Novell SecureLogin

6.0 SP1

www.novell.com

APPLICATION DEFINITION GUIDE

October 13, 2006

Legal Notices

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.
www.novell.com

Online Documentation: To access the online documentation for this and other Novell products, and to get updates, see www.novell.com/documentation.

Contents

About This Guide 9

1 Command Quick Reference 11

2 Application Definition Language Overview 19

2.1 Using Application Definitions 19
2.2 Advantages of Using Application Definitions 19
2.3 Defining Single Sign-On Enabled Applications 20
2.4 Corporate Definitions 20
2.5 What is an Application Definition? 20
2.5.1 Using with Dialog Specifier Commands 20
2.5.2 Capability to Read from and Write to Variables 21

3 Managing Application Definitions 23

3.1 Application Definition Checklist 23
3.2 Exporting and Importing Predefined Applications and Application Definitions. 23
3.2.1 Export Using an XML File 24
3.2.2 Importing in XML Format 26
3.2.3 Modifying in the Personal Management Utility 28
3.2.4 Building an Application Definition in the Personal Management Utility 30
3.3 Windows Application Definition Tools 32
3.3.1 Finding Application Details With Window Finder 32
3.3.2 Finding Application Details with the Login Watcher 34
3.4 Application Definition Elements 36

4 Application Definition Variables 39

4.1 Types of Variables 39
4.1.1 Using a Variable to Change the Default Platform 39
4.1.2 Directory Attribute Variables 40
4.1.3 Stored Variables 40
4.1.4 Runtime Variables 41
4.1.5 Passticket Variables 42
4.2 SecureLogin Supported Variables 42
4.3 Application Definition Conventions 43
4.3.1 Symbols Used 44
4.3.2 Capitalization 44
4.3.3 Comments 44
4.3.4 Switches 44
4.3.5 Variables 45
4.3.6 Indent Sections 45
4.3.7 Blank Line Between Sections 45
4.3.8 Quotation Marks 46
4.3.9 Password Policy Names 46

5 Command Reference 49

5.1 Command Reference Conventions 49

5.1.1 Command Information 49
5.1.2 Web Wizard Application Definition Conventions 50
5.1.3 Site Matching 51
5.1.4 Form/Field/Option Matching 51
5.1.5 Form/Field/Option ID’s 52
5.1.6 Audit Integration 52
5.1.7 One Time Passwords 52
5.2 Commands 53
5.2.1 Regular Expressions 55
5.2.2 AAVerify 55
5.2.3 ADD 58
5.2.4 Attribute 59
5.2.5 AuditEvent 60
5.2.6 BeginSplashScreen/EndSplashScreen 60
5.2.7 Break 61
5.2.8 BooleanInput 63
5.2.9 Call 63
5.2.10 ChangePassword 64
5.2.11 Class 66
5.2.12 ClearPlat 67
5.2.13 ClearSite 70
5.2.14 Click 71
5.2.15 ConvertTime 73
5.2.16 Ctrl 74
5.2.17 DebugPrint 75
5.2.18 Decrement 76
5.2.19 Delay 77
5.2.20 Dialog/EndDialog 78
5.2.21 DisplayVariables 79
5.2.22 Divide 81
5.2.23 DumpPage 82
5.2.24 EndScript 82
5.2.25 Event 83
5.2.26 Event Specifiers 83
5.2.27 FocusInput 83
5.2.28 GenerateOTP 84
5.2.29 GetCheckBoxState 86
5.2.30 GetCommandLine 87
5.2.31 GetEnv 88
5.2.32 GetIni 88
5.2.33 GetMD5 89
5.2.34 GetReg 90
5.2.35 GetSessionName 90
5.2.36 GetText 91
5.2.37 GetURL 91
5.2.38 GoToURL 92
5.2.39 If/Else/Endif 93
5.2.40 Include 96
5.2.41 Increment/Decrement 97
5.2.42 KillApp 98
5.2.43 Local 99
5.2.44 MatchDomain 100
5.2.45 MatchForm 101
5.2.46 MatchField 103
5.2.47 MatchForm 104
5.2.48 MatchOption 106
5.2.49 MatchReferer 107
5.2.50 MatchURL 108

5.2.51 MessageBox . . . 109
5.2.52 Multiply . . . 112
5.2.53 OnException/ClearException . . . 113
5.2.54 Parent/EndParent . . . 115
5.2.55 PickListAdd . . . 117
5.2.56 PickListDisplay . . . 118
5.2.57 PositionCharacter . . . 119
5.2.58 PressInput . . . 120
5.2.59 ReadText . . . 120
5.2.60 RegSplit . . . 123
5.2.61 ReLoadPlat . . . 124
5.2.62 Repeat/EndRepeat . . . 126
5.2.63 RestrictVariable . . . 128
5.2.64 Run . . . 130
5.2.65 SelectListBoxItem . . . 131
5.2.66 SendKey . . . 132
5.2.67 Set . . . 133
5.2.68 SetCheckBox . . . 134
5.2.69 SetCursor . . . 135
5.2.70 SetFocus . . . 136
5.2.71 SetPlat . . . 137
5.2.72 SetPrompt . . . 139
5.2.73 Site/EndSite . . . 141
5.2.74 StrCat . . . 142
5.2.75 StrLength . . . 143
5.2.76 StrLower . . . 144
5.2.77 StrUpper . . . 145
5.2.78 Sub/EndSub . . . 146
5.2.79 Submit . . . 147
5.2.80 Subtract . . . 148
5.2.81 Tag/EndTag . . . 150
5.2.82 TextInput . . . 150
5.2.83 Title . . . 151
5.2.84 Type . . . 152
5.2.85 Sending Keyboard Commands Using `Type` . . . 155
5.2.86 WaitForFocus . . . 156
5.2.87 WaitForText . . . 157

6 Testing Application Definitions 159

6.1 Using the SecureLogin Test Application . . . 159
6.1.1 Example Application Definition for the Test Application . . . 159
6.1.2 Application Definition Explained . . . 161
6.1.3 Dialog Boxes . . . 161

7 Reference Commands and Keys 165

7.1 Windows Keyboard Functions . . . 165
7.1.1 Example for Typing Keys . . . 165
7.1.2 Table of Windows Keyboard Functions . . . 165
7.1.3 Terminal Emulator Command Reference . . . 170

8 Application Definition Commands for SNMP Alerts 173

8.1 Create the SNMP Alert . . . 173
8.2 Example . . . 173

A Documentation Updates **175**

A.1 December 12, 2006 175
A.2 October 13, 2006 175

About This Guide

This document contains information on the following:

- Chapter 1, "Command Quick Reference," on page 11
- Chapter 2, "Application Definition Language Overview," on page 19
- Chapter 3, "Managing Application Definitions," on page 23
- Chapter 4, "Application Definition Variables," on page 39
- Chapter 5, "Command Reference," on page 49
- Chapter 6, "Testing Application Definitions," on page 159
- Chapter 7, "Reference Commands and Keys," on page 165
- Chapter 8, "Application Definition Commands for SNMP Alerts," on page 173

Feedback

We want to hear your comments and suggestions about this manual and the other documentation included with this product. Please use the User Comments feature of each page of the online documentation, or go to www.novell.com/documentation/feedback.html and enter your comments there.

Documentation Updates

This is the Support Pack 1 (SP1) release for Novell SecureLogin 6.0. The version for this support pack in the product is, 6.0.103.

For the most recent version of the *SecureLogin 6.0 SP1 Application Definition Guide*, visit the Novell Documentation Web site (http://www.novell.com/documentation/securelogin60).

Additional Documentation

This *Application Definition Guide* is a part of documentation set for SecureLogin 6.0 SP1. Other documents include:

- *Novell SecureLogin 6.0 SP1 Overview*
- *Novell SecureLogin 6.0.SP1 Administration Guide*
- *Novell SecureLogin 6.0 SP1 Installation Guide*
- *Novell SecureLogin 6.0 SP1 User Guide*
- *Novell SecureLogin 6.0 SP1 Citrix and Terminal Services Guide*
- *Novell SecureLogin 6.0 SP1 Congifuration Guide for Terminal Emulation*

Documentation Conventions

In Novell documentation, a greater-than symbol (>) is used to separate actions within a step and items in a cross-reference path.

Command Quick Reference

1

Table 1-1 Command Quick Reference Table

Command	See
`“#”`	Use the hash symbol to define a line of text as a comment field. Comment fields are used to leave notes. For more information, see Section 3.4, “Application Definition Elements,” on page 36.
`“ ”`	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 more information, seeSection 3.4, “Application Definition Elements,” on page 36.
`“$”`	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 Section 3.4, “Application Definition Elements,” on page 36.
`“?”`	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 Section 3.4, “Application Definition Elements,” on page 36.
`“%”`	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 Section 3.4, “Application Definition Elements,” on page 36.
`“!”`	Use the exclamation mark to define the use of a passticket. A passticket is a one-time password (OTP) that is generated using a combination of an encryption key, encryption offset, and the current time. For more information, see Section 3.4, “Application Definition Elements,” on page 36.
`“\”`	Use the backslash with the `Type` and `Send Key` commands to specify the use of a special function. For more information, see Section 3.4, “Application Definition Elements,” on page 36.
`“@”`	Use this symbol in a similar function to the backslash symbol, except its use is limited to HLLAPI enabled emulators. For more information, see Section 3.4, “Application Definition Elements,” on page 36.
`“-”`	Use the hyphen as a switch within several commands, such as `If` and `Type`. For more information, see Section 3.4, “Application Definition Elements,” on page 36.
`“AAVerify”`	Use `AAVerify` with SecureLogin Advanced Authentication or Novell NMAS to verify the user. It is typically used before the application Username and Password are retrieved and entered into the logon box.For more information, see Section 5.2.2, “AAVerify,” on page 55.
`“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 ouput of another variable, or one of the original numbers. For more information, see Section 5.2.3, “ADD,” on page 58.

Command	See
`“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 Section 5.2.4, “Attribute,” on page 59.
`“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 Section 5.2.5, “AuditEvent,” on page 60
`“BeginSplashScreen/EndSplashScreen”`	Use to display a Novell 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 Section 5.2.6, “BeginSplashScreen/EndSplashScreen,” on page 60.
`“BooleanInput”`	Use `BooleanInput` within a `Site` block to set the state of a Boolean field (either a checkbox or radio button). For more information see, Section 5.2.8, “BooleanInput,” on page 63
`“Break”`	Use `Break` within the `Repeat/EndRepeat` commands to break out of a repeat loop. For more information, see Section 5.2.7, “Break,” on page 61.
`“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 Section 5.2.9, “Call,” on page 63.
`“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 Section 5.2.10, “ChangePassword,” on page 64.
`“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 Section 5.2.11, “Class,” on page 66.
`“ClearPlat”`	Use to reset the last chosen platform, causing subsequent calls to ReLoadPlat to do nothing. For more information, see Section 5.2.12, “ClearPlat,” on page 67.
`“ClearSite”`	Use within a `Site` block to clear the ‘matched’ status for a given site. For more information, see Section 5.2.13, “ClearSite,” on page 70
`“Click”`	When used with windows applications, the `Click` command sends a click instruction to the specified `<#Ctrl-ID>`. For more information, see Section 5.2.14, “Click,” on page 71.
`“ConvertTime”`	Use to convert a numeric time value, for example, `?CurrTime(system)`, into a legible format and stores it in `<String Time>`. For more information, see Section 5.2.15, “ConvertTime,” on page 73.

Command	See
`“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 Section 5.2.16, “Ctrl,” on page 74.
`“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 Section 5.2.19, “Delay,” on page 77.
`“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, Title, and so on). For more information, see Section 5.2.20, “Dialog/EndDialog,” on page 78.
`“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 Section 5.2.21, “DisplayVariables,” on page 79.
`“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 outputted to another variable, or to one of the original numbers. For more information, see Section 5.2.22, “Divide,” on page 81.
`“DumpPage”`	Use the `DumpPage` command to provide information about the current Web page. Use for debugging Web page Application Definitions. For more information, see Section 5.2.23, “DumpPage,” on page 82.
`“EndScript”`	Use the `EndScript` command to immediately terminate execution of the Application Definition. For more information, see Section 5.2.24, “EndScript,” on page 82.
`“Event”`	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. For more information, see Section 5.2.25, “Event,” on page 83.
`“FocusInput”`	Use within a `Site` Block to focus on an input field based on the Boolean value of `“focus”`. For more information see, Section 5.2.27, “FocusInput,” on page 83.
`“GenerateOTP”`	Used to generate a one time password (OTP) is an authentication method in lieu of a traditional fixed and static password. The OTP is a hard token generated by the Vasco Digipass, RSA SecureID or ActivIdentity* Token and Mini Token products or may be produced bya soft token generator funtionality embedded in SecureLogin. For more information, see Section 5.2.28, “GenerateOTP,” on page 84.
`“GetCheckBoxState”`	Use the `GetCheckBoxState` command to return the current state of the specified checkbooks. For more information, see Section 5.2.29, “GetCheckBoxState,” on page 86.
`“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 Section 5.2.30, “GetCommandLine,” on page 87.

Command	See
`“GetEnv”`	Use the `GetEnv` command to read the value of an environment variable and saves it in the specified `<variable>`. For more information, see Section 5.2.31, “GetEnv,” on page 88.
`“GetIni”`	Use the `GetIni` command to read data from INI file. For more information, see Section 5.2.32, “GetIni,” on page 88.
`“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 Section 5.2.33, “GetMD5,” on page 89
`“GetReg”`	Use the `GetReg` command to read data from the registry and save it in the specified `<variable>`. For more information, see Section 5.2.34, “GetReg,” on page 90.
`“GetSessionName”`	Use the `GetSessionName` to find the current HLLAPI session name that is used to connect and returns it to the specified variable. For more information, see Section 5.2.35, “GetSessionName,” on page 90.
`“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 Section 5.2.36, “GetText,” on page 91.
`“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 Section 5.2.37, “GetURL,” on page 91.
`“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 Section 5.2.38, “GoToURL,” on page 92.
`“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 Section 5.2.39, “If/Else/Endif,” on page 93.
`“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 comprise commands supported by the calling application. For more information, see Section 5.2.40, “Include,” on page 96.
`“Increment/Decrement”`	Use the `Increment/Decrement` command to add or subtract from a specified variable. For example, you can use the increment and decrement to count the number of passes a particular Application Definition has made. For more information, see Section 5.2.41, “Increment/Decrement,” on page 97.
`“KillApp”`	Use to terminate an application. For more information, see Section 5.2.42, “KillApp,” on page 98.
`“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 Section 5.2.43, “Local,” on page 99.

Command	See
`“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 Section 5.2.44, “MatchDomain,” on page 100.
`“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 will fail to match. For more information, see Section 5.2.44, “MatchDomain,” on page 100.
`“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 will fail to match. For more information, see Section 5.2.45, “MatchForm,” on page 101.
`“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 Section 5.2.48, “MatchOption,” on page 106.
`“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 Section 5.2.50, “MatchURL,” on page 108.
`“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 Section 5.2.51, “MessageBox,” on page 109.
`“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 Section 5.2.52, “Multiply,” on page 112.
`“OnException/ ClearException”`	Use the `OnException` command to detect when certain conditions are met. Currently, this is when Cancel is pressed 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 Section 5.2.53, “OnException/ ClearException,” on page 113.
`“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 Section 5.2.54, “Parent/EndParent,” on page 115.
`“PickListAdd”`	Use the `PickList` command to allow users with multiple accounts for a particular system to choose the account to which they will log on. For more information, see Section 5.2.55, “PickListAdd,” on page 117.
`“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 Section 5.2.56, “PickListDisplay,” on page 118.
`“PositionCharacter”`	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. For more information, see Section 5.2.57, “PositionCharacter,” on page 119.
`“PressInput”`	Used within a Site block to simulate a keyboard enter event. For more information, see Section 5.2.58, “PressInput,” on page 120.

Command	See
`“ReadText”`	Use the `ReadText` command to run in both Windows and Terminal Launcher 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. For more information, see Section 5.2.59, “ReadText,” on page 120.
`“RegSplit”`	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. For more information, see Section 5.2.60, “RegSplit,” on page 123.
`“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 Section 5.2.61, “ReLoadPlat,” on page 124.
`“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 Section 5.2.62, “Repeat/EndRepeat,” on page 126.
`“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 Section 5.2.63, “RestrictVariable,” on page 128.
`“Run”`	Use the `Run` command to launch the program specified in `<Command>` with the specified optional [`<Arg1> [<Arg2>] …]` arguments. For more information, see Section 5.2.64, “Run,” on page 130.
`“SelectListBoxItem”`	Use the `SelectListBoxItem` command to select entries from a list box. For more information, see Section 5.2.65, “SelectListBoxItem,” on page 131.
`“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 Section 5.2.66, “SendKey,” on page 132.
`“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 Section 5.2.67, “Set,” on page 133.
`“SetCheckBox”`	Use the `SetCheckBox` command to select or clear a check box. For more information, see Section 5.2.68, “SetCheckBox,” on page 134.
`“SetCursor”`	Use the `SetCursor` command to set the cursor to a specified `<ScreenPosition>` or `<X Co-ordinate> <Y Co-ordinate>`. For more information, see Section 5.2.69, “SetCursor,” on page 135.
`“SetFocus”`	Use the `SetFocus` command to set the keyboard focus to a specified <#Ctrl-ID>.For more information, see Section 5.2.70, “SetFocus,” on page 136.
`“SetPlat”`	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. For more information, see Section 5.2.71, “SetPlat,” on page 137.

Command	See
`“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 Section 5.2.72, “SetPrompt,” on page 139.
`“Site/ Endsite”`	`Site/Endsite` begins and ends an Application Definition, in place of `Dialog/ EndDialog`. `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. For more information, see Section 5.2.73, “Site/EndSite,” on page 141
`“StrCat”`	Use the `StrCat` command to append the second data string to the first data string. For example, `StrCat ?Result "SecureRemote " "$Username"`. For more information, see Section 5.2.74, “StrCat,” on page 142.
`“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 Section 5.2.75, “StrLength,” on page 143.
`“StrLower”`	Use the `StrLower` command to modify a variable so that all the characters are lower case. For more information, see Section 5.2.76, “StrLower,” on page 144.
`“StrUpper”`	Use the `StrUpper` command to modify a variable so that all the characters are upper case. For more information, see Section 5.2.77, “StrUpper,” on page 145.
`“Sub/EndSub”`	Use the `Sub/EndSub` commands around a block of lines within an Application Definition to denote a subroutine. For more information, see Section 5.2.78, “Sub/ EndSub,” on page 146.
`“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 Section 5.2.79, “Submit,” on page 147.
`“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 the Add, Divide, and Multiply. For more information, see Section 5.2.80, “Subtract,” on page 148.
`“Tag/EndTag”`	Use the `Tag/EndTag` commands to find HTML tags. For more information, see Section 5.2.81, “Tag/EndTag,” on page 150.
`“TextInput”`	Use within a Site block to input text into a special field. For more information, see Section 5.2.82, “TextInput,” on page 150
`“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 Section 5.2.83, “Title,” on page 151.

Command	See
"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 Section 5.2.84, "Type," on page 152.
"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. Once focus is received, the Application Definition continues. For more information, see Section 5.2.86, "WaitForFocus," on page 156.
"WaitForText"	Use the `WaitForText` command so the Terminal Launcher waits for the specified `<text>` to display before continuing. This command allows the user to wait for particular text to display before continuing. For example, waiting for a username field to display before attempting to type a username. For more information, see Section 5.2.87, "WaitForText," on page 157.

Application Definition Language Overview

2

The capability of Novell® SecureLogin to create proprietary Application Definitions is a powerful feature. This Application Definition command language facilitates SecureLogin 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:

- Section 2.1, "Using Application Definitions," on page 19
- Section 2.2, "Advantages of Using Application Definitions," on page 19
- Section 2.3, "Defining Single Sign-On Enabled Applications," on page 20
- Section 2.4, "Corporate Definitions," on page 20
- Section 2.5, "What is an Application Definition?," on page 20

2.1 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 log in 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.

2.2 Advantages of Using Application Definitions

SecureLogin Application Definitions provide the following advantages:

- Enables you to define single sign-on methods for almost any Windows*, Mainframe, Internet, Intranet, Terminal Server, or UNIX application.
- There is no need to install backend modules on your application servers.
- Provides 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.

- SecureLogin data such as user credentials is stored and protected in the directory.
- On startup, SecureLogin performs the following tasks:
    - Locate these objects in the directory.
    - Cache their encrypted contents in memory (and optionally on disk) for later use by the workstation's SecureLogin agent.

2.3 Defining Single Sign-On Enabled Applications

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

- Full control on deciding which applications need to be single sign-on enabled.
- The ability to update the entire directory database with a new application log on Application Definition by updating a single object.

2.4 Corporate Definitions

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.

2.5 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, in the case of 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 executable file or application may produce. In this manner 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, backend database is down, password expiry, and so on.

SecureLogin follows 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.

2.5.1 Using with Dialog Specifier Commands

It is possible to assign individual sections of an Application Definition to the different windows an executable file may produce, with the use of Dialog Specifier commands. This allows the login dialog box for example to be treated differently from the Error Message box and so on.

Many of the SecureLogin commands such as `Repeat` and `Dialog`, have one or two commands that are used to close them.

2.5.2 Capability to Read from and Write to Variables

Application Definition commands have the capability to 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 is not only able to write information to the screen, but is also able to read from it with the use of commands such as `ReadText`. This can be used to extract usernames, domains in use, error messages and other useful information. Variable Manipulator commands can then be used to perform calculations, break apart information, and join it back again.

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

NOTE: When writing an Application Definition that requires a "-" (dash) in the command syntax, make sure you use a short ASCII dash (en dash) and not an extended dash (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, then an em dash (—) is created. This may have implications for definitions that are created in or have been copied and pasted from Microsoft Word documents.

Managing Application Definitions 3

Application definitions are generally imported, built, or modified in the Management Utility of Novell® 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.

This section contains the following information:

- Section 3.1, "Application Definition Checklist," on page 23
- Section 3.2, "Exporting and Importing Predefined Applications and Application Definitions," on page 23
- Section 3.3, "Windows Application Definition Tools," on page 32
- Section 3.4, "Application Definition Elements," on page 36

3.1 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 correct username or password.
- Entering incorrect username or password.
- User canceling a log on.
- Exceeding maximum password retries.
- User changing own password.
- User attempting to change to illegal password.
- User canceling password change.
- Administrator changing user password.
- User password expiring.
- Account locked out.
- User attempting a simultaneous log on.

3.2 Exporting and Importing Predefined Applications and Application Definitions

SecureLogin provides export and import 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:

- Section 3.2.1, "Export Using an XML File," on page 24
- Section 3.2.2, "Importing in XML Format," on page 26

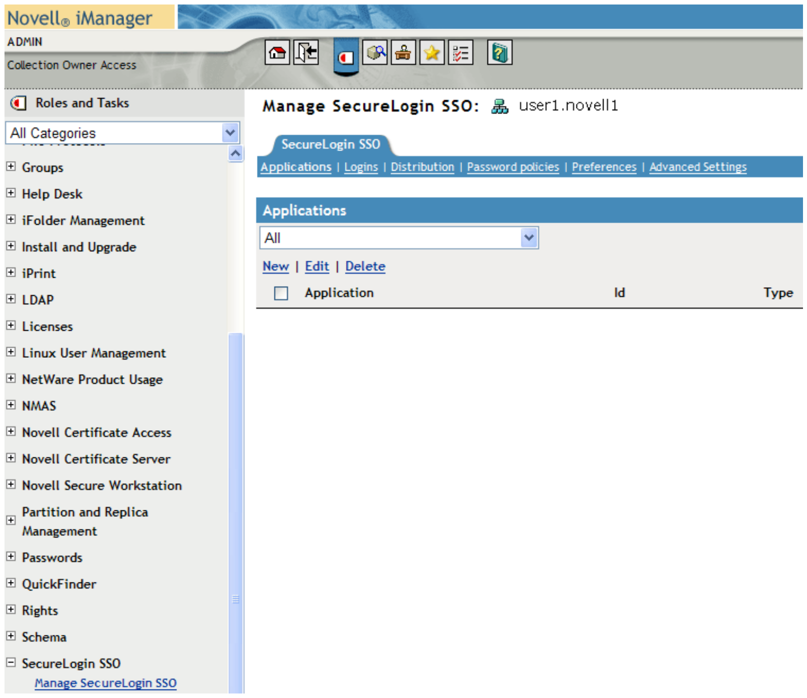
* Section 3.2.3, "Modifying in the Personal Management Utility," on page 28
* Section 3.2.4, "Building an Application Definition in the Personal Management Utility," on page 30

3.2.1 Export Using an XML File

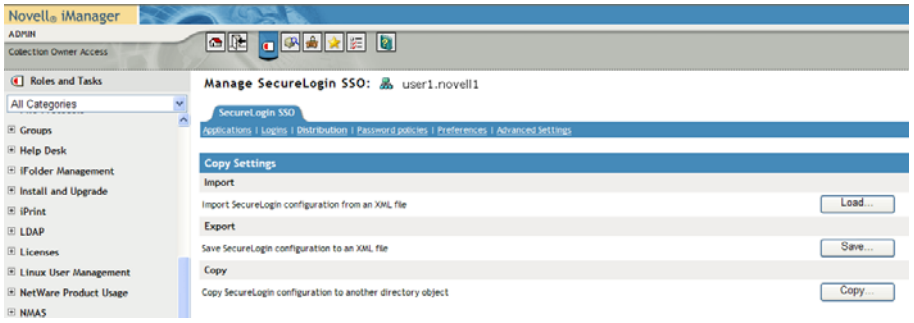
You can manage application definitions for applications by using the iManager, the Microsoft Management Console (MMC) or the SecureLogin Manager (`slmanager.exe`).

To export SecureLogin data using an XML file through iManager, do the following:

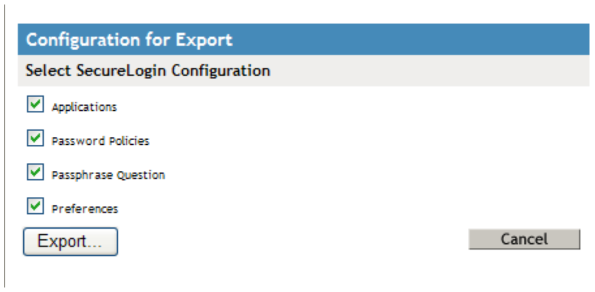
1. Log in to iManager.
2. In the object field, specify your object name, then click *OK*.



3. Click D*istribution*. The distribution details are displayed.


Novell® iManager
ADMIN
Collection Owner Access
Roles and Tasks
All Categories
Groups
Help Desk
iFolder Management
Install and Upgrade
iPrint
LDAP
Licenses
Linux User Management
NetWare Product Usage
NMAS
Manage SecureLogin SSO: user1.novell1
SecureLogin SSO
Applications | Logins | Distribution | Password policies | Preferences | Advanced Settings
Copy Settings
Import
Import SecureLogin configuration from an XML file
Load...
Export
Save SecureLogin configuration to an XML file
Save...
Copy
Copy SecureLogin configuration to another directory object
Copy...

4 Click *Save*. The save dialog box is displayed.



5 In the *Select Securelogin Configuration* section, do the following:

5a Select *Applications*.

5b Clear the other *Select Securelogin Configuration* check boxes.

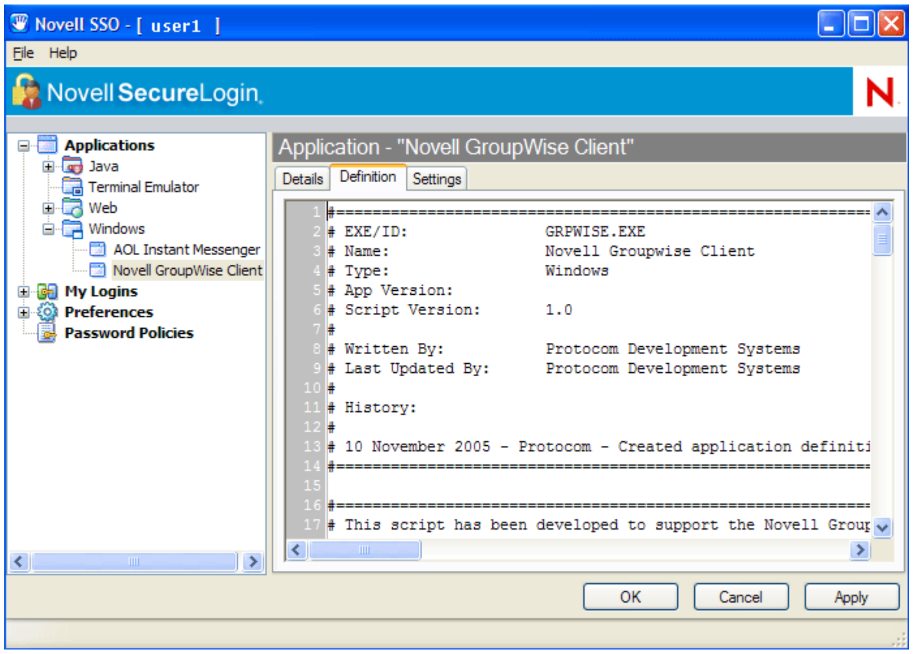
5c Select *Not Encrypted* in the *Select File Protection* section.

NOTE: This option is not present for iManager.

---

6 Click *Export*. The Do you want to open or save dialog box is displayed.

7 Click *Save*. The save dialog box is displayed.



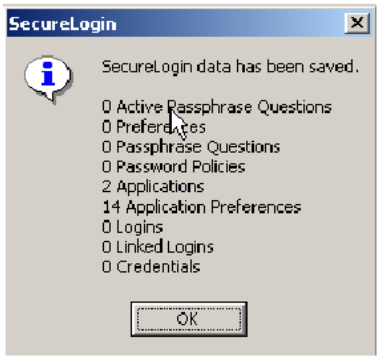
```
#=============================================================
# EXE/ID:                 GRPWISE.EXE
# Name:                   Novell Groupwise Client
# Type:                   Windows
# App Version:
# Script Version:         1.0
#
# Written By:             Protocom Development Systems
# Last Updated By:        Protocom Development Systems
#
# History:
#
# 10 November 2005 - Protocom - Created application definiti
#=============================================================

#=============================================================
# This script has been developed to support the Novell Grou
```

8 Select the file location and specify a file name in the *File Name* field.

9 Make sure XML document is selected in save as type.

10 Click *Save*. The SecureLogin confirmation message is displayed.

11 Click *OK*.



SecureLogin

SecureLogin data has been saved.

0 Active Passphrase Questions
0 Preferences
0 Passphrase Questions
0 Password Policies
2 Applications
14 Application Preferences
0 Logins
0 Linked Logins
0 Credentials

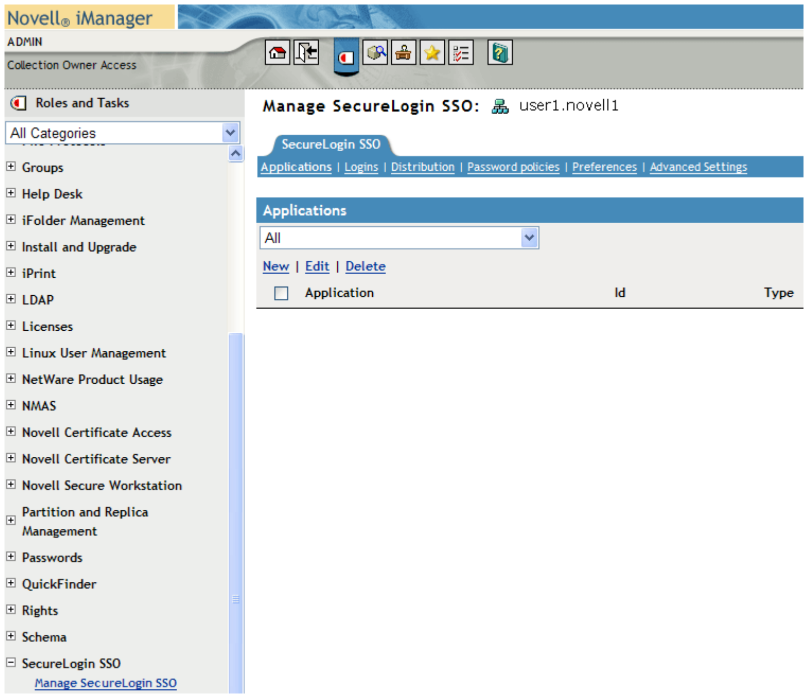
3.2.2 Importing in XML Format

Use the following procedure to import SecureLogin data in XML format:

1 Log in to iManager.

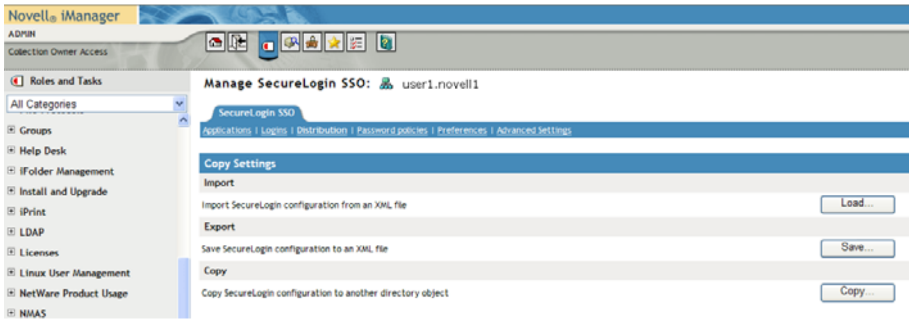
2 Select *Securelogin SSO* > *Manage Securelogin SSO*. The Manage SecureLogin SSO page is displayed.

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



Manage SecureLogin SSO: user1.novell1

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



Manage SecureLogin SSO: user1.novell1

SecureLogin SSO

Applications | Logins | Distribution | Password policies | Preferences | Advanced Settings

Copy Settings

Import

Import SecureLogin configuration from an XML file — Load...

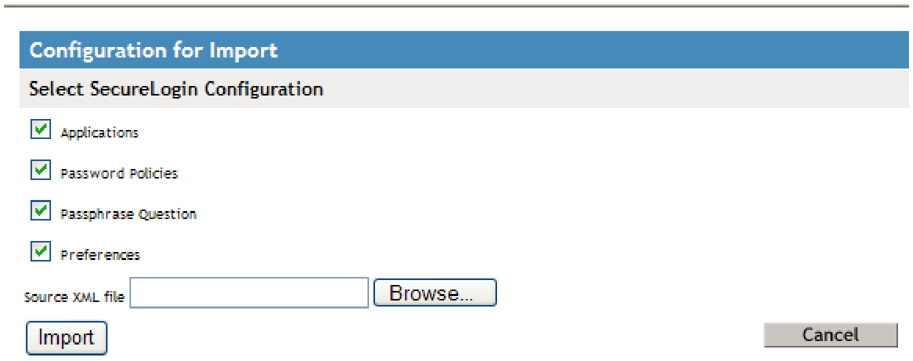
Export

Save SecureLogin configuration to an XML file — Save...

Copy

Copy SecureLogin configuration to another directory object — Copy...

5 Click *Load*. The Load dialog box is displayed.



Configuration for Import

Select SecureLogin Configuration

- [✓] Applications
- [✓] Password Policies
- [✓] Passphrase Question
- [✓] Preferences

Source XML file

6 In the dialog box do the following:

6a Select the *Applications* check box.

6b Clear the other check boxes.

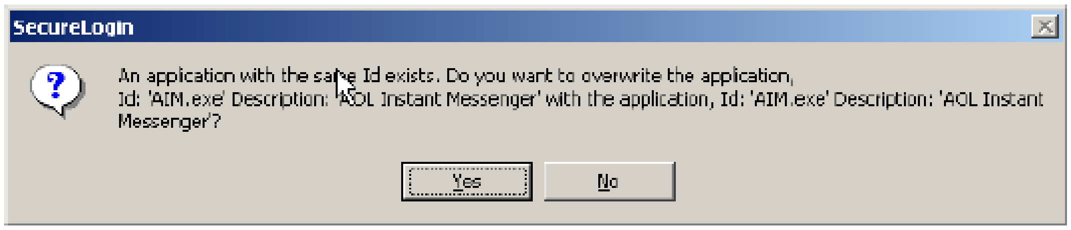
7 Browse and select the exported XML file.

8 Click *OK* to select the file.

The selected:

- Predefined applications and application definition are copied across to the receiving organizational unit or container.
- Securelogin configuration is copied across to the receiving object.

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

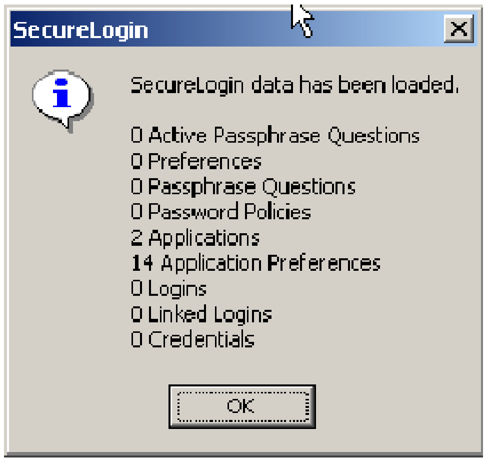


SecureLogin

An application with the same Id exists. Do you want to overwrite the application, Id: 'AIM.exe' Description: 'AOL Instant Messenger' with the application, Id: 'AIM.exe' Description: 'AOL Instant Messenger'?

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

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



SecureLogin data has been loaded.

0 Active Passphrase Questions
0 Preferences
0 Passphrase Questions
0 Password Policies
2 Applications
14 Application Preferences
0 Logins
0 Linked Logins
0 Credentials

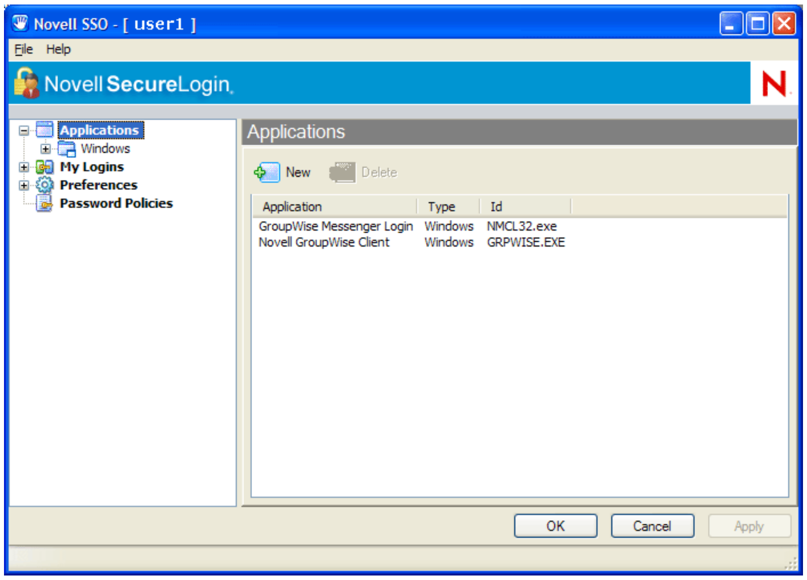
3.2.3 Modifying in the Personal Management Utility

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.

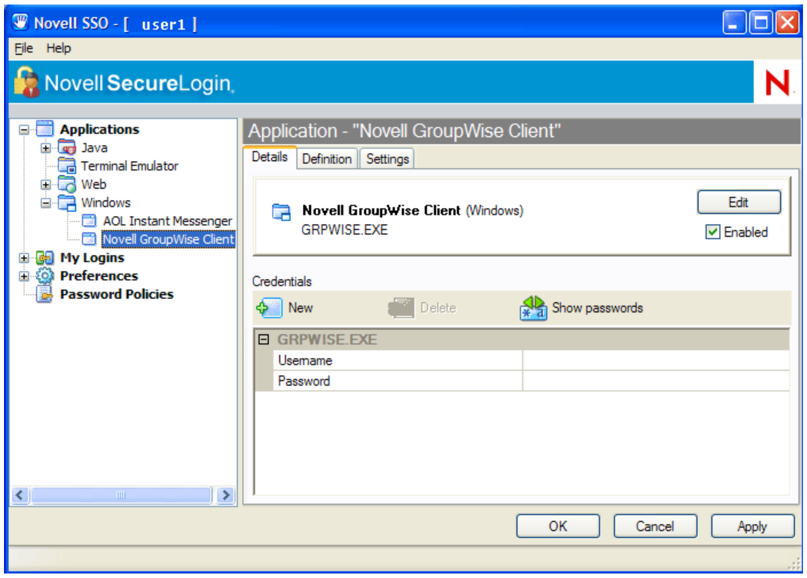
1 Double-click the SecureLogin system tray icon to display the Personal Management Utility.

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



Application	Type	Id
GroupWise Messenger Login	Windows	NMCL32.exe
Novell GroupWise Client	Windows	GRPWISE.EXE

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



Application - "Novell GroupWise Client"

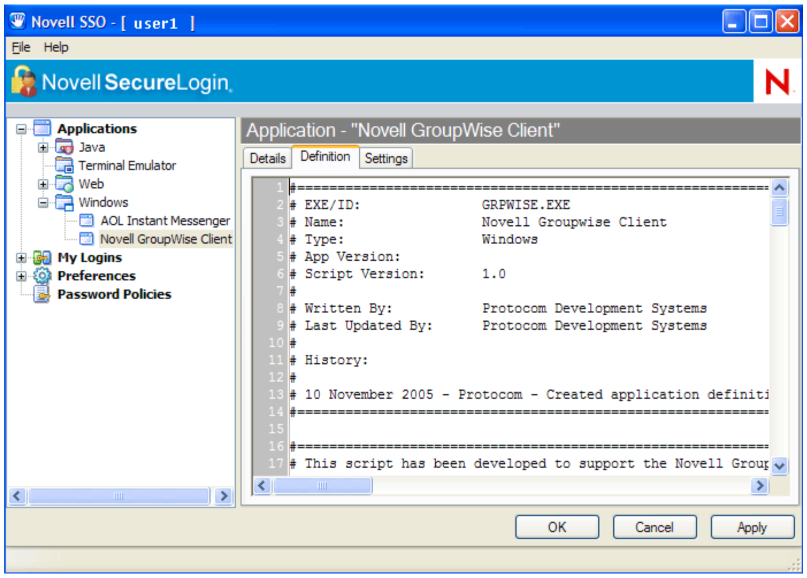
Novell GroupWise Client (Windows)
GRPWISE.EXE

[✓] Enabled

Credentials

GRPWISE.EXE	
Username	
Password	

4 Select the *Definition* tab. The Application Definition editor is displayed.



Application - "Novell GroupWise Client"

```
#============================================================
# EXE/ID:                GRPWISE.EXE
# Name:                  Novell Groupwise Client
# Type:                  Windows
# App Version:
# Script Version:        1.0
#
# Written By:            Protocom Development Systems
# Last Updated By:       Protocom Development Systems
#
# History:
#
# 10 November 2005 - Protocom - Created application definiti
#============================================================

#============================================================
# This script has been developed to support the Novell Grou
```

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

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

---

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

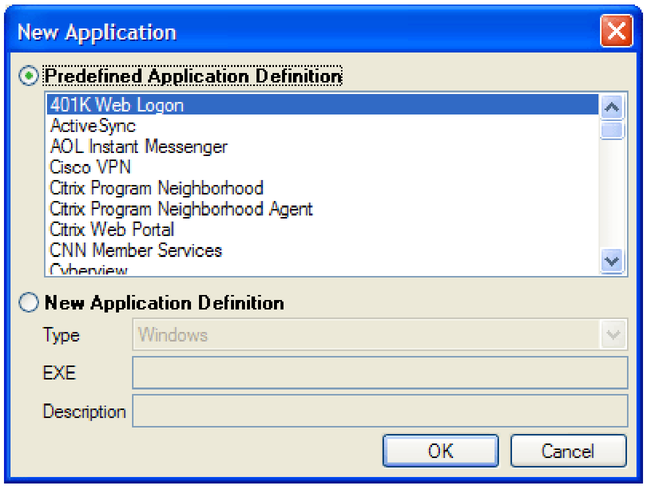
For information on how to modify specific functions see, Chapter 5, “Command Reference,” on page 49.

3.2.4 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 system tray icon to display the Personal Management Utility.
2. On the File menu, point to *New*, and then click *Application*. The New Application dialog box is displayed.

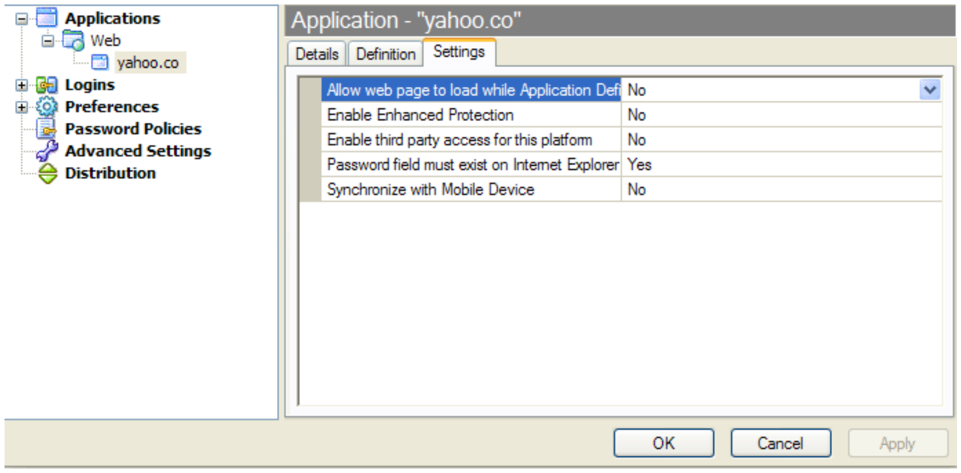


3. Click *New Application Definition*, and select the required application type from the *Type* drop-down list.
4. Specify the name of the application executable for Windows* applications in the *EXE* field.
5. Specify a description and click *OK*. The application definition is added to the left pane 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, and 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

The Settings tab includes the following Advanced options for application definitions and predefined applications.



Allow web page to load while Application Defi	No
Enable Enhanced Protection	No
Enable third party access for this platform	No
Password field must exist on Internet Explorer	Yes
Synchronize with Mobile Device	No

The following table lists the options in the Settings tab:

Table 3-1 *Settings Options*

Option	Description
Allow web page to load while Application Definition is running (Web applications only)	Applies to Microsoft Internet Explorer and Application Definitions created for Web pages and JavaScript log on that execute in a Web page. Set to *No* by default, this suspends completion of any other Internet Explorer tasks until the log on has completed. Selecting *Yes*, SecureLogin allows Internet Explorer to continue functioning while SecureLogin is executing log on.
Enable third party access for this platform	This option is set to *No* by default, disabling API access for this Predefined application or Application definition. Clear to disable API access for this Predefined Application or Application Definition.

Option	Description
Password field must exist on Internet Explorer page for Application Definition to run (Web applications only)	Applies to Microsoft Internet Explorer and Application Definitions created for Web pages and JavaScripts within Web pages. Selecting this option ensures SecureLogin does not execute automated log on for pages without a password field. Clear this option if your Web application returns errors on pages without password fields that you need to handle with SecureLogin. For example, password change successful.
Synchronize with Mobile Device	This option is set to *No* by default, enabling synchronization to an API enabled handheld device, for this Predefined application or Application Definition. Clear to disable synchronization to an API enabled handheld device for this Predefined application or Application Definition.

3.3 Windows Application Definition Tools

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

- Section 3.3.1, "Finding Application Details With Window Finder," on page 32
- Section 3.3.2, "Finding Application Details with the Login Watcher," on page 34

3.3.1 Finding Application Details With Window Finder

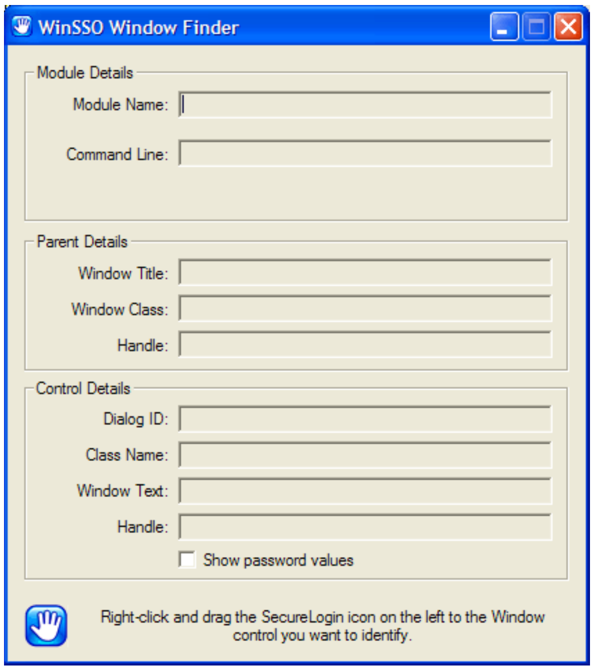
Window Finder finds windows applications details including control and dialog IDs. SecureLogin may require this information to identify specific objects in order to uniquely identify the application.

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.

Start SecureLogin's Window Finder

The following procedure uses the Novell SecureLogin test application provided on the SecureLogin distribution CD.

1 On the Windows start menu, select *All Programs > Novell Securelogin > Window Finder*. The WinSSO Window Finder is displayed.



Right-click and drag the SecureLogin icon on the left to the Window control you want to identify.

2 Right-click the securelogin icon in the dialog box, drag 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	Windows executable name for the selected application. This is the application name for a Windows Application Definition/ Predefined Application.
Command Line	The full command line used to start the application. You can use this information in conjunction with `GetCommandLine` command.
Parent Details section	

Field	Description
Window Title	Title of the window of the selected control.
	Use with the Title command in the `Dialog/EndDialog` section of the Application Definition.
Window Class	Windows class name for this dialog or window.
	Use with the Class command in a `Dialog or EndDialog` section.
Handle	The internal Windows handle for this window.
	Generally not used in Application Definitions.
Control Details section	
Dialog ID	The unique number identifying the control.
	Use with various commands including `Type, SetPlat`, and `Click.`
Class Name	Windows class name for the control.
	SecureLogin supported classes, which include Edit, Combo box, and Static.
Window Text	Test that exists on the control.
	Useful to copy and paste into the Application Definition editor.
	Note or copy the required details from the WinSSO Window Finder window from the relevant fields.
	Click Close to quit and close the WinSSO Window Finder window.

3.3.2 Finding Application Details with the Login Watcher

Login Watcher records log on and Windows application data to provide information that you may need for creating an application definition.

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: Login watcher records all log on information, including usernames and passwords, in a text file. This text file may be a security issue.

Information Details

Information Item	Description
Time	Milliseconds elapsed since logon Watcher started.
Module name	Name of the executable being recorded.

Information Item	Description
Window handle	Unique identifier for the window.
Window text	All text displayed in the window, which includes text entered during log on 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

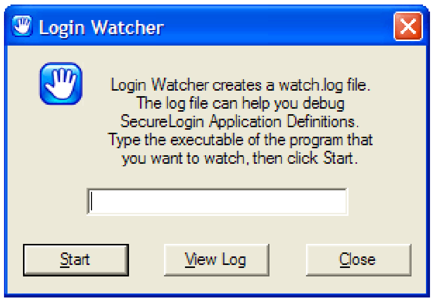
The following procedure uses the SecureLogin test application:

Starting Login Watcher

1 Right-click the SecureLogin task bar icon.

2 Select *close* from the menu.

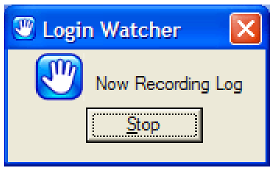
3 Double-click `loginwatch.exe`, by default located at `<...>\program files\novell\securelogin\tools`. the logon watcher dialog box is displayed.



Login Watcher creates a watch.log file. The log file can help you debug SecureLogin Application Definitions. Type the executable of the program that you want to watch, then click Start.

4 Specify the executable filename in the Login Watcher field.

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



Now Recording Log

6 Log on to the relevant application.

7 Click S*top* when logged on successfully to return to the login watcher dialog box.

8 click *View Log*. SecureLogin starts the Notepad application and displays the `watch.txt` file with log on details recorded.

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

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

3.4 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 *Description of symbols*

Symbol	Description
#	Use the hash symbol to define a line of text as a comment field. Comment fields are used to leave notes. Any line that starts with a # is ignored. Use comment lines for the following: ◆ Define sections of an Application Definition, for example the logon window, Change Password window, and so on. ◆ Explain complex sections. ◆ Remove command lines during creation and editing of the Application Definition. This saves having to continuously delete and rewrite lines while testing. ◆ Make notes such as when the Application Definition was written, what version of the software it was written for, and so on. When used within a command, the pound or hash symbol takes on a different meaning. When used as part of a command, such as Class or Type, the symbol specifies a numerical value. You can use these numerical values to specify a target for the command. Further details on this use are provided within the command listings.
" "	Use Quotation marks to group together text or variables that contain spaces. Quotation marks are used with commands such as `Type, MessageBox, If -Text,` and so on. 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 on details." ◆ If -Text "Log on failure"
$	Use the dollar sign to define the use of a SecureLogin variable stored in the directory for later use by that user. These variables are used to store information such as usernames and passwords. Further explanation on the various types of variables is provided in the next chapter.

Symbol	Description
?	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 to store temporary information, such as counting, data processing, and date information. The question mark is also used with several internal system generated variables.
%	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. Examples of the attributes you can use are FCN and %Surname.
!	Use the exclamation mark to define the use of a passticket. A passticket is a one-time password (OTP) that is generated using a combination of an encryption key, encryption offset, and the current time. Such passwords are only valid for a short period of time (generally between 30 seconds and 2 minutes). You can manually define the encryption key and offset, or the SecureLogin can generate it automatically. If the exclamation mark is included as the first character in a text string, then precede it with a backslash, otherwise SecureLogin will attempt to define a passticket. For example, `Type \!xyz"` will type "!xyz" to the application.
\	Use the backslash with the `Type` and `SendKey` commands to specify the use of a special function. The backslash is used in conjunction with values to perform the simulation of pressing keys. Examples of frequently used functions are provided in the following table: **Simulated Key Stroke:** Description **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 this symbol in a similar function to the backslash symbol, except its use is limited to HLLAPI enabled emulators. This symbol is used in conjunction with values to perform the simulation of key presses. For example, use @E to simulate pressing ENTER in a terminal emulator application.
-	Use the hyphen as a switch within several commands, such as `If` and `Type`. The hyphen is used in conjunction with values to modify the behavior of commands (such as -Raw), or switch on or off certain functions (such as -YesNo).

Application Definition Variables 4

This section contains the following information:

- Section 4.1, "Types of Variables," on page 39
- Section 4.2, "SecureLogin Supported Variables," on page 42
- Section 4.3, "Application Definition Conventions," on page 43

4.1 Types of Variables

SecureLogin supports the use of four different types of variables:

- Stored
- Runtime
- Directory attribute
- Passticket

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"`.

This section contains the following information:

- Section 4.1.1, "Using a Variable to Change the Default Platform," on page 39
- Section 4.1.2, "Directory Attribute Variables," on page 40
- Section 4.1.3, "Stored Variables," on page 40
- Section 4.1.4, "Runtime Variables," on page 41
- Section 4.1.5, "Passticket Variables," on page 42

4.1.1 Using a Variable to Change the Default Platform

Each variable defaults to the platform specified in the Application Definition/Predefined Application name. You can use a variable to change the platform, for example you may have an Application Definition named www.website1.com, for example:

```
type $username
type $password password
```

An Application Definition named www.website2.com may use the variables from www.website1.com, for example:

```
type $username(www.website1.com)
type $password(www.website1.com) password
Directory attribute variables
```

SecureLogin has the ability to read directory attributes from the currently logged on user's object. For example:

```
type%cn
```

will read the CN attribute from the currently logged on user's object and type it in. You can only use the percentage sign (%) variables when SecureLogin is configured to use a directory, and only on single-valued text attributes.

4.1.2 Directory Attribute Variables

SecureLogin has the ability to read directory attributes from the currently logged on user's object. For example:

```
type%cn
```

The above command reads the CN attribute from the currently logged on user's object and types it in. You can only use the percentage sign (%) variables when SecureLogin is configured to use a directory, and only on single-valued text attributes.

4.1.3 Stored Variables

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

This section contains the following information:

- "Where the Variables are Stored" on page 40
- "About Using Stored Variables" on page 40

Where the Variables are Stored

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 that you can have a different username variable for one application 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 Section 5.2.71, "SetPlat," on page 137.

About 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 more information, see Section 5.2.21, "DisplayVariables," on page 79 and Section 5.2.10, "ChangePassword," on page 64.

Example of stored variables in use:

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

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

4.1.4 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 with 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 Section 5.2.43, “Local,” on page 99.

Using Runtime Variable

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 using a runtime variable:

```
Dialog
Class #32770
Title "ERROR"
EndDialog
Local?ErrorCount
Increment?ErrorCount
If?ErrorCount Eq "2"
MessageBox "This is the second time you have8 received this error.
Would you like to reset the8 application?" -YesNo?Result
If?Result Eq "Yes"
KillApp "App.exe"
Run "C:\App\App.exe"
Else
Set?ErrorCount "0"
EndIf
EndIf
```

4.1.5 Passticket Variables

Passticket variables are preceded with the exclamation mark symbol (!). To use a passticket variable, you must create and define numerical values for stored variables with the names `$DESKEY` and `$DESOFFSET`. These numbers are then used by the SecureLogin Application Definition or Predefined Application parser to generate the one time password.

Use a Passticket Variable to Generate a Password

Once you have defined the stored variables, use the following passticket variable to generate a password.

```
!<Name of application definition>
```

or

```
!default
```

For example, if you wanted to use a passticket variable for the Outlook application, you would create two stored variables called `$DESKEY` and `$DESOFFSET` under the Outlook application definition. You then set values for the two stored variables, which allows you to use the variable "`!Outlook`" whenever you need to generate a one time password.

You could also use "`!Default`" which automatically reads the values from the current application definition.

If the credentials used to generate One Time Passwords (OTP's) do not already exist in a secured area of the SecureLogin cache (that is, the `$DESKEY` and `$DESOFFSET` variables are not defined), then they are retrieved from the closest SecureLogin Advanced Authentication server. For more information on this, contact Novell Technical Services.

4.2 SecureLogin Supported Variables

SecureLogin is able to read details from the system and use 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.

Table 4-1 *Variables and Descriptions*

Variable	Description
`?SysVersion(system)`	The local SecureLogin windows agent version. You can use this variable to determine if specific support is built into the product running on the user's workstation. Version convention is two digits for each section read from right to left, and leading zeroes are removed. For example, version 3.0.4.0 would be returned as 03000400.
`?BrowserType(system)`	Contains Internet Explorer or Netscape and indicates in which browser the Application Definition or Predefined Application is running. This variable is only set in a Web Application Definition or Predefined Application.

Variable	Description
`?SysUser(system)`	The name of the user currently using SecureLogin Single Sign-On.
`?SysPassword(system)`	The directory password of the user currently using SecureLogin. This variable is only available if the appropriate options are chosen when installing SecureLogin.
`?SysContext(system)`	The context within which the current SecureLogin user's directory object exists.
`?SysTree(system)`	The name of the directory tree that the SecureLogin Single Sign-On is currently using.
`?SysServer(system)`	The name of the server that was entered in the Novell log on. This variable is only available if the Novell client log on extension is installed.
`?CurrTime(system)`	The running time in seconds from January 1970 to the present. You can use this variable to force password changes every X days, and so on. Do not use the Application Definition to force a password change if you wish to continue having the application generate the change password event (recommended). Use this variable on applications where you cannot set a password expiry at the application back end.

4.3 Application Definition Conventions

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.

This section contains the following information:

- Section 4.3.1, "Symbols Used," on page 44
- Section 4.3.2, "Capitalization," on page 44
- Section 4.3.3, "Comments," on page 44
- Section 4.3.4, "Switches," on page 44
- Section 4.3.5, "Variables," on page 45
- Section 4.3.6, "Indent Sections," on page 45
- Section 4.3.7, "Blank Line Between Sections," on page 45
- Section 4.3.8, "Quotation Marks," on page 46
- Section 4.3.9, "Password Policy Names," on page 46

4.3.1 Symbols Used

Table 4-2 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.

4.3.2 Capitalization

Use capitalization where applicable.

Table 4-3 Capitalization

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

4.3.3 Comments

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

Table 4-4 Comments

Instead of...	Use...
`Dialog`	`# Written By: B. Smith 07/Jun/2002`
`Class #32770`	`# Last Modified By: C. Silvagni 13/Mar/2003`
`Title`	`# Logon Dialog Box`
`"Log on"EndDialog`	`Dialog`
	`Class #32770`
	`Title "Log on"`
	`EndDialog`

4.3.4 Switches

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

Table 4-5 Switches

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

4.3.5 Variables

All variable names start with a capital letter.

Table 4-6 Variables

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

4.3.6 Indent Sections

Indent sections between pairs of commands, for example `Dialog, Repeat, If`.

An indent of three spaces is recommended.

Table 4-7 Indent Sections

Instead of...	Use...
`If -Text "Some text"`	`If -Text "Some text"`
`#Do thisElse`	`#Do thisElse`
`#Do This`	`#Do this`
`EndIf`	`EndIf`

4.3.7 Blank Line Between Sections

Leave a blank line between sections, for example, between the `Dialog` Block and the rest of the Application Definition.

Table 4-8 Blank Line Between Sections

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

Write Subroutine Sections

Write subroutine sections at the bottom of the Application Definition and not half way 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.

4.3.8 Quotation Marks

Always use quotation marks around segments of text in commands.

Table 4-9 Enter Table Title Here

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

4.3.9 Password Policy Names

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

Table 4-10 Password Policy Names

Instead of...	Use...
`PasswordPolicy3`	`GroupwisePasswordPolicy`

At the top of the Application Definition enter and comment out information, for example, the author and the date of the last modification.

Table 4-11 *Example*

Instead of...	Use...

`Dialog`
 	
`Class #32770   Title "Log on"`
 	
`EndDialog`
	`# Written By: B. Smith 07/Jun/2002`
	
	`# Last Modified By: C. Silvagni 13/Mar/2003`
	
	`# Logon Dialog Box`
	
	`Dialog`
	
	`Class #32770`
	
	`Title "Log on"`
	
	`EndDialog`

NOTE: Always place the `Title` command after all other commands in the `Dialog` block.

Command Reference

5

This section contains the following information:

- Section 5.1, "Command Reference Conventions," on page 49
- Section 5.2, "Commands," on page 53

5.1 Command Reference Conventions

This section consists of descriptions and examples of the commands that make up Novell® SecureLogin Application Definitions.

NOTE: For a list of commands and corresponding page references, see Chapter 1, "Command Quick Reference," on page 11.

This section contains the following information:

- Section 5.1.1, "Command Information," on page 49
- Section 5.1.2, "Web Wizard Application Definition Conventions," on page 50
- Section 5.1.3, "Site Matching," on page 51
- Section 5.1.4, "Form/Field/Option Matching," on page 51
- Section 5.1.5, "Form/Field/Option ID's," on page 52
- Section 5.1.6, "Audit Integration," on page 52
- Section 5.1.7, "One Time Passwords," on page 52

5.1.1 Command Information

The information for each of the commands includes:

Use With Values

Table 5-1 *Command Description*

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/internet Application Definition. Not compatible with Web Wizard Application Definition language.

Command	Description
Web Wizard	Use a 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.

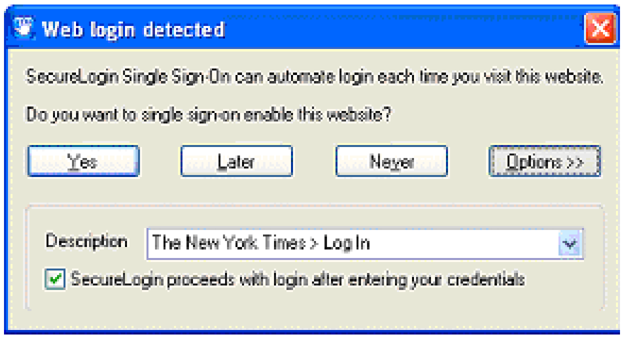
Type Values

Table 5-2 *Command Description*

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.

5.1.2 Web Wizard Application Definition Conventions

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


Web login detected

SecureLogin Single Sign-On can automate login each time you visit this website.

Do you want to single sign-on enable this website?

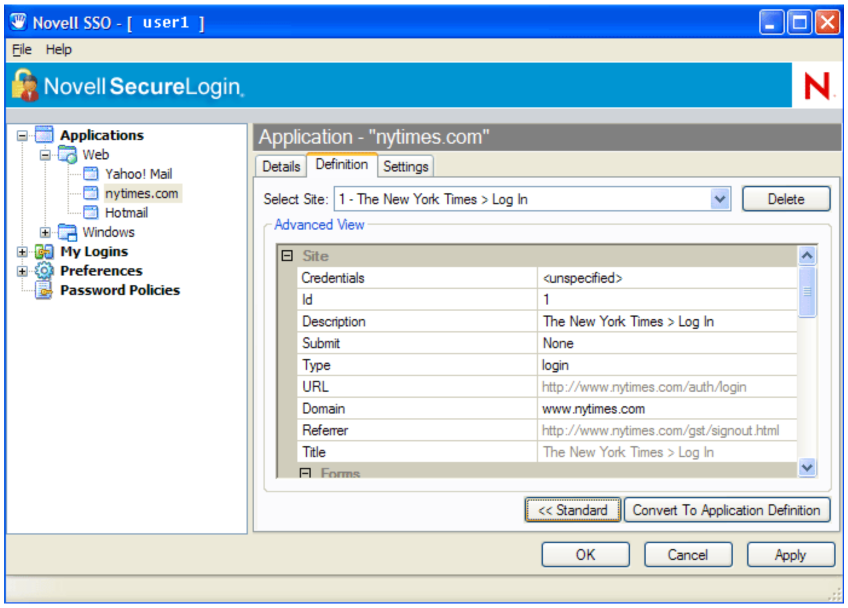
Description: The New York Times > Log In

✓ SecureLogin proceeds with login after entering your credentials

The Web Wizard captures the user's log on details and adds them to the user's Web Application Definitions.

When managing user's Web log on 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 which provides more functionality with their associated values and the option to convert the user's log on credentials to an Application Definition.



Application - "nytimes.com"

Select Site: 1 - The New York Times > Log In

Site	
Credentials	<unspecified>
Id	1
Description	The New York Times > Log In
Submit	None
Type	login
URL	http://www.nytimes.com/auth/login
Domain	www.nytimes.com
Referrer	http://www.nytimes.com/gst/signout.html
Title	The New York Times > Log In
Forms	

For more details on how to manage Application Definitions, see Chapter 3, "Managing Application Definitions," on page 23.

5.1.3 Site Matching

In SecureLogin version 3.5 and higher, Web commands have been added to allow for much finer control of site matching. Detailed information of the loaded Web site can be matched upon 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 more information, see Section 5.2.73, "Site/EndSite," on page 141.

5.1.4 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.

For example:

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

matches a site with two password fields. The first is given the ID '#1:1' , the second is given the id '#1:2'

NOTE:
- Matched items may only be matched once.
- Each ID must be unique and not used previously.

5.1.5 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".

5.1.6 Audit Integration

SecureLogin 6.0 SP1 incorporates a Novell Audit integration for those enterprises that have Novell Audit as part of their infrastructure. Novell Audit allows administrators to audit events from scripts and have the Novell Audit client write audit events in response to certain triggering events.

For more information, see Section 5.2.5, “AuditEvent,” on page 60.

5.1.7 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 helpdesk to reset forgotten passwords, or ensure 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 when 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 OTP will rarely change. System restoration, which may be hours or be many months old, can be achieved without consideration for restoring users' passwords or notifying staff of new passwords.

SecureLogin 6.0 SP1 now provides a secure, robust and scalable infrastructure by integrating ActivCard one time password authentication functionality. It provides administrators access to the application definition command `GenerateOTP` which can be used to generate synchronous authentication and asynchronous authentication soft token support for smartcard user authentication as well as hard token support for Vasco Digipass token generator.

For more information, see Section 5.2.28, “GenerateOTP,” on page 84.

5.2 Commands

This section contains the following information:

- Section 5.2.1, "Regular Expressions," on page 55
- Section 5.2.2, "AAVerify," on page 55
- Section 5.2.3, "ADD," on page 58
- Section 5.2.4, "Attribute," on page 59
- Section 5.2.5, "AuditEvent," on page 60
- Section 5.2.6, "BeginSplashScreen/EndSplashScreen," on page 60
- Section 5.2.7, "Break," on page 61
- Section 5.2.8, "BooleanInput," on page 63
- Section 5.2.9, "Call," on page 63
- Section 5.2.10, "ChangePassword," on page 64
- Section 5.2.11, "Class," on page 66
- Section 5.2.12, "ClearPlat," on page 67
- Section 5.2.13, "ClearSite," on page 70
- Section 5.2.14, "Click," on page 71
- Section 5.2.15, "ConvertTime," on page 73
- Section 5.2.16, "Ctrl," on page 74
- Section 5.2.17, "DebugPrint," on page 75
- Section 5.2.18, "Decrement," on page 76
- Section 5.2.19, "Delay," on page 77
- Section 5.2.20, "Dialog/EndDialog," on page 78
- Section 5.2.21, "DisplayVariables," on page 79
- Section 5.2.22, "Divide," on page 81
- Section 5.2.23, "DumpPage," on page 82
- Section 5.2.24, "EndScript," on page 82
- Section 5.2.25, "Event," on page 83
- Section 5.2.26, "Event Specifiers," on page 83
- Section 5.2.27, "FocusInput," on page 83
- Section 5.2.28, "GenerateOTP," on page 84
- Section 5.2.29, "GetCheckBoxState," on page 86
- Section 5.2.30, "GetCommandLine," on page 87
- Section 5.2.31, "GetEnv," on page 88
- Section 5.2.32, "GetIni," on page 88
- Section 5.2.33, "GetMD5," on page 89
- Section 5.2.34, "GetReg," on page 90
- Section 5.2.35, "GetSessionName," on page 90

- Section 5.2.36, “GetText,” on page 91
- Section 5.2.37, “GetURL,” on page 91
- Section 5.2.38, “GoToURL,” on page 92
- Section 5.2.39, “If/Else/Endif,” on page 93
- Section 5.2.40, “Include,” on page 96
- Section 5.2.41, “Increment/Decrement,” on page 97
- Section 5.2.42, “KillApp,” on page 98
- Section 5.2.43, “Local,” on page 99
- Section 5.2.44, “MatchDomain,” on page 100
- Section 5.2.45, “MatchForm,” on page 101
- Section 5.2.46, “MatchField,” on page 103
- Section 5.2.47, “MatchForm,” on page 104
- Section 5.2.48, “MatchOption,” on page 106
- Section 5.2.49, “MatchReferer,” on page 107
- Section 5.2.50, “MatchURL,” on page 108
- Section 5.2.51, “MessageBox,” on page 109
- Section 5.2.52, “Multiply,” on page 112
- Section 5.2.53, “OnException/ClearException,” on page 113
- Section 5.2.54, “Parent/EndParent,” on page 115
- Section 5.2.55, “PickListAdd,” on page 117
- Section 5.2.56, “PickListDisplay,” on page 118
- Section 5.2.57, “PositionCharacter,” on page 119
- Section 5.2.58, “PressInput,” on page 120
- Section 5.2.59, “ReadText,” on page 120
- Section 5.2.60, “RegSplit,” on page 123
- Section 5.2.61, “ReLoadPlat,” on page 124
- Section 5.2.62, “Repeat/EndRepeat,” on page 126
- Section 5.2.63, “RestrictVariable,” on page 128
- Section 5.2.64, “Run,” on page 130
- Section 5.2.65, “SelectListBoxItem,” on page 131
- Section 5.2.66, “SendKey,” on page 132
- Section 5.2.67, “Set,” on page 133
- Section 5.2.68, “SetCheckBox,” on page 134
- Section 5.2.69, “SetCursor,” on page 135
- Section 5.2.70, “SetFocus,” on page 136
- Section 5.2.71, “SetPlat,” on page 137
- Section 5.2.72, “SetPrompt,” on page 139
- Section 5.2.73, “Site/EndSite,” on page 141

- Section 5.2.74, “StrCat,” on page 142
- Section 5.2.75, “StrLength,” on page 143
- Section 5.2.76, “StrLower,” on page 144
- Section 5.2.77, “StrUpper,” on page 145
- Section 5.2.78, “Sub/EndSub,” on page 146
- Section 5.2.79, “Submit,” on page 147
- Section 5.2.80, “Subtract,” on page 148
- Section 5.2.81, “Tag/EndTag,” on page 150
- Section 5.2.82, “TextInput,” on page 150
- Section 5.2.83, “Title,” on page 151
- Section 5.2.84, “Type,” on page 152
- Section 5.2.85, “Sending Keyboard Commands Using Type,” on page 155
- Section 5.2.86, “WaitForFocus,” on page 156
- Section 5.2.87, “WaitForText,” on page 157

5.2.1 Regular Expressions

^Match character at beginning of the string. The expression "^A" will match an ‘A’ only at the beginning of the string.

^The caret (^) immediately following the left-bracket ([) has a different meaning. It is used to exclude the remaining characters within brackets from matching the target string. The expression "[^0-9]" indicates that the target character should not be a digit.

$ The dollar sign ($) will match the end of the string. The expression "abc$" will match the sub-string "abc" only if it is at the end of the string.

| The alternation character (|) allows either expression on its side to match the target string. The expression "a|b" will match ‘a’ as well as ‘b’.

. The dot (.) will match any character.

* The asterix (*) indicates that the character to the left of the asterix in the expression should match 0 or more times.

+ The plus (+) is similar to asterix but there should be at least one match of the character to the left of the + sign in the expression.

? The question mark (?) matches the character to its left 0 or 1 times.

() The parenthesis affects the order of pattern evaluation and also serves as a tagged expression that can be used when replacing the matched string with another expression.

[] Brackets ([and]) enclosing a set of characters indicates that any of the enclosed characters may match the target

5.2.2 AAVerify

Table 5-3 *Command Description*

Use With	Startup, Terminal Launcher, Web and/or Windows
SecureLogin Version	All (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 NMAS 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.

Description

Use `AAVerify` with SecureLogin Advanced Authentication or Novell NMAS to verify the user. It is typically used before the application Username and Password are retrieved and entered into the login dialog box.

This provides application reauthentication using a strong log on method. For example, a user might be forced to enter their smartcard and PIN before the application logs 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`.

NMAS Specific: If `AAVerify` is called with no arguments, then the currently logged on user is reauthenticated using the log on method that they used for their current log on.

AA Specific: When `AAVerify` is called in an AA environment, the -method parameter must be present. The method must be one of the following:

- Any
- Biometric
- Smartcard
- Token
- Password
- Passphrase
- Directory
- Password
- SecureID

You can specify more than one `-method` argument. In this case the user is allowed to reauthenticate with any of the specified methods. For example, you could use the command to request authentication using a fingerprint device or a smartcard.

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 ACLs 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 8 "BSmith" -
Tree "Production" ?Result
```

Example	Windows Application Definition
	
	This example detects the login dialog box, but before SecureLogin Single Sign-On enters the user's credentials, it prompts the user to provide their Advanced Authentication credentials such as smartcard and PIN, biometric and token.
	
	`# Log on Dialog Box`
	`Dialog`
	`Ctrl #32770`
	`Title “Log on”`
	`EndDialog`
	`AAVerify –Method “Enhanced Password” ?Result`
	`If ?Result Eq “True”`
	`Type “$Username” #1001`
	`Type “$Password” #1002`
	`Click #1`
	`Else`
	`MessageBox “Authentication failed! Please verify your smart card is inserted and your PIN is correct. IT x453”`
	`EndIf`

5.2.3 ADD

Table 5-4 *Description of Add Command*

Used With	Startup, Terminal Launcher, Web and/or Windows
SecureLogin Version	3.0
Type	Variable Manipulator
Usage	`Add <Variable1> <Variable2> [?Result]`

Arguments	`<Variable1>`
	
	The first argument, the number to which the second argument is added. This argument contains 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 the result of the equation.

Description	Adds one number to 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.

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

5.2.4 Attribute

Table 5-5 *Description of Attribute*

Use With	Advanced Web Application Definition
SecureLogin Version	3.5
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 Section 5.2.81, “Tag/EndTag,” on page 150.

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`

5.2.5 AuditEvent

Table 5-6 *Description of AuditEvent*

Use With	Startup, Terminal Launcher, Web and Windows applications definitions for those enterprises that have Novell Audit as part of their infrastructure
SecureLogin Version	6.0 SP1
Type	Specifier
Usage	`AuditEvent [<message>]`
Arguments	`<message>` The variable or text string passed to the Novell Audit server.
Description	Use `AuditEvent` to log SecureLogin events to the Novell Audit server. If the `Type` command is used with `ChangePassword` command to generate a `$password` variable, this will then trigger a log event to the Novell Audit server.

Example	If the Audit platform agent is not present on the workstation, no event is logged.
	
	`AuditEvent "message"`
	
	The parameter “message” is the string passed to the Novell Audit server.
	
	`AuditEvent $message`
	
	The parameter `$message` is the variable passed to the Novell Audit server.

5.2.6 BeginSplashScreen/EndSplashScreen

Table 5-7 *BeginSplashScreen/EndSplashScreen Description*

Use with	Terminal Launcher (Generic and Advanced Generic Only)
SecureLogin Version	3.0.4
Type	Action

Usage	`BeginSplashScreen`
	
	`EndSplashScreen`

Arguments	None
Description	Use to display a Novell splash screen across the whole Terminal Emulator window. This is used to mask any flashing and so on that is produced 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 Single Sign-On waits 2 seconds for it to connect. The splash screen is displayed to cover the flashing, the log on is detected, the username is entered, then the splash screen disappears.
	`Delay 2000`
	`BeginSplashScreen`
	`WaitForText "ogin:"`
	`Type $Username`
	`EndSplashScreen`
	`Type @E`

5.2.7 Break

Table 5-8 *Description of Break*

Use With	Startup, Terminal Launcher, Web and Windows
SecureLogin Version	3.5.1
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 word 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 is logged in, 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 "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
```

Example 2 (Contd)

```
#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
EndAccount is locked for some reason, possibly
inactive.   If -Text "Account Locked"If#
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
```

5.2.8 BooleanInput

Table 5-9 *Description of Boolean Input*

Use With	Advanced Application Definitions created using the Web Wizard.
SecureLogin Version	3.5.1
Type	Action
Usage	`BooleanInput #FormID:FieldID -check "check"`

Arguments	`#FormID`
	The ID to be given to the matched form. The ID must be a static unsigned integer.
	`#FieldID`
	The ID to be given to the matched field. The ID must be a static unsigned integer.
	`-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 checkbox or radio button).

Example	In this example the value of field #1:3 is being checked by the Application Definition.
	`# === Login Application Definition #2 ==`
	`# === Google Initial Login ====`
	`#=========================================`
	`Site Login -userid “Google Log On” -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`

5.2.9 Call

Table 5-10 *Description of Call*

Use With	Startup, Terminal Launcher, Web and/or Windows
SecureLogin Version	3.5.1

Type	Flow control
Usage	`Call <SubRoutine>`
Arguments	`<SubRoutine>` The name of the subroutine called. This name must be identical to the name specified 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 the subroutine completes, the Application Definition resumes executing from the command immediately following the `Call` command.
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. Subroutines are useful when you would otherwise have to repeat the same lines of Application Definition over again.

```
Repeat
If -Text "Username"
Call "Log on"
EndIf
If -Text "Wrong Password"
Call "WrongPassword"
EndIf
Delay 100
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 verify your password and click
OK to retry log on. IT x453."? $Password
Call Log on
EndSub
```

5.2.10 ChangePassword

Table 5-11 *Description of ChangePassword*

Use With	Startup, Terminal Launcher, Web and Windows
SecureLogin Version	All

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 invokes the random password generator.

Description	Use `ChangePassword` to change a single variable and is used 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. For more information see Section 5.2.64, “Run,” on page 130.

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 username and password, and then the 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 $Password #1006`
	`Click #1#`
	`Change Password Failed Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Change Password Failed"`
	`EndDialog`
	`# Set the password back as the password change failedSet`
	`$Password $PasswordBackupMessageBox "The change password process failed. Please retry the password change at your next log on. IT x453."`

5.2.11 Class

Table 5-12 *Description of Class*

Use With	Startup, Windows
SecureLogin Version	All
Type	Dialog Specifier
Usage	`Class <-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 execution of 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 SecureLogin 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.
	
	`# Log on Dialog Box`
	`Dialog`
	`Class "#32770"`
	`Title "Log on"`
	`EndDialog`
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

5.2.12 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 log on 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 Reentering User ID Details

If the log on 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 enter them again. However, this could result in changing the wrong credentials if the user selects the incorrect User ID.

To resolve this issue, you can use the `SetPlat`, `ReLoadPlat` and `ClearPlat` commands. `ReloadPlat` sets the current User ID to the one which was chosen last 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.

For more information see Section 5.2.61, “ReLoadPlat,” on page 124 and Section 5.2.12, “ClearPlat,” on page 67.

Table 5-13 *Description of ClearPlat*

Use With	Startup, Terminal Launcher, Web and/or Windows
SecureLogin Version	V.3.6.0 SP1
Type	Action
Usage	There are three main places where code needs to be added to use the `ClearPlat` command. They are: **Application Startup** When an application first starts up, use `ClearPlat` to clear the previously chosen platform. You can 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 Log on** Call `ClearPlat` to allow the user to relog on 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 (Cont.)	`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 ===`

5.2.13 ClearSite

Table 5-14 *Description of ClearSite*

Use with	Startup, Terminal Launcher, Web and/or Windows and Advanced Application Definitions created using the Web Wizard.
SecureLogin Version	3.6.10
Type	Action
Usage	`ClearSite “SiteName”`
Arguments	`SiteName”` The name of the site to clear
Description	Used inside a `Site` block 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.

Example 1	In this example the user is redirected to the main Google portal and any previous user information is cleared.
	`GotoURL “http://www.google.com”`
	`ClearSite Login`

Example 2	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.
	`MessageBox "Would you like to login again?" -`
	`yesno`
	`?Continue`
	`If ?Continue eq "Yes"`
	`TextInput #1:1 -value "$Username"`
	`TextInput #1:2 -value "$Password"`
	`FocusInput #1:2 -focus "true"`
	`BooleanInput #1:3 -check "false"`
	`PressInput`
	`Else`
	`ClearSite Login`
	`EndIf`

5.2.14 Click

Table 5-15 *Description of Click*

Use With	Java, Web, Windows
SecureLogin Version	All
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 >]]`
Web Usage	`Click <#Number>`

Arguments	◆ `<#Ctrl-ID>` The ID number of the control to be pressed. ◆ `[-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 Section 5.2.24, "EndScript," on page 82. ◆ `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 often clicks 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 Single Sign-On 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. 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. For more information see Section 5.2.24, "EndScript," on page 82.

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

Example 1	Windows Application Definition
	
	This example detects the login dialog box, enters the username and password, and clicks the button number 1.
	`# Log on 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 username and password, and clicks the login button.
	`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.
	`# Log on Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Log on"`
	`End Dialog`
	`Type $Username`
	`Type $Password`
	`Click -X 12 -Y 24`

5.2.15 ConvertTime

Table 5-16 *Description of ConvertTime*

Use With	Startup, Terminal Launcher, Web and Windows
SecureLogin Version	3.0.4
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 stores it in `<String Time>`.

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

5.2.16 Ctrl

Table 5-17 *Description of Ctrl*

Use With	Startup, Windows, Java
SecureLogin Version	All
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 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 ID's 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.
	`# Log on Dialog Box`
	`Dialog`
	`Ctrl #1 "OK"`
	`Ctrl #2 "Cancel"`
	`Ctrl #3 "Help"`
	`Title "Log on"`
	`EndDialog`
	`Type $Username`
	`Type "\T"`
	`Click #1`

5.2.17 DebugPrint

Table 5-18 *Description of DebugPrint*

Use With	All
SecureLogin Version	6.0
Type	Action
Usage	`DebugPrint <data>`
Arguments	`<data>` The text displayed to the user. `<Data>` can be serveral strings, variables, or a combination of both.
Description	Use the `DebugPrint` command to display the text specified in the `<data>` variable in the 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").
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.
	`# Login Dialog`
	`#`
	`Dialog`
	`    Class "#32770"`
	`    Title "Log on"`
	`EndDialog`
	`    ReadText #1003 ?ServerText`
	`    RegSplit "Server: (.*)" ?ServerText ?Server-`
	`    Name`
	`    DebugPrint "Setting the platform to " ?Server-`
	`    Name`
	`    SetPlat ?ServerName`
	`    Type $Username #1001`
	`    Type $Password #1002`
	`    Click #1`

5.2.18 Decrement

***Table 5-19** Description of Decrement*

Use with	All
SecureLogin Version	3.5.1 to 6.0
Type	Variable Manipulator
Usage	Decrement `<variable>`
Arguments	`<variable>` The name of the variablle to decrease in value.
Description	Use the `Decrement` command to subtract from a specified variable. For example, you can use decrement to count the number of passes a particularApplication 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 logon 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. For more information, see Section 5.2.41, “Increment/Decrement,” on page 97.
Syntax Examples	Decrement ?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.

```
#Log on 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

# Log on Successful Message
Dialog
    Ctrl #1
    Title “Log on Successful”
EndDialog

Set ?RunCount “0”
```

5.2.19 Delay

Table 5-20 *Description of Delay*

Use with	All
SecureLogin Version	3.5.1 to 6.0
Type	Action
Usage	`Delay <Time Period>`
Arguments	`<Time Period>` A period of time, expressed in milliseconds (1/1000 of a second), during which the 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 logon box, but the Application Definition waits half a second before acting upon it to make sure that the box is complete.
	`# Log on Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Log on"`
	`EndDialog`
	
	`Delay 500`
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

5.2.20 Dialog/EndDialog

Table 5-21 *Description of Dialog/EndDialog*

Use With	Java, Windows
SecureLogin Version	All
Type	Dialog Specifier
Usage	`DialogEndDialog`
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 then 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 dialog box, the Application Definition parses the `Type` and `Click` commands to complete the login process.
	`# Log on Dialog Box`
	`Dialog`
	`Ctrl #1 "OK"`
	`Title "Log on"`
	`Parent`
	`Title "Application 1"`
	`EndParent`
	`EndDialog`
	
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

5.2.21 DisplayVariables

Table 5-22 *Description of Display Variables*

Use With	All
SecureLogin Version	All
Type	Action
Usage	`DisplayVariables [<User Prompt>] [<Variable> [<variable>] …]`

Arguments	`[<User Prompt>]`
	
	Optional, customized text displayed in the Enter SecureLogin Variables dialog box.
	
	`[<Variables>]`
	
	The name of the variables for which you want the user prompted. If not specified, SecureLogin prompts 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 log on is unsuccessful due to an incorrect username or password, the `DisplayVariables` command prompts the user to edit the stored username or password values. The log on process proceeds as normal from that point. You can 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 is 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	Windows Application Definition
	
	This example detects the Wrong Password dialog box, and SecureLogin prompts the user to enter a new username 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`

5.2.22 Divide

Table 5-23 *Description of Divide*

Use With	Startup, Terminal Launcher, Web and/or Windows
SecureLogin Version	3.0
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, or the result of the equation.

Description	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. **NOTE:** This is an integer arithmetic that is 5/2, not 2.5.

Syntax Examples	`Divide “1” “2”?Result`
	
	`Divide ?Login Attempts ?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`

5.2.23 DumpPage

Table 5-24 Description of Dump Page

Use With	Advanced Web Application Definition
SecureLogin Version	3.5
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`

5.2.24 EndScript

Table 5-25 Description of EndScript

Use With	Startup, Terminal Launcher, Web and/or Windows
SecureLogin Version	All
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 dialog box, and SecureLogin enters the username 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 "Log on Failure"`
	`Ctrl #1`
	`EndDialog`
	`ReadText #65535 ?ErrorMsg`
	`If "Incorrect Password" -In ?ErrorMsg`
	`MessageBox "You have entered an incorrect password"`
	`EndScript`
	`EndIf`

5.2.25 Event

Table 5-26 *Description of Event*

Use With	Windows
SecureLogin Version	3.5
Type	Dialog Specifier
Usage	`Event <Event>`
Arguments	`<Event>` The application event to monitor. This corresponds to a Windows event, which usually begins with WM_.
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.

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

5.2.26 Event Specifiers

For details on Windows Event commands, see the Microsoft* MSDN Web site located at `http://msdn.microsoft.com`. Microsoft's Spy++, or similar Windows* message spy tools, are also useful for trapping event names in specific windows. Information regarding Spy ++ is also available on the MSDN Web site.

5.2.27 FocusInput

Table 5-27 *Description of FocusInput*

Use With	Startup, Terminal Launcher, Web and/or Windows and Advanced Application Definitions created using the Web Wizard.
SecureLogin Version	3.5.1
Type	Action
Usage	`FocusInput #FormID:FieldID [-focus "focus"]`

Arguments	`#FormID`
	
	The ID to be given to the matched form. The ID must be a static unsigned integer.
	
	`#FieldID`
	
	The ID to be given to the matched field. The ID must be a static unsigned integer.
	
	`-focus "focus"`
	
	Focuses the input field based upon the Boolean value of "focus". The Boolean value can be either "true" or "false.

Description	Used to focus on an input field based upon the Boolean value of "focus".

Example	In this example the value of field #1:2 is being checked by the Application Definition.
	`# === Login Application Definition #2 ==`
	`# === Google Initial Login ====`
	`#=========================================`
	`Site Login -userid “Google Log On” -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`

5.2.28 GenerateOTP

Table 5-28 *Description of GenerateOTP*

Use With	Startup, Terminal Launcher, Web and Windows
SecureLogin Version	3.5.0 and higher
Type	Action
Usage	`GenerateOTP -mode <string>-challenge <string>`

Arguments

`<result>`

A variable that receives the value of the one time password that is generated.

`-mode`

Specifies the type One Time Password (OTP) that is dynamically generated. The default value for mode is set to "soft" for the Vasco soft token. Setting this to "AISC-SKI" makes SecureLogin use the algorithm to generate an OTP based on the user’s smartcard.

`-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

A one time password (OTP) is an authentication method specifically designed to avoid the security exposures inherit with traditional fixed and static password usage.

OTPs rely upon a pre-defined 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.

GenerateOTP was an undocumented command initially developed in SecureLogin version 3.5.1 to meet a specific client requirement and was for use with the Vasco Digipass hard token generator. For information regarding this configuration please contact Novell Technical Support about Mainframe OTP solutions.

In SecureLogin version 6 the GenerateOTP command was enhanced to incorporate OTP soft token generation functionality embedded in Smartcard functionality.

Soft tokens can be generated in synchronous and asynchronous mode which now allows soft tokens to be loaded onto mobile devices such PDAs and can even sent to cell phones as SMS text messages.

Synchronous Mode: Synchronous authentication replaces static alpha/ numeric passwords with a pseudo random code that is dynamically generated at configured time intervals generally around each 60 seconds. 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 alpha/numeric passwords with a pseudo random code that is dynamically generated based on a shared encryption key, the current time and a challenge/response combination.

The application definition example 1 shows a typical command structure to enable OTP for use with a Vasco Digipass hard token generator.

The application definition example 2 shows a typical command structure to enable OTP for use with the Smartcard technology.

Example	In SecureLogin version 6, the GenerateOTP command has been enhanced to integrate with smartcards.
	
	In Synchronous mode the GenerateOTP command will require the administrator to pass the -mode variable, AISC-SKI to the command.
	
	In this instance AISC-SKI is the Smartcard and SKI is the name of the applet used on the smartcard.
	
	An example application definition enabling synchronous 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 ?OtpResult #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 checks for a variable named mode before assuming the default which is to generate a Vasco Token. Values passed into the command via the script over rides 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 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 smartcard has not been authenticated the user is prompted to enter a PIN to unlock the card. This is required only once as the PIN is normally cached.

5.2.29 GetCheckBoxState

Table 5-29 *Description of GetCheckBoxState*

Use with	Advanced Web Application Definition
SecureLogin Version	3.5
Type	Action
Usage	`GetCheckBoxState <#Item Number> <Variable>`

Arguments	`<Item Number>`
	The ID of the checkbox.
	`<Variable>`
	The target variable for the status of the specified check box. Value returned is Checked or 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 checkbooks.

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

5.2.30 GetCommandLine

Table 5-30 *Description of GetCommandLine*

Use with	Startup, Windows
SecureLogin Version	3.0.4
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 database for use with multiple logon 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:\Winnt\Notepad.exe"  KillApp Notepad.exe`
	`EndIf`

5.2.31 GetEnv

Table 5-31 *Description of GetEnv*

Use with	All
SecureLogin version	3.5
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`

5.2.32 GetIni

Table 5-32 *Description of GetIni*

Use With	Windows, Web, Terminal, Java*
SecureLogin Version	3.5
Type	Action
Usage	`GetIni <ini file> <section> <key> <variable>`

Arguments	`<Ini File>`
	
	This is the filename 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 environment variable data

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 ?NotesDefaultIDFile`

5.2.33 GetMD5

Table 5-33 *Description of GetMD5*

Use With	All
SecureLogin Version	3.5
Type	Action
Usage	`GetMD5 <value>`
Arguments	`<value>` Returns the MD5 hash value.
Description	Use the `GetMD5` command to generate an MD5 hash value of the current process the script running for. `GetMD5` works only 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 is 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 as a variable which is then passed in as the argument to the command, which could be a `?tmp` or *$hash_value* type variable.
	
	`GetMD5 ?tmp`
	
	or
	
	`GetMD5 $hash_value`
	
	The MD5 hash value would normally be obtained from the windows finder tool on a window from the applicaiton, then copy the MD5 hash from WindowFinder. This MD 5 value would then be put in a script then the `GetMD5` command used to compare the two MD5 hash values. If the MD5 hash values do not match, then the excutable file may have been changed.

5.2.34 GetReg

Table 5-34 Description of GetReg

Use With	All
SecureLogin Version	3.5
Type	Action
Usage	`GetReg <regentry> <variable>`

Arguments	`<regentry>`
	This is the registry entry to read.
	`<variable>`
	This variable defines where to store the retrieved environment variable data.

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` ValueValid 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""_`
	`?ProductIDIf ?ProductID noteq "xxxxxxxxxx"_`
	`#Not corporate desktop`
	`EndScript`
	`EndIf`

5.2.35 GetSessionName

Table 5-35 Description of GetSessionName

Use With	Terminal Emulator
SecureLogin Version	3.5
Type	Action
Usage	`GetSessionName <?variable>`
Arguments	`<Variable>` The target variable that the session name is copied into.
Description	Use the `GetSessionName` to find the current HLLAPI session name that is used to connect and returns it to the specified variable.

Example	Windows Application Definition `GetSessionName ?Session_name`

5.2.36 GetText

Table 5-36 Description of GetText

Use With	Web, Terminal Launcher
SecureLogin Version	3.0
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 a large Web Application Definition that might contain several If `-Text` statements. Under Netscape, each If -Text statement scans the screen to find the specified text, each scan of the screen results in the screen flashing. However, by 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 Log on. If Log on exists, SecureLogin enters the credentials and submits them automatically.
	`GetText ?Text`
	`If "Log on" -In ?Text`
	`Type $Username`
	`Type $Password Password`
	`EndIf`

5.2.37 GetURL

Table 5-37 Description of GetURL

Use With	Web
SecureLogin Version	3.0
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 applications. You will now be redirected to the Intranet home page."`
	`GoToURL "http://Intranet"`
	`EndIf`

5.2.38 GoToURL

Table 5-38 *Description of GoToURL*

Use with	Web
SecureLogin version	3.5.1
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 specify http:// before the URL.

Example	Web Application Definition
	
	This example detects an incorrect password message, displays a message box informing the user, and then browses the Novell Web site.
	`If -Text "Incorrect Password"`
	`MessageBox "You have entered an incorrect password"`
	`GoToURL "http://www.novell.com"`
	`EndIf`

5.2.39 If/Else/Endif

Table 5-39 Description of If/Else/Endif

Use with	Startup, Terminal Launcher and/or Windows
SecureLogin version	All
Type	Flow Control

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

Usage 2	`If <Value1> <Eq|NEq > <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 -exists <Variable>   #`
	`Do This`
	`[Else]   #`
	`Do This`
	`EndIf`

Arguments	`<Value1>`
	The left hand side of the expression for evaluation.
	`<Value2>`
	The right hand 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:** Evaluates to true if the left hand side is equal to the right hand side. ◆ **NEq:** Evaluates to true if the left hand side is not equal to the right hand side. ◆ **-In:** Evaluates to true if the left hand side is a substring of the right hand side. ◆ **-NotIn:** Evaluates to true if the left hand side is not a substring of the right hand side. 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 `NEq` operators are not case sensitive, while the `-In` and `-NotIn` operators are case sensitive. **Numerical Comparison Operators Supported** Two numerical comparison operators are supported by the If command Gt/Lt. The command evaluates to true if the left hand side is greater than/less than the right hand side. This is a numerical comparison, so the right hand side and left hand side must be numbers. An operator is supplied to check for the existence of a stored variable`:- Exists:` Evaluates to true if the specified variable exists. Finally, an operator is 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.

Syntax Examples	`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.

```
# Log on 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
# Log on 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 log on 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 –Exists $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.company.com/SecureLogin Details.htm”`
	`EndIf`
	`EndIf`

5.2.40 Include

Table 5-40 *Description of Include*

Use With	All
SecureLogin Version	3.0
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 logon dialog, the `notepad.exe` Application Definition is executed, and then the user's credentials are entered.
	`# Log on Dialog Box`
	`Dialog`
	`Class #32770`
	`Title “Log on”`
	`EndDialog`
	`Include Notepad.exe`
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

5.2.41 Increment/Decrement

Table 5-41 *Description of Increment/Decrement*

Use With	All
SecureLogin Version	All
Type	Variable Manipulator

Usage	`Increment <Variable>`
	
	`Decrement <Variable>`

Arguments	`<Variable>` The name of the variable to increase or decrease in value.
Description	Use the `Increment/Decrement` command to add or subtract from a specified variable. For example, you can use the increment and 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 logon 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.

Syntax examples	`Increment ?RunCount`
	
	`Decrement ?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.
	`#Log on 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`
	`# Log on Successful Message`
	`Dialog`
	`Ctrl #1`
	`Title "Log on Successful"`
	`EndDialog`
	`Set ?RunCount "0"`

5.2.42 KillApp

Table 5-42 *Description of KillApp*

Use With	All
SecureLogin Version	All
Type	Action
Usage	`KillApp <Process-Name>`
Arguments	`<Process-Name>` The name of the process to terminate.
Description	Use to terminate an application.

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.
	`#Log on 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`
	`# Log on Successful Message`
	`Dialog`
	`Title “Log on Successful”`
	`Ctrl #1`
	`EndDialog`
	`Set ?RunCount “0”`

5.2.43 Local

Table 5-43 *Description of Local*

Use with	All
SecureLogin Version	All
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 only exists 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 and not store 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 alerts the user, then closes the application.
	`# Invalid Log on Message`
	`Dialog`
	`Class #32770`
	`Title "Log on Failure"`
	`EndDialog`
	`Local ?RunCount`
	`Increment ?RunCount`
	`If ?RunCount Gt "5"`
	`MessageBox "Closing Application"`
	`KillApp "PasswordText.exe"`
	`EndIf`
	`Type $Username`
	`Type $Password`

5.2.44 MatchDomain

***Table 5-44** Description of MatchDomain*

Use With	Advanced Application Definitions created using the Web Wizard.
SecureLogin Version	3.5.1
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 http://www.yahoo.com/mymail/login
Example	This example the web site www.google.com is being matched by the Application Definition.

```
# === Login Application Definition #2 ==
# === Google Initial Login ====
#=========================================
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
```

5.2.45 MatchForm

Table 5-45 *Description of MatchForm*

Use With	Advanced Application Definitions created using the Web Wizard.
SecureLogin Version	3.5.1
Type	Action
Usage	`MatchForm #FormID [-optional] [-name "name"] [-action "action"] [-method "method"] [-target "target"]`

Arguments

`FormID`

The ID to be given to a matching form. The ID must be a static unsigned integer.

`-optional`

Specifies that matching this form is not required to successfully match site.

`-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 fails to match.

Example

In this example the form name “log on” within the web site www.google.com .com is being matched by the Application Definition.

```
# === Login Application Definition #2 ==
# === Google Initial Login ====
#=========================================
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 form name may be a “null”
MatchForm #1 -name “”
```

5.2.46 MatchField

Table 5-46 *Description of MatchField*

Use With	Advanced Application Definitions created using the Web Wizard.
SecureLogin Version	3.5.1
Type	Action
Usage	`MatchField #FormID:FieldID [-optional] [-name "name"] [-type "type"] [-value "value"] [-defaulValue "defaultValue"]`

Arguments	`FieldID`
	
	The ID to be given to the matched field. The ID must be a static unsigned integer.
	
	`-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
	◆ File
	◆ Image
	◆ Hidden
	◆ Password
	◆ Radio
	◆ Reset
	◆ Submit
	◆ Text
	◆ Select-multiple
	◆ Select-one
	
	`-value "value"`
	
	Match against the field value.
	
	`-defaultValue "defaultValue"`
	
	Match against the fields default value.

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 fails to match.

Example	In this example the web site fields Email, Password and Cookie within the web site www.google.com .com are being matched by the Application Definition.
	`# === Login Application Definition #2 ==`
	`# === Google Initial Login ====`
	`#========================================`
	`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`

5.2.47 MatchForm

Table 5-47 *Description of MatchForm*

Use With	Advanced Application Definitions created using the Web Wizard.
SecureLogin Version	3.5.1
Type	Action
Usage	`MatchForm #FormID [-optional] [-name "name"] [-action "action"] [-method "method"] [-target "target"]`

Arguments	`FormID`
	The ID to be given to a matching form. The ID must be a static unsigned integer.
	`-optional`
	Specifies that matching this form is not required to successfully match site.
	`-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 fails to match.
Example	In this example the form name “log on” within the web site www.google.com .com is being matched by the Application Definition.

```
# === Login Application Definition #2 ==
# === Google Initial Login ====
#=========================================
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 form name may be a “null”
MatchForm #1 -name “”
```

5.2.48 MatchOption

Table 5-48 *Description of MatchOption*

Use With	Advanced Web Application Definitions created using the Web Wizard.
SecureLogin Version	3.5.1 to 6.0
Type	Action
Usage	`MatchOption #FormID:FieldID:OptionID [-optional] [-text "text"] [-value "value"]`
Arguments	**OptionID** The ID to be given to the specific option within the given field. The ID is a static, unsigned integer. -**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.
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	This example the form name “log on” within the secure web site www.lotto.com .com is being matched by the Application Definition.

```
# === Login Application Definition #4 ==
# === Lotto User Initial Login ====
#========================================
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 -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
```

5.2.49 MatchReferer

Table 5-49 *Description of MatchReferer*

Use With	Advanced Web Application Definitions created using the Web Wizard.
SecureLogin Version	3.5.1 to 6.0
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	This example the referring HTML page “www.lotteries/index.html” is being matched by the Application Definition.
	`# === Login Application Definition #5 ==`
	`# === Lotto User Initial Login ====`
	`#========================================`
	`Site Login -userid “Member Log In” -initial`
	`MatchForm #1 -name “log in”`
	`MatchReferer “www.Lotteries.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`

5.2.50 MatchURL

Table 5-50 *Description of MatchURL*

Use With	Advanced Web Application Definitions created using the Web Wizard.
SecureLogin Version	3.5.1
Type	Action
Usage	`MatchURL "URL"`

Arguments	`MatchURL`
	Used inside a `Site` block, `MatchURL` is used to filter a Site based on its URL. If the URL doesn't match, the site block fails to match.
	`"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.

Description	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.

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`

5.2.51 MessageBox

Table 5-51 *Description of Message Box*

Use With	Startup, Terminal Launcher, Web and Windows
SecureLogin Version	All
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 top most 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 -YesNo and `-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 been 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. Use a runtime `<?Variable>` to capture the MessageBox result immediately after the flag. The variable value is set to Yes, No, or Cancel.

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 elected not to change your
password."
EndIf
```

Example 2	Terminal Launcher 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 if failing.

The `WaitForText` cuts off the first character because it finds both Password and password, and responds to all password entry points.

```
MessageBox "Beginning wait for Log on prompt"
WaitForText "ogin:"
MessageBox "Log on detected, now entering
Username"
Type $Username
MessageBox "Username entered, now simulating
Enter"
Type @E
MessageBox "Enter has been simulated. Now
waiting for?
Password"WaitForText "password:"
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"
```

5.2.52 Multiply

Table 5-52 *Description of Multiply*

Use With	All
SecureLogin Version	3.0
Type	Variable Manipulator
Usage	`Multiply <Variable1> <Variable2> [?Result]` **NOTE:** You must use integer arithmetic.

Arguments	`<Variable1>`
	
	The multiplicand, the first argument, is the number multiplied 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 multiplier, the second argument, is the number by which the first number is multiplied. `<Variable2>` can be a SecureLogin variable or numeric value.
	
	`[?Result]`
	
	Optional, the product, or result of the equation.

Description	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.

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`

5.2.53 OnException/ClearException

Table 5-53 Description of OnException/ClearException

Use With	All
SecureLogin Version	3.0.4
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. Currently two exceptions are supported:
	1. `ChangePasswordCancelled`. When a user clicks Cancel on the Change Password dialog box.
	2. `EnterVariablesCancelled`. When a user clicks Cancel on the automatic variable prompt dialog box.
	
	`<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 pressed 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`
	
	`ErrorClearException ChangePasswordCancelled`

Example 1	Windows Application Definition In this example the log on failed because the user has invalid credentials stored. This provides the user with an opportunity to verify their username 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.

```
# Log on Failed Dialog Box
Dialog
Class #32770
Title "Log on Failed"
EndDialog
OnException EnterVariablesCancelled Call
Variables Cancelled
DisplayVariables "Please verify your Username
and Password and try again.  Helpdesk x5555."
ClearException EnterVariablesCancelled
Type $Username #1001
Type $Password #1002
Click #1
Sub VariablesCancelled
OnException EnterVariablesCancelled Call
Variables Cancelled
Display Variables "You cannot cancel this
verification dialog box. Please verify your
Username and Password when prompted and click OK
to retry log on."
ClearException EnterVariablesCancelled
EndSub
```

Example 2	Windows Application Definition
	
	This example prompts the user to change their password. SecureLogin must handle password changes so the password is updated both in the application and in the user's 3DES encrypted store in the directory against their user object.
	`# 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 x5555”`
	`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 x5555”`
	`Type $Password #1007`
	`Type $Password #1008`
	`ClearException ChangePasswordCancelled`
	`EndSub`

5.2.54 Parent/EndParent

Table 5-54 *Description of Parent/EndParent*

Use With	Windows
SecureLogin Version	All
Type	Dialog Specifier

Usage	`ParentEnd`
	
	`Parent`

Arguments	None

Description	Use the `Parent` 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 Logon 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 dialogs parent. For example, it is possible to get a Application Definition to click a button on the parent window. An example of this use is shown in Example 2. **EndParent Command** 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 work, try using the `Parent` command. It is able to handle windows that are within windows, and so on.

Example 1	Windows Application Definition
	
	This example specifies the dialog box that is used for log on. In this case, the parent of the logon box has a class of "Centura:MDIFrame".
	
	`#Log on Dialog BoxDialog`
	`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 Logon windows parent.
	`# Log on Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Log on"`
	`EndDialog`
	`Type $Username #1001`
	`Type $Password #1002`
	`Parent`
	`Click #1`
	`EndParent`

5.2.55 PickListAdd

Table 5-55 *Description of PickListAdd*

Use With	All
SecureLogin Version	All
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 `PickList` 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 the `PickList` command to choose from multiple sessions on one mainframe account. In fact, use the `PickList` to build a list of databases, phone numbers, or any list from which your 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.

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

5.2.56 PickListDisplay

Table 5-56 *Description of PickListDisplay*

Use With	Startup, Terminal Launcher, Web and/or Windows
SecureLogin Version	All
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.
	
	`-NoEdit`
	
	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`
	
	`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.
	`# Log on 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 ?AccountType $Username #1001`
	`Type $Password #1002`
	`Click #1`

5.2.57 PositionCharacter

Table 5-57 *Description of Position Character*

Use With	Password Policy Application Definitions
SecureLogin Version	All
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

5.2.58 PressInput

Table 5-58 Description of PressInput

Use With	Advanced Web Application Definitions created using the Web Wizard.
SecureLogin Version	3.5.1
Type	Action
Usage	`PressInput [#FormID:FieldID [-press "press"]]`
Arguments	`PressInput` Simulates a keyboard enter event. Optionally focusing a given field beforehand. `-press "pr`ess" Description Simulates pressing the keyboard enter key.ess"

Example	This example the `PressInput` command within the Application Definition is the equivalent of clicking the Sign On button on the www.google.com web site.
	`# === Login Application Definition #2 ==`
	`# === Google Initial Login ====`
	`#========================================`
	`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`

5.2.59 ReadText

Table 5-59 Description of ReadText

Use With	Terminal Launcher, Windows. This command applies specifically to HLLAPI, WinHLLAPI and HLLAPI 16 terminal emulators.

SecureLogin Version	All
Type	Action
Windows Usage	`ReadText <#Ctrl-ID> <?Variable>`
Terminal Launcher Usage	`ReadText <?Variable> <Character-Number> <Row-Number>` `<Column-Number>`

Arguments	`<#Ctrl-ID>`
	The control ID number of the text to read.
	`<?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	Use the `ReadText` command to run in both Windows and Terminal Launcher 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 Section 5.2.39, “If/Else/Endif,” on page 93 and Section 5.2.36, “GetText,” on page 91.

Example 1	HLLAPI emulator `Readtext ?result "X" "Y" "Z"` X = The number of characters to read. Y= The row from which the characters are read. Z= The column from which the characters are read
Example 2	Windows script `ReadText #1004 ?result`

Syntax examples	`ReadText #301 ?Text`
	
	`ReadText ?Text 4 6`

Example 1	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.
	`# Log on 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 Helpdesk on x3849."`
	`Endscript`
	`EndIf`

Example 2	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.
	`# Log on 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 3	Terminal Launcher Application Definition
	
	This example reads a message in a Terminal Emulator and displays the message in a user friendly format.
	`ReadText ?Message 30 24 2`
	`MessageBox ?Message`

5.2.60 RegSplit

Table 5-60 *Description of RegSplit*

Use With	All
SecureLogin Version	All
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.

Example

Windows Application Definition

This example copies text from Control ID #301 to the `?Text` variable. The `RegSplit` command is then used to strip the username details out of the text that was read. The platform is set to that username, and the correct password is entered by SecureLogin.

```
# Log on 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
```

Open Text Example	`#?InputString: "This is a long string with a few components in it"`
Command	`RegSplit "This (.*) a long (.*) with (.*) components (.*)" ?InputString ?First ?Second ?Third ?Fourth`
Result	`?First = "is", ?Second = "string", ?Third = "a few", ?Fourth = "in it"`

5.2.61 ReLoadPlat

When an application first presents a logon screen, SecureLogin displays a message box 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.

Resolving the Issue of Reentering User ID Details

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 reenter 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 not previously selected, the command leaves it unset.

For more information see Section 5.2.61, “ReLoadPlat,” on page 124 and Section 5.2.12, “ClearPlat,” on page 67.

***Table 5-61** Description of ReLoadPlat*

Use With	Startup, Terminal Launcher, Web and/or Windows
SecureLogin Version	All
Type	Action

Usage	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.
Arguments	None
Description	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.
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 ====
```

Example (Cont.)	
	```
	# ==== 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
	EndScriptEndSub
	# ==== EndSection: Change Credentials
	Cancelled ===
	```

5.2.62 Repeat/EndRepeat

Table 5-62 *Description of Repeat/EndRepeat*

Use With	All
SecureLogin Version	All
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 repeat loop in the Application Definition.

```
# Initial System Log on
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."
```

Example (Cont.)

```
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
EndRepeat
```

5.2.63 RestrictVariable

Table 5-63 *Description of RestrictVariable*

Use With	All
SecureLogin Version	All
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>`. Any variable specified must match the policy or it is not saved. 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 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#
Log on 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 restricts 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`
	`# Log on 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 ?NewPwd Random`
	`Type ?NewPwd #1005`
	`Type ?NewPwd #1006`
	`Set $Password ?NewPwd`
	`Click #1`

5.2.64 Run

Table 5-64 *Description of Run*

Use With	Startup, Terminal Launcher, Web and/or Windows
SecureLogin Version	All
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
	
	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`

5.2.65 SelectListBoxItem

Table 5-65 *Description of SelectListBoxItem*

Use With	Advanced Web Application Definitions
SecureLogin Version	All
Type	Action
Usage	`SelectListBoxItem <Text of Item to set to> [<#Item Number>] [<-multiselect>]`

Arguments	`<Text of Item to set to>`
	
	The text item that you want SecureLogin to select in the list box.
	
	`<#Item Number>`
	
	When multiple list boxes are found, this specifies which list box to address.
	
	`<-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 instruction on determining item numbers, see Section 5.2.24, “EndScript,” on page 82.

Example	`SelectListBoxItem "Remember Defects" #2 -multiselect`
	
	`SelectListBoxItem "Remember Enhancements" #2 -multiselect`

5.2.66 SendKey

***Table 5-66** Description of SendKey*

Use With	Terminal Launcher
SecureLogin Version	All
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. **About 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 Section 5.2.85, “Sending Keyboard Commands Using Type,” on page 155 and Section 5.2.84, “Type,” on page 152. For more information on these functions, see Chapter 7, “Reference Commands and Keys,” on page 165.

Example	Terminal Launcher Application Definition
	
	The example sends the username and password to the terminal emulator.
	`#Send Username`
	`SendKey "DJones"`
	`SendKey "\N"#`
	`Send Password`
	`SendKey "Hu7%f"`
	`SendKey "\N"`

5.2.67 Set

Table 5-67 *Description of Set*

Use With	All
SecureLogin Version	All
Type	Action
Usage	`Set <Variable> <Data>`

Arguments	`<Variable>`
	The variable to which the data is being assigned.
	`<Data>`
	The text or variable read from and assigned to the variable.

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.
	`# Log on 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.
	`# Log on 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 ?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 `Ctrl #15`, and sets the `$Database` variable so the user does not have to set the variable.
	`# Log on Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Log on"`
	`EndDialog`
	`ReadText #15 ?Database`
	`If -Exists $Database`
	`Else`
	`Set $Database ?Database`
	`EndIf`

5.2.68 SetCheckBox

Table 5-68 *Description of SetCheckBox*

Use With	Advanced Web Application Definition
SecureLogin Version	3.5
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	see below

```
Messagebox "Scroll down so you can see the
'Search Language' section and all the Languages
with the check boxes then click OK on this
messagebox"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"elseset
?cmd37 "Setcheckbox failed"
endifset
checkbox #1 "unchecked"
setcheckbox #2 "unchecked"
setcheckbox #3 "unchecked"
setcheckbox #4 "unchecked"
setcheckbox #26 "unchecked"
setcheckbox #27 "unchecked"
Messagebox "Did it clear all the languages
except Norwegian" -yesno ?
advweb2
if ?advweb2 eq yes
set ?cmd38 "setcheckbox command worked"
else
set ?cmd38 "setcheckbox failed"
endif
```

5.2.69 SetCursor

Table 5-69 *Description of SetCursor*

Use With	Terminal Launcher (Only available in HLLAPI and some DDE emulators)
SecureLogin Version	All
Type	Action
Usage 1	`SetCursor <Screen-Position>`
Usage 2	`SetCursor <X Co-ordinate> <Y Co-ordinate>`

Arguments	`<Screen-Position>`
	
	The position on the screen to move the cursor.
	
	`<X Co-ordinate>`
	
	The horizontal coordinate. When specified, a row or column conversion is carried out before the cursor is set to the position.
	
	`<Y Co-ordinate>`
	
	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 `<ScreenPosition>` or `<X Co-ordinate>` `<Y Co-ordinate>`. 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 Launcher 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`

5.2.70 SetFocus

Table 5-70 *Description of SetFocus*

Use With	Java, Web, Windows
SecureLogin Version	All
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 `<#Ctrl-ID>`. A valid `<#Ctrl-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.
	`# Log on Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Log on"`
	`EndDialog`
	`SetFocus #1001`
	`Type $Username`
	`Type \T`
	`Type $Password`
	`Type \N`

5.2.71 SetPlat

Table 5-71 *Description of SetPlat*

Use With	All
SecureLogin Version	All
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` (For more information see Section 5.2.61, "ReLoadPlat," on page 124)
	
	`<#Ctrl-ID>`
	
	The control ID number of the regular expression.

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

Web Application Definition

A dialog box appears when you try to access a password-protected site using Netscape Navigator.

When you specify the Title, Class, Username, and Password fields for this dialog box they are always the same. If you stored the Username and Password against this platform without using the `SetPlat` command, the same Username and Password for www.serversystems.com is entered to log on to any site (and are obviously invalid for any other site).

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
```

Example 1 (Contd)	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 username for Control Panel at www.serversystems.comNext, 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 When the user has run the Application Definition, they will see the Username 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.
	`# log on 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`

5.2.72 SetPrompt

Table 5-72 *Description of SetPrompt*

Use with	All
SecureLogin Version	All
Type	Action
Usage	`SetPrompt <Prompt-Text>`
Arguments	`<Prompt-Text>` The customized text prompt displayed in the Enter SecureLogin Variables dialog box.

Description	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 Section 5.2.21, “DisplayVariables,” on page 79. **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, and places the `SetPrompt` command at the bottom of the Application Definition.
	
	`# Log on Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Log on"`
	`EndDialog`
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`
	`SetPrompt "Please enter your Username 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 Helpdesk x4564"`

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.
	
	`# Log on Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Log on"`
	`EndDialog`
	`SetPrompt "Enter Username==>"`
	`Type $Username #1001`
	`SetPrompt "Enter Password==>`
	`"Type $Password #1002`
	`Click #1`
	`SetPrompt "Please enter your Username 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 Helpdesk x4564"`

5.2.73 Site/EndSite

Table 5-73 *Description of Site/EndSite*

Use With	Advanced Web Application Definitions created using the Web Wizard.
SecureLogin Version	3.6.1
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/End-Dialog` 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.
	
	`-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.

Description	`SIte/EndSite` begins and ends an Application Defintion, in place of `Dialog/EndDialog`. The `Site/EndSite` commands have been added to allow for much finer control of web site matching. No longer is a URL all that can be matched on. Detailed information of the loaded web site can now be matched upon and used to execute blocks of scripting commands. `Site/EndSite` blocks are used to define all the parameters SecueLogin would expect to find on a web page to run the applicxation 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 would locate the web site www.mybank.com.
	`# === My Bank Initial Login ===`
	`Site "www.mybank.com" -userid "My Login Credentials"`
	`-initial`
	`EndSite`

Example 2	This simple example would locate the web site www.google.com, locate the login form and log on to the users account using the users email address, account number and password.
	`# === Login Application Definition #2 ==`
	`# === Google Initial Login ====`
	`#=========================================`
	`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`

5.2.74 StrCat

Table 5-74 *Description of StrCat*

Use With	All
SecureLogin Version	All
Type	Action

Usage	`StrCat <Variable> <Input-String1> <Input-String2>`

Arguments	`<Variable>`
	The variable to which you want to 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 username from #1001 into ?Username and uses the `StrCat` command to join the username onto the password. The result is a LogonID, which SecureLogin uses to log on to the system.
	`# Log on Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Log on"`
	`EndDialog`
	`ReadText #1001 ?Username`
	`StrCat ?LoginID ?Username $Password`
	`Type ?LoginID #1002`
	`Click #1`

5.2.75 StrLength

Table 5-75 *Description of StrLength*

Use With	All
SecureLogin Version	3.0.4
Type	Variable Manipulator
Usage	`StrLength <Destination> <String>`

Arguments	`<Destination>`
	The output variable. Also the input variable if no source is specified.
	`<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.
	
	`# Log on 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`

5.2.76 StrLower

Table 5-76 *Description of StrLower*

Use with	All
SecureLogin Version	3.0.4
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 username from #1001 and copies it into `?Username`. The `StrLower` command is then used to make sure the username is all lower case.
	`# Log on Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Log on"`
	`EndDialog`
	`ReadText #1001 ?Username`
	`StrLower ?LowerCaseUsername ?Username`
	`Type ?LowerCaseUsername #1002`
	`Click #1`

5.2.77 StrUpper

Table 5-77 *Description of StrUpper*

Use With	All
SecureLogin Version	3.0.4
Type	Variable Manipulator

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 username from #1001 and copies it into `?Username`. The `StrUpper` command is then used to make sure the username is all upper case.
	`# Log on Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Log on"`
	`EndDialog`
	`ReadText #1001 ?Username`
	`StrUpper ?UpperCaseUsername ?Username`
	`Type ?UpperCaseUsername #1002`
	`Click #1`

5.2.78 Sub/EndSub

Table 5-78 *Description of Sub/EndSub*

Use With	Startup, Terminal Launcher, Web and/or Windows
SecureLogin Version	3.5.1
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. You can also call a subroutine using the `Call` command. For more information, see Section 5.2.10, “ChangePassword,” on page 64.

Example	Terminal Launcher 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 before the next part of the Application Definition.
	`If -Text "Log on"`
	`Call "Log on"`
	`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`

5.2.79 Submit

Table 5-79 *Description of Submit*

Use With	Web
SecureLogin Version	All
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 Netscape.
	
	The function performed by the `Submit` command is automatically performed by Web Application Definition by default. For example, the Application Definition:
	
	`Type $Username`
	
	`Type $Password Password`
	
	Types the username 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, Submit,` or `Click`. If any of these commands are used, you must use the `Submit` command or some other means to submit the form.
	
	Furthermore, an automatic submit does not occur if you type text into a specific text entry field. For example, in the Application Definition segment 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 username and password and then executes a manual Submit.
	`Type $Username #1`
	`Type $Password #2`
	`Submit`

5.2.80 Subtract

Table 5-80 *Description of Subtract*

Use With	Startup, Terminal Launcher, Web and/or Windows
SecureLogin Version	3.0
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. 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 the `Add, Divide`, and `Multiply`.

For more information see Section 5.2.4, "Attribute," on page 59 and Section 5.2.22, "Divide," on page 81.

NOTE: The `Subtract` command correctly subtracts when `<StartValue>`, `<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
```

5.2.81 Tag/EndTag

Table 5-81 Description of Tag/End Tag

Use With	Advanced Web Application Definition
SecureLogin Version	All
Type	Tag Specifier
Usage	`Tag`
	`EndTag`
Arguments	None
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`

5.2.82 TextInput

Table 5-82 Description of TextInput

Use With	Advanced Web Application Definitions created using the Web Wizard.
SecureLogin Version	3.5.1
Type	Action
Usage	`TextInput #FormID:FieldID -value "value"`
Arguments	`#FormID`
	The ID to be given to the matched form. The ID must be a static unsigned integer.
	`#FieldID`
	The ID to be given to the matched field. The ID must be a static unsigned integer.
	`-value "value"`
	The text value to be input.
Description	Used inside a Site block to input text into a specied field.

Example	In this example the text value of the system username and Password are passed to the Application Definition.
	`# === Login Application Definition #2 ==`
	`# === Google Initial Login ====`
	`#========================================`
	`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`

5.2.83 Title

Table 5-83 *Description of Title*

Use With	Java, Windows
SecureLogin Version	All
Type	Dialog Specifier
Usage	`Title <Window-Title>`
Arguments	`<Window-Title>` The text to test against the window title.

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. Always place the `Title` command after all other commands in the `Dialog` block. **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 Section 5.2.11, "Class," on page 66 and Section 5.2.16, "Ctrl," on page 74. **NOTE:** Use the SecureLogin Window Finder tool to determine the window title.

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

5.2.84 Type

Table 5-84 Description of Type

Use With	Java, Terminal Launcher, Web and/or Windows
SecureLogin version	All
Type	Action
Terminal Usage	`Type [-Raw] <Text>`

Windows Usage	`Type <Text> [<#Ctrl-ID>]`
	
	`Type [-Raw] <Text>`

Web Usage	`Type <Text> [<#Field-ID>]`
	
	`Type <Text> ["password"]`

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. Further, instead of trying to set the text in the field directly, this option simulates actual keystrokes, causing SecureLogin to type into whichever window has focus.
	
	`<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 Windows Finder Tool to extract these control IDs. For more information, see *Windows Specific* on page 151.
	
	`[<#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 151.
	
	`[password]`
	
	For Web Application Definitions, this optional argument specifies to perform this type into the password field on this form. If `[password]` is used, that application's Application Definition cannot use a `<#Ctrl-ID>` argument. For more information, see *Web Specific* on page 151.

Description

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 Section 5.2.66, "SendKey," on page 132.

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 Windows Finder Tool. SecureLogin Single Sign-On 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` option, then you cannot use the `<#Ctrl-ID>` argument.

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 `<#FieldID>` 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 position 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. See *Web Application Definition* on page 153.

Example 1

Windows Application Definition

This example is a typical use of the `Type` command in a Windows Application Definition.

```
# Log on 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 switch. This switch is not actually required in this instance, and is only there as an example.
	`# Calculator Is Active`
	`Dialog`
	`Class #SciCalc`
	`Title "Calculator"`
	`EndDialog`
	`Type -Raw "15"`
	`Type -Raw "+"`
	`Type -Raw "20"`
	`Type -Raw "="`

Example 3	Web Application Definition
	
	This example uses the SecureLogin agent to automatically generate this Application Definition for the mail.yahoo.com site. This example shows the use of Password as the [<Field Name>] argument.
	`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`

5.2.85 Sending Keyboard Commands Using `Type`

SecureLogin can send special keyboard keystrokes to Windows and Internet based applications to emulate the user's keyboard entry. The `Type` command can pass keystrokes through to the window that the Application Definition is working. These special commands include the ability to select Menu items, send ALT, and send other keyboard combinations.

Special Key Commands

Table 5-85 *Description of 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>.

Type	Simulates
`\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. For more information on available keyboard sequences you can use with the `Type` command see Section 7.1, "Windows Keyboard Functions," on page 165.

Table 5-86 *Description of Raw Key Commands*

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

Terminal Launcher uses the High Level Language Application Programming Interface (HLLAPI) to interface with a wide range of mainframe emulators that implement this programming standard. The list are the `@` commands that you can use in the SecureLogin Application Definition Type command. These commands perform specific emulator and mainframe functions. For example, you can send an ENTER, TAB, or cursor key or issue a mainframe emulator print screen or reset function.

The @ commands are used in Application Definition language in the following format:

- `TYPE @ command`
- `WAITFORTEXT "Log on:"`
- `Type $username`
- `Type @T`
- `Type $password`
- `Type @E`

For more information on the available terminal emulator commands that you can use within a terminal emulator Application Definition see Chapter 7, "Reference Commands and Keys," on page 165.

5.2.86 WaitForFocus

Table 5-87 *Description of WaitForFocus*

Use With	Windows
SecureLogin Version	All
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 figure to a negative number (for example `WaitForFocus "#1065" "-1"`) for the `<Repeat-Loops>` never to expire. If the `<#Ctrl-ID>` is set to 0 (zero), it loops until the window defined in the `Dialog/ EndDialog` statement is given keyboard focus. **NOTE:** Do not place `WaitForFocus` commands within `Dialog / EndDialog` statements.

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

Example	Windows Application Definition
	
	This example has the SecureLogin waiting indefinitely for window #301 to get focus. Once the Logon dialog box is detected, it enters the user credentials.
	`# Log on Dialog Box`
	`Dialog`
	`Class #32770`
	`Title "Log on"`
	`EndDialog`
	`WaitForFocus #301 "-1"`
	`Type $Username`
	`Type \T`
	`Type $Password`
	`Type \N`

5.2.87 WaitForText

Table 5-88 *Description for WaitForText*

Use With	Terminal Launcher
SecureLogin Version	All

Type	Flow Control
Usage	`WaitForText <Text>`
Arguments	`<Text>` The text for which the Application Definition is waiting.
Description	Use the `WaitForText` command so the Terminal Launcher waits for the specified `<Text>` to display before continuing. This command allows the user to wait for particular text to display before continuing. For example, waiting for a username field to display before attempting to type a username. 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. If `WaitForText` is not working, try leaving the initial letter off the `<Text>` to avoid any conflict with case sensitivity. For example, `WaitForText` ogin will work regardless of whether the word log on is presented on the terminal screen as Log on or log on. However, `WaitForText "Log on"` will only work if the word log on is presented on the screen as "Log on". 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 uses the SecureLogin to wait for the text ogin: to appear on the emulator screen before entering the username. It will then wait for assword: to display before entering the password.
	
	`WaitForText "ogin:"`
	`Type $Username`
	`Type @E`
	`WaitForText "assword:"`
	`Type $Password`
	`Type @E`

Testing Application Definitions 6

This section contains the following information:

- Section 6.1, "Using the SecureLogin Test Application," on page 159

6.1 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 on
- Wrong password
- Password change

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

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

6.1.1 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`, per 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
Application Definition continued on the next page
# ==== 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 Username and Password to access NSL Test.
SecureLogin will remember and automatically log you on in future. IT
Helpdesk x4546"
# ==== 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 PSL Test Application failed. The password
for this app must be single when it first starts up. IT Helpdesk x4563"
# 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 Username and Password, then start
the change password routine
Application Definition continued on the next page
Set ?PwdBackup $Password
Type $Username #1015
Type $Password #1004
ChangePassword ?NewPwd "Please enter a new password for the
application."
Type ?NewPwd #1005
Type ?NewPwd #1006
Click #1
# Change Password Successful message
Dialog
Class "#32770"
Ctrl #65535 "You have changed the password successfully."
Title "Change Successful"
EndDialog
# Clear Application owned message and accept new password
Click #2
Set $Password ?NewPwd
# ==== End Section: Change Password ====
```

6.1.2 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 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 will run when other dialog boxes owned by the same executable with the same information appear.

When SecureLogin detects all the information between `Dialog` and `EndDialog` is contained in the dialog box on the screen (for example, the application logon 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.

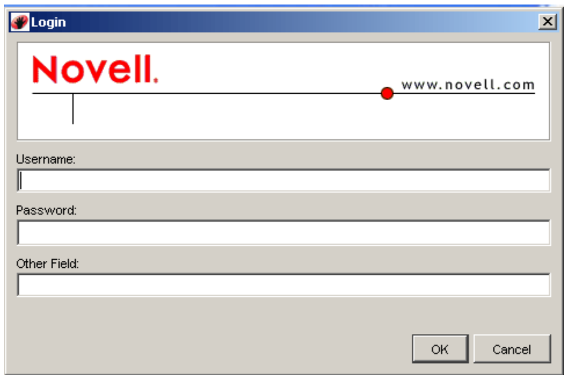
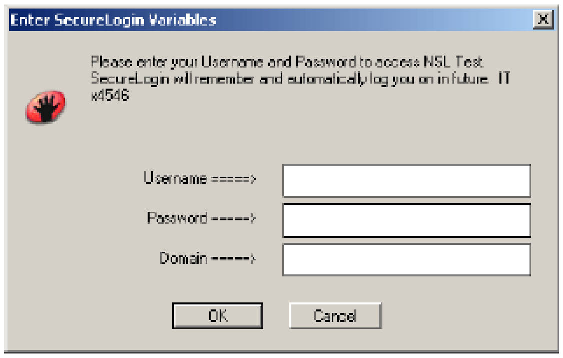
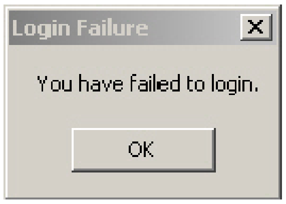
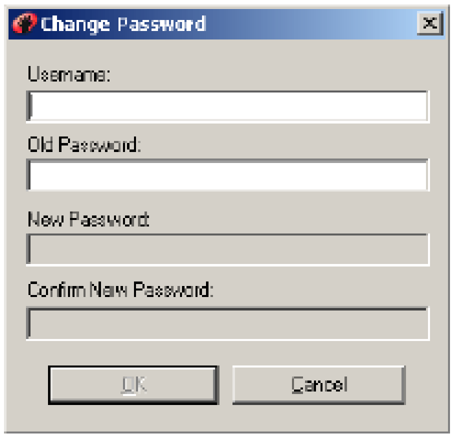
6.1.3 Dialog Boxes

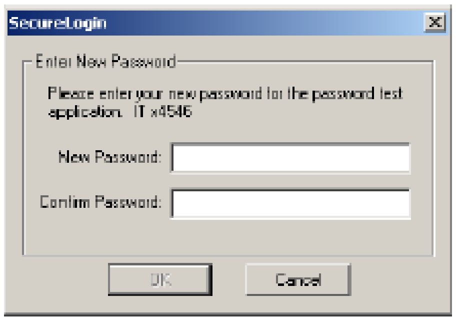
The following Application Definition example shows screen captures of the relevant dialog boxes. You can use the Window Finder tool to gather information about the title of the window, class names, dialog IDs, and so on. Use the wizard to automate the Application Definition creation.

Table 6-1 *Description*

Application Definition SectionComments	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 will watch 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.

`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 they have no credentials stored.
	
	When the user first runs a newly single sign-on enabled application, SecureLogin will prompt for their logon credentials, and store and remember them for future log on attempts.

Application Definition SectionComments	Comments
 Login Novell. www.novell.com Username: Password: Other Field: OK Cancel	Title is Log on. Class is #32770. Username field is Control ID #1001. Password field is Control ID #1002. Other field is Control ID #1003. The OK button is Control ID #1.
 Enter SecureLogin Variables Please enter your Username and Password to access NSL Test SecureLogin will remember and automatically log you on in future. IT x4546 Username =====> Password =====> Domain =====> OK Cancel	This dialog box is only displayed the first time the Application Definition is run by a user. It prompts the user to enter their credentials for SecureLogin to store them. The `SetPrompt` command is used throughout the example Application.
 You have failed to login.	This is the logon failure dialog box. Title is logon Failure. Class is #32770. The OK button is Control ID #2. The error message is Control ID #65535
 Change Password Username: Old Password: New Password: Confirm New Password: OK Cancel	This is the change password dialog box. Username field is Control ID #1015. Old Password field is Control ID #1004. New Password field is Control ID #1005. Confirm field is Control ID #1006. The OK button is Control ID #1.

Application Definition SectionComments	Comments
SecureLogin Enter New Password Please enter your new password for the password test application. IT x4546 New Password: Confirm Password: 	The `ChangePassword` command is used in the example Application Definition to display a dialog box for the user to enter their new password. The dialog box is customized to provide more information for the user.

Reference Commands and Keys

7

This section contains the following information:

- Section 7.1, "Windows Keyboard Functions," on page 165

7.1 Windows Keyboard Functions

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

For more information about the Type command, see Section 5.2.84, "Type," on page 152.

7.1.1 Example for Typing Keys

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 7-1 *Example Typing Keys*

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

7.1.2 Table of Windows Keyboard Functions

Table 7-2 *Enter Table Title Here*

Function	Decimal	Comment
Left Mouse button	1	
Right Mouse button2	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

Function	Decimal	Comment
Enter	13	
Shift	16	
CTRL	17	
ALT	18	
Pause	19	
Cap Lock	20	
Escape	27	
Space	32	
Page Up	33	
Page Down	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	

Function	Decimal	Comment
A	65	
B	66	
C	67	
D	68	
E	69	
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	

Function	Decimal	Comment
Keypad 2	98	
Keypad 3	99	
Keypad 4	100	
Keypad 5	101	
Keypad 6	102	
Keypad 7	103	
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	

Function	Decimal	Comment
F19 Key	130	
F20 Key	131	
F21 Key	132	
F22 Key	133	
F23 Key	134	
F24 Key	135	
Num Lock Key	144	
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 +

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

7.1.3 Terminal Emulator Command Reference

The following table lists the terminal commands in Terminal Emulator Application Definitions.

Table 7-3 *Terminal Emulator Command Reference*

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 Spaces)
@O	Space	@A@T	Print Presentation Space

The Type Command	Meaning	The Type Command	Meaning
@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
@3	PF3/F3	@A@i	Turquoise
@4	PF4/F4	@A@l	Reset Host Colours
@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

The Type Command	Meaning	The Type Command	Meaning
@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	SrcLk (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
@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

Application Definition Commands for SNMP Alerts

8

SecureLogin produces SNMP (Simple Network Management Protocol) for network monitoring software to trap. A simple application definition command is used to send the alerts.

NOTE: You may have to copy the *LIBSNMP.DLL* file to the Windows\System32 directory for SNMP support to work.

8.1 Create the 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:

```
Run C:\Progra~1\Novell\Secure~1\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.

8.2 Example

The following is an example Application Definition:

```
Dialog
Class #32770
Title "Incorrect Password"
EndDialog
Run C:\Progra~1\Novell\Secure~1\Slsnmp.exe SNMPCommunity1
192.168.156.23 "PSL - Incorrect password in finance system."
MessageBox "You have entered an incorrect password.  The administrator
has been notified.  Please restart the application and try again."
KillApp "PasswordText.exe"
```

Documentation Updates

A

This section lists updates to the Novell SecureLogin 6.0 Application Definition Guide that have been made since the initial release of Novell SecureLogin 6.0.

The information helps you to keep current on documentation updates and, in some cases, software updates (such as a Support Pack release).

The information is grouped according to the date when the Novell SecureLogin 6.0 Application Definition Guide was republished. Within each dated section, the updates are listed by the names of the main table of contents sections.

The Novell SecureLogin 6.0 Application Definition Guide has been updated on the following dates:

- Section A.1, "December 12, 2006," on page 175
- Section A.2, "October 13, 2006," on page 175

A.1 December 12, 2006

Location	Description
Section 5.2.17, "DebugPrint," on page 75	Description of `DebugPrint` command included.
Section 5.2.18, "Decrement," on page 76	Description of `Decrement` command included.
Section 5.2.48, "MatchOption," on page 106	Description of `MatchOption` command included.
Section 5.2.49, "MatchReferer," on page 107	Description of `MatchReferer` command included.

A.2 October 13, 2006

Location	Description
Section 5.2.1, "Regular Expressions," on page 55	A section with information on Regular Expressions included.