username** field is Control ID #1015. The **Old Password** field is Control ID #1004. The **New Password** field is Control ID #1005. The **Confirm New Password** field is Control ID #1006. The **OK** button is Control ID #1.
	The `ChangePassword` command is used in the example application definition to display a dialog box for the user to enter a new password. The dialog box is customized to provide more information for the user.

8 Reference Commands and Keys

Windows Keyboard Functions

The following reference tables list the Windows keyboard functions. You can use these functions in conjunction with the Type command by referencing the appropriate keyboard code.

Do not type quotation marks before and after the keys. In this case the keys are taken literally, as shown in the following table.

Table 8-1 *Typing Keys*

For this command	Type
Alt+Print Screen	\Alt+|44
Shift+Home	\Shift+|36
Shift+End	\Shift+|35

For more information about the Type command, see “Type” on page 175.

Table 8-2 *Windows Keyboard Functions*

Function	Decimal	Comment
Left mouse button	1	
Right mouse button	2	
CTRL-Break	3	
Middle mouse button	4	
X1 mouse button	5	
X2 mouse button	6	
Backspace	8	
Tab	9	
Clear	12	5 on the keypad
Enter	13	
Shift	16	
Ctrl	17	
Alt	18	

Function	Decimal	Comment
Pause	19	
Cap Lock	20	
Escape	27	
Space	32	
PageUp	33	
PageDown	34	
End	35	
Home	36	
Left-arrow	37	
Up-arrow	38	
Right-arrow	39	
Down	40	
Select	41	
Execute	43	
Print	44	
Insert	45	
Delete	46	
Help Key	47	
0	48	
1	49	
2	50	
3	51	
4	52	
5	53	
6	54	
7	55	
8	56	
9	57	
A	65	
B	66	
C	67	
D	68	
E	69	

Function	Decimal	Comment
F	70	
G	71	
H	72	
I	73	
J	74	
K	75	
L	76	
M	77	
N	78	
O	79	
P	80	
Q	81	
R	82	
S	83	
T	84	
U	85	
V	86	
W	87	
X	88	
Y	89	
Z	90	
Left Windows Key	91	
Right Windows Key	92	
Application Key	93	
Sleep Key	94	
Keypad 0	96	
Keypad 1	97	
Keypad 2	98	
Keypad 3	99	
Keypad 4	100	
Keypad 5	101	
Keypad 6	102	
Keypad 7	103	

Function	Decimal	Comment
Keypad 8	104	
Keypad 9	105	
Keypad asterisk (*)	106	
Keypad plus sign (+)	107	
Keypad separator	108	
Keypad minus sign (-)	109	
Keypad period (.)	110	
Keypad slash mark (/)	111	
F1 key	112	
F2 key	113	
F3 key	114	
F4 key	115	
F5 key	116	
F6 key	117	
F7 key	118	
F8 key	119	
F9 key	120	
F10 key	121	
F11 key	122	
F12 key	123	
F13 key	124	
F14 key	125	
F15 key	126	
F16 key	127	
F17 key	128	
F18 key	129	
F19 key	130	
F20 key	131	
F21 key	132	
F22 key	133	
F23 key	134	
F24 key	135	
Num Lock key	144	

Function	Decimal	Comment
Scroll Lock	145	
Left Shift	160	
Right Shift	161	
Left Control	162	
Right Control	163	
Left Menu	164	
Right Menu	165	
Browser Back key	166	Applies to Windows 2000 +
Browser Forward key	167	Applies to Windows 2000 +
Browser Refresh key	168	Applies to Windows 2000 +
Browser Stop key	169	Applies to Windows 2000 +
Browser Search key	170	Applies to Windows 2000 +
Browser Favorites key	171	Applies to Windows 2000 +
Browser Start and Home key	172	Applies to Windows 2000 +
Volume Mute key	173	Applies to Windows 2000 +
Volume Down key	174	Applies to Windows 2000 +
Volume Up key	175	Applies to Windows 2000 +
CD Next Track key	176	Applies to Windows 2000 +
CD Previous Track key	177	Applies to Windows 2000 +
CD Stop Media key	178	Applies to Windows 2000 +
CD Play/Pause key	179	Applies to Windows 2000 +
Launch Mail key	180	Applies to Windows 2000 +
Media Select key	181	Applies to Windows 2000 +
Start Application 1 key	182	Applies to Windows 2000 +
Start Application 2 key	183	Applies to Windows 2000 +
;	186	Semi Colon/Colon
=	187	Equals/Plus Key
,	188	Comma/Less Than
-	189	Minus/Underscore
.	190	Period/Greater Than
/	191	Slash/Question Mark
`	192	Single Open Quote/Tilde
[	219	Left Square/Curley Bracket

Function	Decimal	Comment
\	220	Back slash/Pipe
]	221	Right Square/Curley Bracket
'	222	Single Close Quote Double Quote
Play Key	250	
Zoom Key	251	

Terminal Emulator Commands

The following table lists the terminal commands in terminal emulator application definitions.

The Type Command	Meaning	The Type Command	Meaning
@B	Left Tab	@A@C	Test
@C	Clear	@A@D	Word Delete
@D	Delete	@A@E	Field Exit
@E	Enter	@A@F	Erase Input
@F	Erase EOF	@A@H	System Request
@H	Help	@A@I	Insert Toggle
@I	Insert	@A@J	Cursor Select
@J	Jump (Set Focus)	@A@L	Cursor Left Fast
@L	Cursor Left	@A@Q	Attention
@N	New Line	@A@R	Device Cancel (Cancels Print Presentation Space)
@O	Space	@A@T	Print Presentation Space
@P	Print	@A@U	Cursor Up Fast
@R	Reset	@A@V	Cursor Down Fast
@T	Right Tab	@A@Z	Cursor Right Fast
@U	Cursor Up	@A@9	Reverse Video
@V	Cursor Down	@A@b	Underscore
@X*	DBCS (Reserved)	@A@c	Reset Reverse Video
@Y	Caps Lock (No action)	@A@d	Red
@Z	Cursor Right	@A@e	Pink
@0	Home	@A@f	Green
@1	PF1/F1	@A@g	Yellow
@2	PF2/F2	@A@h	Blue

The Type Command	Meaning	The Type Command	Meaning
@3	PF3/F3	@A@i	Turquoise
@4	PF4/F4	@A@l	Reset Host Colors
@5	PF5/F5	@A@j	White
@6	PF6/F6	@A@t	Print (Personal Computer)
@7	PF7/F7	@A@y	Forward Word Tab
@8	PF8/F8	@A@z	Backward Word Tab
@9	PF9/F9	@A@	- Field -
@a	PF10/F10	@A@<	Record Backspace
@b	PF11/F11	@A@	+ Field +
@c	PF12/F12	@S@x	Dup
@d	PF13	@S@E	Print Presentation Space or Host
@e	PF14	@S@y	Field Mark
@f	PF15	@X@c	Split Vertical Bar (¦)
@g	PF16	@X@7	Forward Character
@h	PF17	@X@6	Display Attribute
@i	PF18	@X@5	Generate SO/SI
@j	PF19	@X@1	Display SO/SI
@k	PF20	@M@0	VT Numeric Pad 0
@l	PF21	@M@1	VT Numeric Pad 1
@m	PF22	@M@2	VT Numeric Pad 2
@n	PF23	@m@3	VT Numeric Pad 3
@o	PF24	@M@4	VT Numeric Pad 4
@q	End	@M@5	VT Numeric Pad 5
@s	ScrLk (No action)	@M@6	VT Numeric Pad 6
@t	Num Lock (No action)	@M@7	VT Numeric Pad 7
@u	Page Up	@M@8	VT Numeric Pad 8
@v	Page Down	@M@9	VT Numeric Pad 9
@x	PA1	@M@-	VT Numeric Pad
@y	PA2	@M@,	VT Numeric Pad
@z	PA3	@M@.	VT Numeric Pad
@M@h	VT Hold Screen	@M@e	VT Numeric Pad Enter
@M@N	Control Code SO	@M@f	VT Edit Find

The Type Command	Meaning	The Type Command	Meaning
@M@M	Control Code CR	@M@i	VT Edit Insert
@M@L	Control Code FF	@M@r	VT Edit Remove
@M@K	Control Code VT	@M@s	VT Edit Select
@M@J	Control Code LF	@M@p	VT Edit Previous Screen
@M@I	Control Code HT	@M@n	VT Edit Next Screen
@M@H	Control Code BS	@M@a	VT PF1
@M@G	Control Code BEL	@M@b	VT PF2
@M@F	Control Code ACK	@M@c	VT PF3
@M@(space)	Control Code NUL	@M@d	VT PF4
@M@E	Control Code ENQ	@M@O	ControlCode S1
@M@D	Control Code EOT	@M@Q	ControlCode DC1
@M@C	Control Code ETX	@M@P	ControlCode DLE
@M@B	Control Code STX	@M@A	ControlCode SOH

9 Application Definition Commands for SNMP Alerts

SecureLogin produces Simple Network Management Protocol (SNMP) traps for use with SNMP based network monitoring software. One or more traps can be configured within a single application definition script to indicate errors or other status related information.

NOTE

- Copy the `LIBSNMP.DLL` file from the SecureLogin 8 CD located at `<CD_ROOT>\SecureLogin\Tools\Unsupported\SNMP` to the `Windows\System32` folder.
- You can also find the following files in the same location as the `LIBSNMP.DLL` file.
  - `SecureLogin.mib`: mib file for SNMP management console.
  - `slsnmp.exe`: Executable to send SNMP trap.

Creating an SNMP Alert

In order to produce an SNMP alert, place the following command in the application definition where you would like to create the alert:

NOTE: The required `slsnmp.exe` file is not copied to the machine during installation. It must to be copied from the SecureLogin 8 CD to the `\Program Files\NetIQ\SecureLogin` folder.

```
Run C:\Program Files\NetIQ\SecureLogin\slsnmp.exe <Community Name>
<Host IP Address> <Text>
```

Where:

- *<Community Name>* is the case-sensitive community name to which this computer sends trap messages.
- *<Host IP Address>* is the IP address of the SNMP host.
- *<Text>* is the text displayed as the message at the host.

Example

The following is an example application definition:

```
Dialog
  Class #32770
  Title "Incorrect Password"
EndDialog
Run "C:\Program Files\NetIQ\SecureLogin\Slsnmp.exe" SNMPCOmmunity1 192.168.156.23
"Incorrect password in finance system.
MessageBox "You have entered an incorrect password. The administrator has been
notified. Please restart the application and try again."
 "PasswordText.exe"
```