

# Novell Nsure™ SecureLogin

3.51.2

[www.novell.com](http://www.novell.com)

SCRIPTING GUIDE

April 08, 2005



Novell®

## Legal Notices

Novell, Inc. makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Novell, Inc. makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

You may not use, export, or re-export this product in violation of any applicable laws or regulations including, without limitation, U.S. export regulations or the laws of the country in which you reside.

Copyright © 2005 Novell, Inc. All rights reserved. No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of the publisher.

Novell, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.novell.com/company/legal/patents/> and one or more additional patents or pending patent applications in the U.S. and in other countries.

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

[www.novell.com](http://www.novell.com)

Nsure SecureLogin 3.51.2 Scripting Guide

April 08, 2005

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

## **Novell Trademarks**

BorderManager is a registered trademark of Novell, Inc. in the United States and other countries.

ConsoleOne is a registered trademark of Novell, Inc. in the United States and other countries.

eDirectory is a registered trademark of Novell, Inc. in the United States and other countries.

GroupWise is a trademark of Novell, Inc.

NDS is a registered trademark of Novell, Inc. in the United States and other countries.

NMAS is a trademark of Novell, Inc.

Novell is a registered trademark of Novell, Inc. in the United States and other countries.

Novell iChain is a registered trademark of Novell, Inc. in the United States and other countries.

Novell iFolder is a registered trademark of Novell, Inc. in the United States and other countries.

Novell SecretStore is a registered trademark of Novell, Inc. in the United States and other countries.

ZENworks is a registered trademark of Novell, Inc. in the United States and other countries.

## **Third-Party Materials**

All other third-party trademarks are the property of their respective owners.



# Contents

- About This Guide** **9**
- 1 Introduction to Scripting** **11**
  - The Scripting Language . . . . . 11
  - Scripting Basics. . . . . 12
  - Structuring and Executing Scripts . . . . . 13
  - Types of Scripts . . . . . 13
- 2 Using Symbols and Variables** **15**
  - Symbols Used in Scripts . . . . . 15
    - The Pound Symbol (#) . . . . . 15
    - Quotation Marks (""). . . . . 16
    - The Dollar Sign (\$) . . . . . 16
    - The Question Mark (?) . . . . . 16
    - The Percent Sign (%) . . . . . 16
    - The Exclamation Mark (!) . . . . . 16
    - The Backslash (\) . . . . . 16
    - The @ Symbol . . . . . 17
    - The Hyphen (-) . . . . . 17
  - Understanding Script Variables. . . . . 17
    - Stored Variables . . . . . 17
    - Runtime Variables . . . . . 18
    - Directory Attribute Variables . . . . . 19
    - Passticket Variables . . . . . 20
    - Variables and Values . . . . . 20
- 3 Best Practices in Scripting** **23**
- 4 Working with Scripts** **27**
  - Accessing or Editing Scripts . . . . . 27
  - Using Corporate Scripts . . . . . 28
  - Finding Control IDs . . . . . 28
  - Advanced Windows Scripting. . . . . 30
    - Frame Support . . . . . 31
- 5 SecureLogin Commands** **33**
  - AAVerify . . . . . 34
  - Add . . . . . 36
  - Attribute. . . . . 37
  - BeginSplashScreen / EndSplashScreen . . . . . 37
  - Break . . . . . 38
  - Call . . . . . 39
  - ChangePassword. . . . . 40
  - Class . . . . . 41
  - ClearPlat . . . . . 42
  - Click . . . . . 44

ConvertTime . . . . .	46
Ctrl . . . . .	46
Delay . . . . .	47
Dialog / EndDialog . . . . .	48
DisplayVariables . . . . .	49
Divide . . . . .	50
DumpPage . . . . .	51
EndScript . . . . .	52
Event . . . . .	52
GetCheckBoxState . . . . .	53
GetCommandline . . . . .	54
GetEnv . . . . .	54
GetIni . . . . .	55
GetReg . . . . .	55
GetSessionName . . . . .	56
GetText . . . . .	56
GetURL . . . . .	57
GotoURL . . . . .	58
If / Else / EndIf . . . . .	58
Include . . . . .	62
Increment / Decrement . . . . .	63
KillApp . . . . .	64
Local . . . . .	64
MessageBox . . . . .	65
Multiply . . . . .	67
OnException/ClearException . . . . .	68
Parent / EndParent . . . . .	70
PickListAdd . . . . .	72
PickListDisplay . . . . .	72
PositionCharacter . . . . .	73
ReadText . . . . .	74
RegSplit . . . . .	76
ReloadPlat . . . . .	76
Repeat / EndRepeat . . . . .	77
RestrictVariable . . . . .	79
Run . . . . .	80
SelectListBoxItem . . . . .	81
SendKey . . . . .	82
Set . . . . .	82
SetCheckBox . . . . .	84
SetCursor . . . . .	85
SetFocus . . . . .	86
SetPlat . . . . .	86
SetPrompt . . . . .	89
Strcat . . . . .	90
StrLength . . . . .	91
StrLower . . . . .	92
StrUpper . . . . .	93
Sub / EndSub . . . . .	93
Submit . . . . .	94
Subtract . . . . .	95
Tag/EndTag . . . . .	96
Title . . . . .	97
Type . . . . .	98
Sending Keyboard Commands by Using Type . . . . .	100

WaitForFocus . . . . .	101
WaitForText . . . . .	102
<b>6 Practicing Your Scripting Skills</b>	<b>105</b>
Using the Wizard to Create a Script . . . . .	105
Viewing the Wizard's Script. . . . .	107
Experimenting with a Script. . . . .	107
Creating a Password Policy. . . . .	107
Creating Your Own Script. . . . .	108
Experiment: Using Window Finder . . . . .	111
Adding a MessageBox . . . . .	113
Changing Passwords . . . . .	113
<b>7 Keystrokes and Functions</b>	<b>115</b>
Sending Special Keystrokes . . . . .	115
@ Commands Used with Emulators . . . . .	116
<b>8 Troubleshooting Scripts</b>	<b>119</b>
Logging In to Web Sites . . . . .	119
Deriving Application Names from Strings. . . . .	120
Using the -NoEdit Switch . . . . .	121
Using Microsoft JVM and Internet Explorer. . . . .	121
<b>A Quick-Reference Chart</b>	<b>123</b>
<b>B FAQs on Scripting</b>	<b>127</b>
One Script, Two Sets of Credentials . . . . .	127
Cache All Passwords, Prompt for Each Application . . . . .	127
<b>C Trapping SNMP Alerts</b>	<b>129</b>
Producing an Alert . . . . .	129
Example Script . . . . .	129
<b>D Keyboard Functions and Codes</b>	<b>131</b>
<b>E Event Specifiers</b>	<b>137</b>
<b>F Error Codes</b>	<b>141</b>
Error Codes with Tips. . . . .	141





# About This Guide

The *SecureLogin Scripting Guide* is for network administrators. The following sections provide information on scripting:

- ◆ Chapter 1, “Introduction to Scripting,” on page 11
- ◆ Chapter 3, “Best Practices in Scripting,” on page 23
- ◆ Chapter 2, “Using Symbols and Variables,” on page 15
- ◆ Chapter 4, “Working with Scripts,” on page 27
- ◆ Chapter 5, “SecureLogin Commands,” on page 33
- ◆ Chapter 6, “Practicing Your Scripting Skills,” on page 105
- ◆ Chapter 7, “Keystrokes and Functions,” on page 115
- ◆ Chapter 8, “Troubleshooting Scripts,” on page 119
- ◆ Appendix A, “Quick-Reference Chart,” on page 123
- ◆ Appendix B, “FAQs on Scripting,” on page 127
- ◆ Appendix C, “Trapping SNMP Alerts,” on page 129
- ◆ Appendix D, “Keyboard Functions and Codes,” on page 131
- ◆ Appendix E, “Event Specifiers,” on page 137

## Documentation Conventions

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

A trademark symbol (®, ™, etc.) denotes a Novell trademark. An asterisk (\*) denotes a third-party trademark.

When a single pathname can be written with a backslash for some platforms or a forward slash for other platforms, the pathname is presented with a backslash. Users of platforms that require a forward slash, such as Linux or UNIX, should use forward slashes as required by your software.

## User Comments

We want to hear your comments and suggestions about this manual and the other documentation included with this product. Please use the User Comment feature at the bottom of each page of the online documentation, or go to [www.novell.com/documentation/feedback.html](http://www.novell.com/documentation/feedback.html) and enter your comments there.

## Documentation Updates

For the most recent version of this and other SecureLogin guides, see Novell Nsure SecureLogin 3.51.2 on the [Novell documentation Web site \(http://www.novell.com/documentation\)](http://www.novell.com/documentation).

## Additional Documentation

This *Guide* is part of a documentation set for SecureLogin 3.51.2. Other documents include the following:

- ◆ The Help systems in SecureLogin on the desktop as well as SecureLogin snap-ins to ConsoleOne<sup>®</sup> or Microsoft\* Management Console.
- ◆ The [Nsure SecureLogin 3.51.2 Installation Guide](#) (installing SecureLogin, migrating secrets from earlier versions, and configuring Secure Workstation)
- ◆ The [Nsure SecureLogin 3.51.2 Administration Guide](#) (tools and tasks to manage SecureLogin and configure terminal emulators)
- ◆ The [Nsure SecureLogin 3.51.2 Terminal Services Guide](#) (configuring Citrix servers)
- ◆ The [Nsure SecureLogin 3.51.2 Configuration Guide for Terminal Emulation](#) (how to configure Terminal Launcher for selected terminal emulators)
- ◆ The [Nsure SecureLogin 3.51.2 User Guide](#) (using SecureLogin to enable applications for single sign-on)

# 1

## Introduction to Scripting

This section provides information on the following:

- ◆ “The Scripting Language” on page 11
- ◆ “Scripting Basics” on page 12
- ◆ “Structuring and Executing Scripts” on page 13
- ◆ “Types of Scripts” on page 13

### The Scripting Language

The SecureLogin scripting language is a key feature of single sign-on functionality. The scripting language enables SecureLogin to be compatible with almost all network environments and applications, including those that are developed in-house, without the need to modify any application code.

SecureLogin uses the scripting language 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 a script to enter the username, password, and background authentication information.

The script language is used in individual application scripts to retrieve and enter the correct login details. These scripts are stored and secured in a directory (for example, Novell® eDirectory™) to ensure maximum security, support for single-point administration, and manageability.

The script language is used to automate many login processes, such as multi-page logins and login panels requiring other information (such as a surname or telephone number) stored in the directory. The script language also contains the commands required to automate password changes on behalf of users and request user input when it is required.

The scripting language has the following advantages:

- ◆ Enables you to define single sign-on methods for almost any Windows\*, mainframe, Internet, intranet, terminal server, or UNIX\* application.
- ◆ Provides single sign-on functionality without installing back-end modules on your application servers.
- ◆ Provides the flexibility for you and your application owners to choose what to do after an application-generated message is detected.

This feature gives 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.

- ◆ Stores SecureLogin data (for example, user credentials and application scripts) in the directory and protects the data.
- ◆ Can use Novell SecretStore<sup>®</sup> technology to provide additional benefits:
  - ◆ Provides an additional level of security.
  - ◆ Enables you to share secrets with other applications (for example, Novell iChain<sup>®</sup> and Novell Portal Services).
  - ◆ Enables you to use NICI between the workstation and the server.

SecretStore requires Novell eDirectory.

- ◆ On startup, locates objects in the directory and caches their encrypted contents in memory (and optionally on disk) for later use by the workstation's SecureLogin single sign-on agent.

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

- ◆ Full control of which applications are single-sign-on enabled.
- ◆ The ability to update the entire directory database with a new application login script by updating a single object.

The corporate scripts are stored in a Container object rather than 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.

## Scripting Basics

A script is essentially a list of instructions that SecureLogin follows to perform various tasks upon various windows. For example, for Windows applications (\*.exe files) a script is written for each executable file that you want SecureLogin to act upon. In that script, you are able to assign different instructions to each screen that an executable file or application might produce. Therefore, you have the choice of acting upon only the login panel, selected windows, or every window (for example, account locked, invalid username, invalid password, expired password) that the executable file produces.

SecureLogin follows scripts from left to right, top to bottom. However, with the use of Flow Control commands (for example, Call) you can skip, repeat or jump to certain parts of the script.

With the use of Dialog Specifier commands, you can assign individual sections of a script to the different windows that an executable file might produce. Such assignments allow the login dialog box, for example, to be treated differently from the “wrong password” dialog box.

The scripting language can read from and write to variables. These variables enable SecureLogin to use corporate scripts while still keeping each individual user's secrets securely stored in the directory. The scripting language can also read attributes (for example, the user's full name or phone number) from the username's attributes in the directory. For more information on variables, see [“Understanding Script Variables” on page 17](#).

SecureLogin is able to write information to the screen as well as read from it with the use of commands such as ReadText. You can use this functionality to extract usernames, domains in use, and error messages. You can then use Variable Manipulator commands to perform calculations, break apart information, and join the information back together.

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

## Structuring and Executing Scripts

A script is a simple piece of text that is stored by the SecureLogin script broker. Scripts store the login name, password, and any other information in fields required for authentication. Scripts are stored in the local database and in the directory.

Each script has a name, called the application name, which uniquely identifies it within a particular single sign-on database. In addition, each script has a type, known as the application type (prebuilt, Windows, Web, or Java\*). The application type specifies the type of application the script refers to and which of the SecureLogin components executes it.

SecureLogin scripts execute sequentially from the first line. There are no flow control mechanisms as such. However, in some instances a component might choose not to execute certain statements, as in the Dialog / EndDialog or If/Else/EndIf statements.

Each line in the script consists of one or more arguments. Arguments are separated by white space (spaces and tabs), unless they are enclosed in quotation marks. For example, the following line contains three arguments:

```
A simple "command to get started"
```

The arguments are as follows:

- ◆ A
- ◆ simple
- ◆ "command to get started"

After a script has been broken into arguments, the quotation marks are removed. If you need to specify an actual quotation mark in a script, precede it with a backslash (for example, \").

The first argument on a line is the command. It specifies the action that the line takes. The rest of the arguments on the line, if any, are passed to that command. Different commands take varying numbers of arguments. For a list of commands and their arguments, see [Chapter 5, “SecureLogin Commands,” on page 33](#).

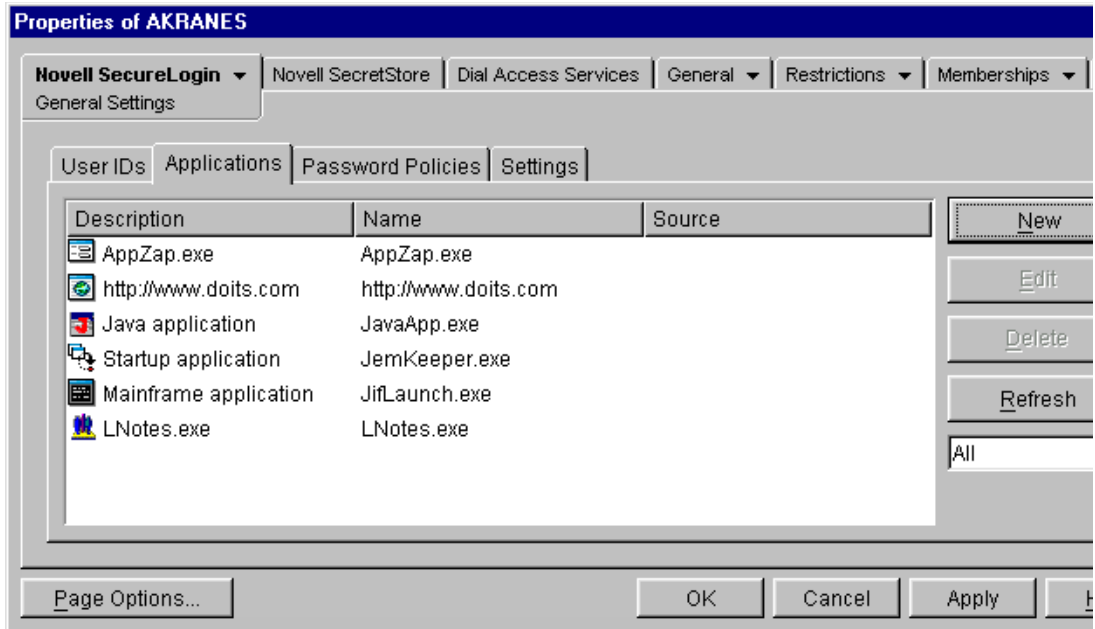
A line that begins with a # character is treated as a comment and is ignored in the script language. The following example illustrates the use of the # character:

```
Window "login"  
Delay 30  
#SecureLogin ignores this line and the next two lines  
#while executing the script.  
#The Delay command is used to wait for the window to be created correctly.  
Type "$Username"
```









Scripts are interpreted as SecureLogin components to perform the sign-in process. This functionality ensures that any variables that are substituted are current.

## Types of Scripts

Using the Applications tab, you can view a list of applications that are enabled for single sign-on. The Description column displays information about the application, including icons that represent the type of script stored for that application.



The following table provides information on the icons.

Script Type	Description
Windows application 	For Windows-based applications. Represented by a generic window icon.
Web application 	For Web-based applications. Maintains backward compatibility with older scripts. Represented by a ringed planet. For new Web scripts, use the Advanced Web script type.
Advanced Web 	For Web-based applications. Enables SecureLogin to use legacy script commands along with commands that were introduced with SecureLogin 3.5.
Java application 	For Java-based applications. Represented by a red J.
SecureLogin Startup 	For applications that are executed during the startup of SecureLogin. Represented by a circular arrow.
Terminal Launcher 	For applications that require access via an emulator. Represented by a black-with-white monitor.
Lotus Notes 	For scripts that are used to log in to Lotus Notes.
Corporate script 	For applications applied at a Container level. Represented by a down-arrow in the lower left corner.

When you add an application that has a prebuilt script, SecureLogin automatically enters a description for that application. When you add an application that doesn't have a prebuilt script, the name that you enter to describe the application appears in the Description column.

# 2

## Using Symbols and Variables

This section contains information on the following:

- ◆ “Symbols Used in Scripts” on page 15
- ◆ “Understanding Script Variables” on page 17

### Symbols Used in Scripts

The SecureLogin scripting language uses the following symbols to define the function of lines in the script:

- ◆ “The Pound Symbol (#)” on page 15
- ◆ “Quotation Marks (””)” on page 16
- ◆ “The Percent Sign (%)” on page 16
- ◆ “The Exclamation Mark (!)” on page 16
- ◆ “The Backslash (\)” on page 16
- ◆ “The @ Symbol” on page 17
- ◆ “The Hyphen (-)” on page 17

### The Pound Symbol (#)

Use the pound or hash symbol (#) to define a line of text as a comment field, so that you can leave notes within a script. The script engine ignores any line that starts with a # symbol.

You can use comment lines to do the following:

- ◆ Define sections of a script, such as login window or change password window.
- ◆ Explain complex sections of a script.
- ◆ Remove command lines from a script while the script is being written and edited.

Removing lines by commenting them saves having to continuously delete and rewrite lines while testing.

- ◆ Make notes, such as when the script was written and what version of the software the script was written for.

When used within a command (for example, Class or Type), the pound or hash symbol takes on a different meaning, specifying a numerical value. This numerical value can be used to specify a target for the command. The command listings provide additional details. See [Chapter 5, “SecureLogin Commands,” on page 33](#).

## Quotation Marks (""")

Use quotation marks (""") to group text or variables that contains spaces. Use these symbols with commands such as Type, MessageBox, and If -Text. Without quotation marks, command lines such as the following won't work as expected:

```
Type Database 2
MessageBox Confirm your login details.
If-Text Login failure
```

For these command lines to work, quotation marks must be used to group the text:

```
Type "Database 2"
MessageBox "Confirm your login details."
If-Text "Login failure"
```

## The Dollar Sign (\$)

Use the dollar sign (\$) to define a SecureLogin variable that is persistent. Use these variables to store information such as usernames and passwords. For more information on the \$ variable, see [“Stored Variables” on page 17](#).

## The Question Mark (?)

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. However, with the use of the Local command, these variables are reset each time the script is started. Use these variables to store temporary information, such as counting, data processing, and date information.

The question mark is also used with several internal system-generated variables. For more information on the ? variable, see [“Runtime Variables” on page 18](#).

## The Percent Sign (%)

Use the percent 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 %CN and %Surname.

For more information on the types of variables, see [“Understanding Script Variables” on page 17](#).

## The Exclamation Mark (!)

Use the exclamation mark (!) to define the use of a passticket. A passticket is a one-time password that is generated using a combination of an encryption key, encryption offset, and the current time. Such passwords are only valid for a short time (from 30 seconds up to 2 minutes). The encryption key and offset can be defined manually or automatically generated for the program.

For more information, see [“Passticket Variables” on page 20](#).

## The Backslash (\)

Use the backslash symbol (\) with the Type and SendKey commands to specify the use of a special function. The symbol is used in conjunction with values to simulate keystrokes. For example, use \N to simulate pressing the Enter key in a Windows application.



For details on the values that can be used with the backslash symbol, see the command listings in [Chapter 5, “SecureLogin Commands,” on page 33](#).

## The @ Symbol

The @ symbol is similar to the backslash symbol. However, the @ symbol is limited to HLLAPI-enabled emulators. Use it in conjunction with values to simulate keystrokes. For example, use @E to simulate pressing the Enter key in a terminal emulator application.

For more information on the @ symbol, see [“@ Commands Used with Emulators” on page 116](#) and the command listings.

## The Hyphen (-)

Use the hyphen (-) as a switch within several commands (for example, If and Type). Use it in conjunction with values to modify the behavior of commands (such as -Raw), or to switch certain functions (such as -YesNo) on or off.

For details on the values that you can use with the hyphen, see the command listings in [Chapter 5, “SecureLogin Commands,” on page 33](#).

## Understanding Script Variables

This section contains information on the following:

- ◆ [“Stored Variables” on page 17](#)
- ◆ [“Runtime Variables” on page 18](#)
- ◆ [“Directory Attribute Variables” on page 19](#)
- ◆ [“Passticket Variables” on page 20](#)
- ◆ [“Variables and Values” on page 20](#)

Generally, don't use spaces when you specify variables. For example, specify \$Username\_Alias instead of \$Username Alias. If you use spaces, enclose the entire variable in quotation marks (for example, “\$Username Alias”).

Each variable defaults to the platform specified in the script name. By using a variable, you can change this by using a variable. For example, you might have the following script, named www.website1.com:

```
Type $username  
Type $password password
```

You might want to use these variables in a script named www.website2.com:

```
Type $username (www.website1.com)  
Type $password (www.website1.com) password
```

## Stored Variables

Stored variables are the most common style of variable used in SecureLogin scripts. They are preceded with a dollar sign (\$). Use these variables to store the values used during the login process, such as usernames, passwords, and any other required details.

The values of these variables are stored in the directory under the User object. The values are encrypted so that only the user can access them.

Variables can be stored separately for each application's script, so that the username variable is different for each application. However, you can set an application to read variables from another application's script. This is useful for applications that share user accounts or passwords. For details on how to do this, see the description in [“SetPlat” on page 86](#).

If a stored variable is referenced in a script, and no value has been stored for that variable (for example, the first time the program is run), SecureLogin prompts the user to enter a value for the variable. This is an automatic process. It is also possible to manually trigger this process to prompt a user to enter new values for particular variables. For details on how to do this, see the description of [“DisplayVariables” on page 49](#) and [“ChangePassword” on page 40](#).

### Example: Stored Variables in Use

```
Dialog
  Class #32770
  Title "Login"
End Dialog

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

To hide a variable from an administrator by displaying it as \*\*\*\* instead of clear text, begin the variable name with \$Password. For example, \$Password PIN is protected, but \$PIN isn't.

## Runtime Variables

SecureLogin is able to read details from the system and use the details to create variables that can be incorporated into the scripting language. These variables are automatically generated as runtime variables. They can be used in the same manner within any application definition.

In general, use runtime variables for date information, to store calculations, or to process data. You can also use runtime variables for temporary passwords and usernames.

Runtime variables are preceded with the question mark symbol (?). They have two modes: Normal and Local. Normal runtime variables are reset each time SecureLogin is started. Local runtime variables are reset each time the script is started. Runtime variables are Normal by default. For details on how to switch a runtime variable from General to Local mode, see the description in [“Local” on page 64](#).

Runtime variables aren't stored in the directory or the SecureLogin local cache. They are used straight from the computer's memory. For this reason, don't use runtime variables to store usernames, passwords, or other details that SecureLogin will need to access in the future. If runtime variables are used for such details, the user will be prompted to enter them each time the script is run or each time SecureLogin is restarted.

Users aren't prompted for \$variables that have no value. These variables are given the value NOTSET.

The following table lists the runtime variables that SecureLogin supports.

Variable Name	Description
?SysVersion(system)	The local SecureLogin Windows agent version. This variable can be used to determine whether specific support is built into the product running on the user's workstation. The format of the variable is major.minor.subminor.build. For example, 03050109 represents v3.5.1.9, in WW.XX.YY.ZZ format.
?BrowserType(system)	Contains either Internet Explorer or Netscape and indicates which browser the script is running in. This variable is set only in a Web script.
?SysUser(system)	The name of the user currently using SecureLogin.
?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 where the current SecureLogin user's directory object exists.
?SysTree(system)	The name of the directory tree that SecureLogin is currently using.
?SysServer(system)	The name of the server that was entered in the Novell login dialog box. This variable is only available if the Novell client login extension is installed.
?CurrTime(system)	The running time in seconds from January 1970 to the present. Use this variable to force password changes every x days. Don't use scripting to force a password change if you want to continue having the application generate the change password event (recommended). Use this variable on applications where a password expiration can't be set at the application's back end.

### Example: A Runtime Variable in Use

```

Dialog
  Class #32770
  Title "ERROR"
EndDialog

Local ?ErrorCount
Increment ?ErrorCount
If ?ErrorCount eq "2"
  MessageBox "This is the second time you have received this error. Would
you like to reset the application?" -YesNo ?Result
  If ?Result eq "Yes"
    KillApp "App.exe"
    Run "C:\App\App.exe"
  Else
    Set ?ErrorCount "0"
  EndIf
EndIf

```

## Directory Attribute Variables

SecureLogin is able to read directory (for example, eDirectory) attributes from the currently logged-in user's object. For example,

Type %cn

reads the CN attribute from the currently logged in user's object, then types the attribute.

Use % variables only when SecureLogin is configured to use a directory and only on single-valued text attributes.

## Passticket Variables

Passticket variables are used to generate one-time passwords. 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. The SecureLogin script parser uses these numbers to generate the one-time password.

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

```
!Name of application definition
```

or

```
!default
```

For example, to use a passticket variable for the Outlook application, you create two stored variables called \$DESKEY and \$DESOFFSET under the Outlook application definition. You then set values for the two stored variables. You can then use the variable *!Outlook* whenever you need to generate a one-time password.

You can also use !Default, which automatically reads the values from the current application definition.

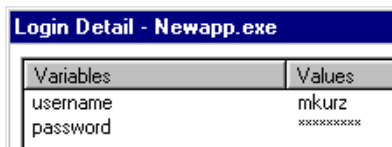
If the credentials used to generate one-time passwords (OTPs) don't already exist in a secured area of the SecureLogin cache (that is, the \$DESKEY and \$DESOFFSET variables aren't defined), they are retrieved from the closest SecureLogin Advanced Authentication Server.

If the \$DESKEY and \$DESOFFSET variables are not given values, SecureLogin generates random values the first time a password is generated and stores the values for later use.

## Variables and Values

SecureLogin stores your username and password in the form of a variable and its value. Your username and password are not included in the script. Instead, a variable is used in the script. The value of the variable is your username or password.

Logins consist of a set of variables. You can use any name for a variable. A variable can contain any text. As the following figure illustrates, the Variable column usually just contains the password and username for a particular application. However, in some more complicated applications, there might be other variables.



Variables	Values
username	mkurz
password	xxxxxxxx

This example has two variables: username and password. The script for this application has the following line:

```
Type $Username
```

The variable \$Username is written in the script. The value of \$Username in this example is mkurz. When the script runs, SecureLogin looks for the variable \$Username in the user's login details. There it finds and reads the value mkurz. SecureLogin enters the value mkurz into the login dialog box.

At runtime, the value of the variable \$Username (mkurz) is read. However, in the script you only see the variable \$Username.



# 3

## Best Practices in Scripting

Use the following rules when writing a SecureLogin script. Although these rules are not compulsory, they accomplish the following:

- ♦ Make reading the script easier.
- ♦ Help you modify the scripts if you need to make changes later.

Example scripts in this guide follow these rules.

### Using Capital Characters

Use capital characters where applicable.

Use This	Instead of This
<code>MessageBox "Some text" -YesNo ?Result</code>	<code>Messagebox "Some text" -yesno ?result</code>

### Indenting

Indent sections of scripts between pairs of commands, such as Dialog/EndDialog, Repeat/EndRepeat, and If/Else. An indent of three spaces is optimal.

Use This	Instead of This
<code>If -Text "Some text"     #Do this Else     #Do this EndIf</code>	<code>If -Text "Some text" #Do this Else #Do This EndIf</code>

### Leaving Blank Lines

Leave a blank line between sections of the script, such as the Dialog block and the rest of the script.

Use This	Instead of This
<code># Login Dialog Box Dialog     Class #32770     Title "Login" EndDialog  Type \$Username #1001 Type \$Password #1002 Click #1</code>	<code># Login Dialog Box Dialog     Class #32770     Title "Login" EndDialog Type \$Username #1001 Type \$Password #1002 Click #1</code>

## Placing and Naming Subroutine Sections

Place subroutine sections of the script at the bottom of the script, not halfway through. The name of the subroutine should describe its function. It shouldn't simply be a numeric name. The name should follow the rules for capitalizing.

## Using Quotation Marks for Text in Commands

Even if quotation marks aren't required, always use them around segments of text in commands.

Use This	Instead of This
Type "Text"	Type Text
Or	Or
If -Text "Login"	If -Text Login

## Capitalizing Variables

Begin variable names with a capital letter.

Use This	Instead of This
Type \$Username	Type \$username

## Placing Switches

Place switches directly after the command (for example, Type -Raw, If -Text).

Use This	Instead of This
Type -Raw \$Username	Type \$Username -Raw

## Password Policy Names

Use program names to represent password policy names for the program they are used for. Don't use numerical names.

Use This	Instead of This
GroupwisePasswordPolicy	PasswordPolicy3

## Hiding Variables

If you want to hide a variable from an administrator by displaying the variable as \*\*\*\* instead of clear text, begin the variable name with \$Password. For example, \$PasswordPIN will be protected, but \$PIN won't be protected.

## Using Comments

Use comments throughout the script to explain what each section does and how it does it. At the top of the script, enter and comment out information such as who wrote the script and the date that the script was last modified.



**NOTE:** To help explain example scripts in the SecureLogin Commands section, this *Guide* places explanations to the left of the scripts. For example, see Example: Windows Script in ["AAVerify" on page 34](#).

---

<b>Use This</b>	<b>Instead of This</b>
<pre>#Written by M. Kurz June 7, 2002 #Modified by C. Bertrand July 3, 2004  #Login Dialog Box Dialog     Class #32770     Title "Login" EndDialog</pre>	<pre>Dialog     Class #32770     Title "Login" EndDialog</pre>

---

### **Using the Include Command**

Wherever possible, use the Include command to create generic scripts for commonly used elements, such as password change procedures. For common processes within the script, use subroutines.



# 4

## Working with Scripts

To help you customize the login capabilities of your users, this section provides information on the following:

- ♦ “Accessing or Editing Scripts” on page 27
- ♦ “Using Corporate Scripts” on page 28
- ♦ “Finding Control IDs” on page 28
- ♦ “Advanced Windows Scripting” on page 30

Also see [Building Blocks for SecureLogin Scripting \(http://www.novell.com/coolsolutions/slmag/features/a\\_scripting\\_building\\_blocks\\_sl.html\)](http://www.novell.com/coolsolutions/slmag/features/a_scripting_building_blocks_sl.html).

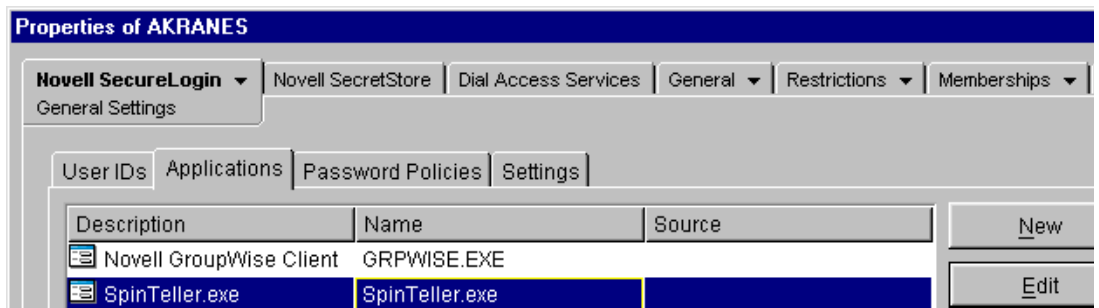
### Accessing or Editing Scripts

Each single-sign-on-enabled application has a script. A basic script tells SecureLogin how to log in to the application. You can create more involved scripts that allow you to perform other password management tasks, such as detecting expired passwords and generating new passwords.

SecureLogin has a scripting wizard as well as a host of prebuilt scripts. These features enable you to easily enable a broad range of applications for single sign-on.

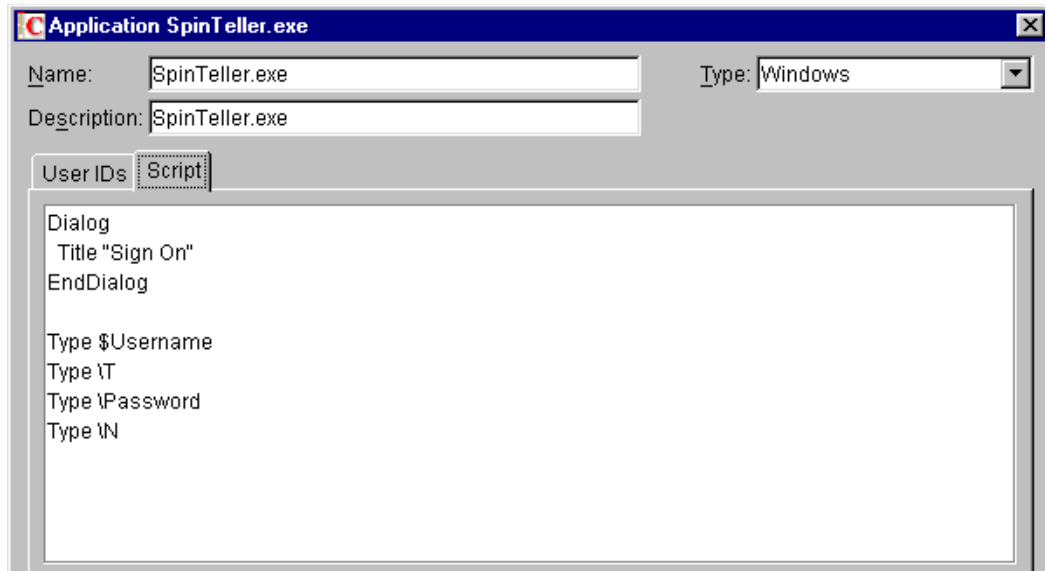
You manage scripts for applications by using ConsoleOne<sup>®</sup>, the Microsoft Management Console (MMC), SecureLogin Manager (slmanager.exe), or the SecureLogin workstation client.

- 1 Right-click an object (for example, an OU or User object), then click Properties.
- 2 Click Novell SecureLogin > General Settings > Applications.



- 3 Click an application, click Edit, then click Script.

The following figure illustrates the Script tab and an example simple script.



#### 4 Make changes.

For commands used in scripts, along with example scripts for those commands, see [Chapter 5, “SecureLogin Commands,”](#) on page 33.

To experiment with a sample script and a test application, see [Chapter 6, “Practicing Your Scripting Skills,”](#) on page 105.

For a scenario to enable authentication to MyRealBox through single sign-on, see [Using Novell SecureLogin to Enable Web Applications for Single Sign-On \(http://developer.novell.com/research/appnotes/2002/may/02/apv.htm#1228584\)](#) in the May 2002 issue of *AppNotes*

## Using Corporate Scripts

Corporate scripts reduce administrative tasks. You can create a container for corporate scripts, place a corporate script there, then point all users to that script. All users in or directed to that corporate script get their settings from that corporate script.

For information on setting up corporate scripts, see [“Managing Corporate Scripts”](#) in *Nsure SecureLogin 3.51.2 Administration Guide*.

## Finding Control IDs

A control ID is a number that uniquely identifies a field, such as a button, within a window. Many script commands related to logging in to Windows applications require a control ID.

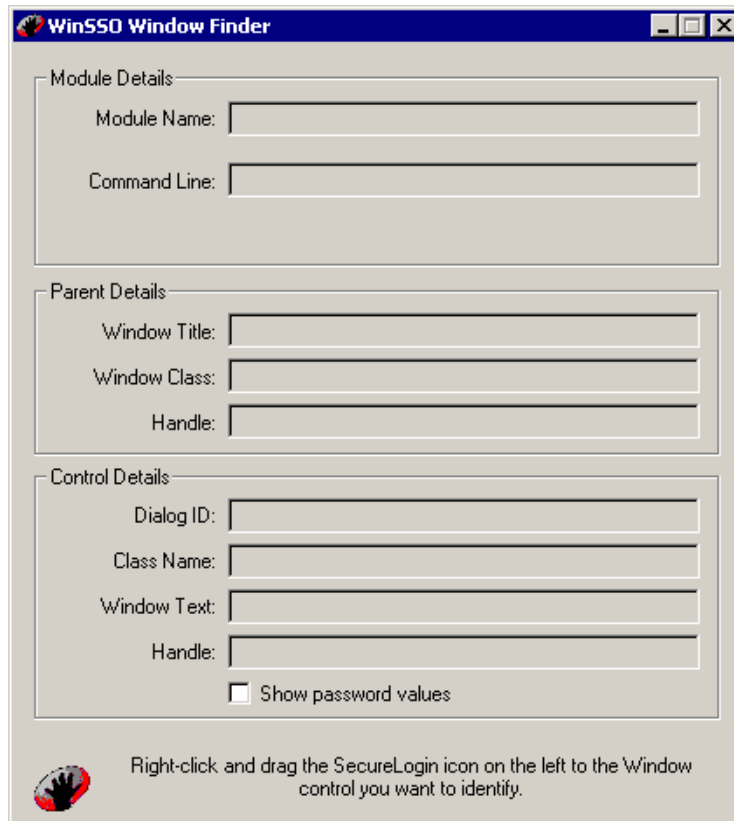
To help you determine these control IDs, SecureLogin includes a tool called Window Finder. This tool displays information about a control that you have selected.

To inspect a control:

- 1 Click Start > Programs > Novell SecureLogin > Window Finder.
- 2 Right-click the SecureLogin icon and drag it over the control of interest.

The Window Finder tool displays the details of the control.

If an application page hides the Window Finder, click the WinSSO Window Finder icon on the system tray.



The following table provides information on fields in the dialog box:

Field	Description
Module Name	The name of the executable that created the window. Use this name for the application name of the Windows single sign-on script.
Command Line	The path to the module or executable.
Window Title	The title of the window that contains the control. You can use this title in a window or title statement.
Window Class	A field for information only. Each window has a class associated with it.
Handle	The handle of the parent window.
Dialog ID (Control ID)	A unique identifier. Each control has a unique identifier, called the control ID. Use this number as the target for Type, Click, Ctrl, and SetPlat statements. For information on each of these commands, see <a href="#">Chapter 5, "SecureLogin Commands," on page 33</a> .
Class Name	A name that determines the type of the control. For single sign-on to work correctly, the SecureLogin Windows component must be able to read and write text to the specified control. The class name determines the type of the control and whether reading and writing is possible. Supported classes include edit, combobox, and static.

Field	Description
Window Text	A field that displays the text contained within the control. This information can be useful in troubleshooting and for writing the regular expression required by the Setplat command.
Handle	The handle of the control window.

## Advanced Windows Scripting

Advanced Windows Scripting (AWS) is an extension to the single sign-on scripting language. AWS enables arbitrary Windows messages to trigger scripts. In earlier versions of SecureLogin, a script written for an application was triggered when (and only when) the application sent a WM-CREATE message. AWS provides Event, which is a single new specifier.

The Event command takes exactly one parameter, which is the Windows event that triggers execution of the controlled block. The following script illustrates this block:

```
## BeginSection: "Global Script Configuration"
## EndSection: "Global Script Configuration"
## BeginSection: "Login Window"
Dialog
  Class "#32770"
  Title "Novell iFolder Login"
  Ctrl #1
  Ctrl #1092
  Event WM_ACTIVATE
EndDialog
ReadText #1092 ?Message
If ?Message eq "Place a shortcut to the iFolder on the desktop"
  If ?Failure eq 1
    Set ?Failure <notset>
    EndScript
  Else
    Setprompt "Username:"
    Type $Username #1007
    Setprompt "Password:"
    Type $Password #1079
    Setprompt "iFolder Server Name:"
    Type $Optional #1001
    Click #1
    Setprompt "Enter your iFolder account information."
  Endif
EndIf
## EndSection: "Login Window"

Dialog
  Parent
    Title "Novell iFolder Login"
  EndParent
  Title "Novell iFolder"
  Ctrl #2
EndDialog
Readtext #65535 ?ErrorMessage
If ?ErrorMessage eq "You must enter a server address."
  Click #2
  Set ?Failure 1
EndIf
```

Advanced Windows Scripting meets two requirements:

- ◆ It handles login dialog boxes that are created some time before they are displayed.

In earlier SecureLogin releases, SecureLogin launched scripts on a Create event. Whenever a window was created, SecureLogin could key off that event. When the login dialog box was created, SecureLogin was able to log in from that event.

However, some applications have a feature where the login dialog box is created long before it is displayed and before a user is able to actually log in. When this login dialog box is created, it sends a WM\_CREATE message, which triggers any associated script.

**Scenario: SecureLogin before AWS.** You log in to Novell iFolder®. The Create event fires a script and logs you in. However, iFolder creates and instantiates a subsequent login. You close iFolder but still have it running on your system tray. You log in to iFolder again. SecureLogin is unable to recognize that event.

Using AWS, you can delay execution of the login script until, for example, the login dialog box is activated (and fires a WM\_ACTIVATE message). SecureLogin 3.51.2 recognizes the second event. SecureLogin can key off Create, Activate, Destroy, mouse clicks, and other events.

- ◆ It adds value to applications that SecureLogin already handles.

For example, a login system allows the user to choose from *n* different servers by using a combo box. With AWS, you can delay execution of the script until the user has selected a server from the combo box. You cause the delay by using the EM\_SETSEL message. The script can then read which server has been selected, then choose an appropriate credential set.

With AWS, SecureLogin can enable additional applications. Also, scripts no longer need to be Startup scripts so that all the applications launch. The applications can start at any time.

To use AWS, edit the application scripts by adding events. For a list of events and other information on AWS, see [“Event” on page 52](#).

## Frame Support

In earlier versions, Web pages were scripted as whole pages, and SecureLogin couldn't distinguish between frames within the Web page. The Advanced Web scripting feature enables you to script for a particular frame within a Web page.

**Scenario: Second Set of Fields.** A Web page has three frames in it. Two of these frames have username and password fields. Previous versions of NSL would probably have entered the username and password into the first set of fields. With SecureLogin 3.51.2, you can populate a particular frame by naming the SecureLogin application the URL of that particular frame.

To find the URL of the frame, right-click the frame and select Properties. The next time that you navigate to the Web page that contains that frame, the credentials are automatically populated into that frame.

You can usually get the address of the frame by right-clicking in the frame and selecting Properties.





# 5

## SecureLogin Commands

This section provides information on commands used in SecureLogin scripts. The commands are listed alphabetically.

Following the command, a table provides information in the following format:

Item	Description
<b>Use with:</b>	Java: Use the command as part of a Java script. Startup scripts: Use the command in startup scripts. Terminal Launcher: Use the command in Terminal Launcher scripts. Web: Use the command in Web site scripts. Windows: Use the command in Windows application scripts.
<b>SecureLogin Version:</b>	All: You can use the command in all versions. Version number: The version that the command was introduced in.
<b>Type:</b>	Action: Use the command to perform an action. For example, the Type command types information into an application. Dialog specifier: Use the command to define dialog boxes. For example, see <a href="#">“Parent / EndParent” on page 70</a> and <a href="#">“Class” on page 41</a> . Flow control: Use the command to direct SecureLogin to a specific location in the script. For example, see <a href="#">“Repeat / EndRepeat” on page 77</a> and <a href="#">“EndScript” on page 52</a> . Variable manipulator: Use the command to modify variables. For example, see <a href="#">“Add” on page 36</a> and <a href="#">“Subtract” on page 95</a> .
<b>Usage:</b>	The command argument / <i>variable</i> . Variables, values, text, and other items that you type are italicized in the tables. Optional items that you type are placed in brackets ([ ]).
<b>Arguments:</b>	Argument / <i>variable</i> : A brief explanation of the argument or variable.
<b>Description:</b>	An explanation of the command and how it is used.
<b>Syntax Examples:</b>	Examples of the various ways the command can be written in a script.
<b>Example:</b>	An example script.
Script type	
Script explanation	

The Eq and Noteq comparators are case insensitive. The Seq and SNotEq comparators have been created for use when case sensitivity is required. The commands themselves are not case sensitive.

You can use seq, Seq, or SEQ. However the string the command is matching with is case sensitive. The string “hello” is not equal to “Hello”.

## AAVerify

Item	Description
<b>Use with:</b>	Startup scripts, Terminal Launcher, Web, Windows
<b>SecureLogin Version:</b>	All (Arguments were added in version 3.0.)
<b>Type:</b>	Action
<b>Usage:</b>	<i>AAVerify -Method NMAS sequence -User User object -Tree Tree name</i> [?Result]
<b>Arguments:</b>	
Method	The Novell® Modular Authentication Services (NMAS™) login method that you want to use. If you don't specify a method or login sequence, AAVerify uses the method that was chosen during initial authentication to the directory.
User	The DN of the user that you want to use for the AAVerify command. If you don't specify a username, AAVerify re-authenticates the currently logged-in user.
Tree	The user's NDS® or eDirectory™ tree name. This argument must be used with the -User argument.
[?Result]	An optional variable (preferably a temporary variable) that receives the result of the AAVerify command. The variable is set to either True for success or False for failure.

Item	Description
<b>Description:</b>	<p data-bbox="616 147 1469 278">Used with SecureLogin Advanced Authentication or NMAS to verify the user, typically before the application Username and Password are retrieved and entered into the login box. AAVerify provides reauthentication to an application, using a strong login method. AAVerify is extremely secure.</p> <p data-bbox="616 288 1469 419">For example, a user can be forced to enter a smart card and PIN before the application will log in via single sign-on, even though the application natively knows nothing about smart cards and PINs. If the verification succeeds, the [?Result] is set to True. Otherwise, it is set to False.</p> <p data-bbox="616 429 1469 499">If NMAS is not installed on the workstation, the script sends an error, or an error is returned via [?Result].</p> <p data-bbox="616 510 1469 641">To enable AAVerify with NMAS, make sure that nmas.dll is in the PATH. Also make sure that the NMAS client and specified login sequence are installed and properly configured. For details, see <a href="http://www.novell.com/documentation/lg/nmas21/index.html">Novell Modular Authentication Services (http://www.novell.com/documentation/lg/nmas21/index.html)</a>.</p> <p data-bbox="616 651 1469 721"><b>NMAS Specific:</b> If AAVerify is called with no arguments, the currently logged-in user is re-authenticated by using the login method used for the current login.</p> <p data-bbox="616 731 1469 802"><b>AA Specific:</b> When AAVerify is called in an AA environment, the -method parameter must be present. The method must be one of the following:</p> <ul data-bbox="616 812 862 1155" style="list-style-type: none"> <li>◆ Any</li> <li>◆ Biometric</li> <li>◆ Smart card</li> <li>◆ Token</li> <li>◆ Password</li> <li>◆ Passphrase</li> <li>◆ Directory password</li> <li>◆ SecureID</li> </ul> <p data-bbox="616 1175 1469 1266">If you specify more than one -method argument, you can re-authenticate with any of the specified methods. For example, the command could be used to request authentication using a fingerprint device or a smart card.</p> <p data-bbox="616 1276 1469 1407"><b>IMPORTANT:</b> When the AAVerify command is added to a script, AAVerify increases the security of the target application only if the script can't be altered. If the script can be modified or overwritten, the AAVerify command can be removed, thereby removing the additional security.</p> <p data-bbox="616 1417 1469 1518">Therefore, restrict access to scripts through directory ACLs and SecureLogin settings. With such restrictions in place, only a small, trusted group of administrators can modify, add, or override scripts.</p>
<b>Syntax Examples:</b>	<pre data-bbox="616 1538 1469 1667">AAVerify AAVerify -Method "Enhanced Password" ?Result AAVerify -Method "Enhanced Password" -User "mkurz" - Tree "Production" ?Result</pre>

Item	Description
<b>Example:</b>	# Login Dialog Box
<b>Windows Script</b>	Dialog
The login dialog box is detected. However,	Title "Login"
before SecureLogin enters the user's	Ctrl #32770
credentials, it prompts	EndDialog
Advanced	AAVerify -Method "Enhanced Password" ?Result
Authentication	If ?Result Eq "True"
credentials (for	Type \$Username #1001
example, a smart card	Type \$Password #1002
and PIN, biometric	Click #1
device, or token).	Else
	Messagebox "Authentication failed. Verify that your smart card is inserted and that your PIN is correct. IT x453"
	EndIf

## Add

Item	Description
<b>Use with:</b>	Startup scripts, Terminal Launcher, Web, Windows
<b>SecureLogin Version:</b>	3.0
<b>Type:</b>	Variable manipulator
<b>Usage:</b>	Add <i>Variable1 Variable2</i> [?Result]
<b>Arguments:</b>	
<i>Variable1</i>	The first argument, the number that the second argument will be added to. If the optional ?result argument is not passed in, this argument also contains the result of the addition equation. If you use <i>Variable1</i> without the ?Result argument, <i>Variable1</i> must be a SecureLogin variable. Otherwise, <i>Variable1</i> can be any numeric value.
<i>Variable2</i>	The second argument, the number added to the first argument in the equation. <i>Variable2</i> can be a SecureLogin variable or a numeric value.
[?Result]	Optional. The sum or result of the equation.
<b>Description:</b>	Adds one whole number to another. (Doesn't add fractions.) The numbers can be hard-coded into the script, or they can be variables. The result can be output to another variable or to one of the original numbers.
<b>Syntax Examples:</b>	Add 1 2 ?Result Add ?LoginAttempts ?LoginFailures Add ?LoginAttempts ?LoginFailures ?Result Add ?LoginAttempts 3 Add ?LoginAttempts 3 ?Result

Item	Description
<b>Example</b>	ReadText #103 ?Number1
<b>Windows Script:</b>	ReadText #104 ?Number2
The values of Control IDs 103 and 104 are read into variables. From there they are added, and the result is typed into Control ID 1. From there, they are added, and the result is typed into control ID 1.	Add ?Number1 ?Number2 ?Result Type ?Result #1

## Attribute

Item	Description
<b>Use with:</b>	Advanced Web Script
<b>SecureLogin Version:</b>	3.5
<b>Type:</b>	Specifier
<b>Usage:</b>	<i>Attribute Attribute Name Attribute Value</i>
<b>Arguments:</b>	
<i>Attribute Name</i>	The name of the HTML attribute to discover.
<i>Attribute Value</i>	The value that the above HTML attribute must contain for the condition to be true.
<b>Description:</b>	The Attribute specifier works with the Tag/EndTag command and specifies which HTML attributes and attribute values must exist for that particular HTML tag.
<b>Example:</b>	Tag "Form" Attribute "Name" "Login" EndTag
SecureLogin finds the form that has an attribute of "name" with a value of "login."	

## BeginSplashScreen / EndSplashScreen

Item	Description
<b>Use with:</b>	Terminal Launcher (Generic and Advanced Generic only)
<b>SecureLogin Version:</b>	3.0.4
<b>Type:</b>	Action
<b>Usage:</b>	BeginSplashScreen EndSplashScreen

Item	Description
<b>Arguments:</b>	None
<b>Description:</b>	Displays a Novell splash screen across the terminal emulator window. This command is used to mask any flashing, etc. that is produced by SecureLogin selecting text from the screen. A Delay command at the start of the script ensures that the emulator window is in place before the splash screen is displayed.
<b>Example:</b>	Delay 2000
<b>Terminal Launcher Script</b>	BeginSplashScreen WaitForText "ogin:" Type \$Username EndSplashScreen Type @E
After launching the emulator, SecureLogin waits two seconds for it to connect. The splash screen displays to cover the flashing. SecureLogin detects a login and enters a username. The splash screen disappears.	

## Break

Item	Description
<b>Use with:</b>	Startup scripts, Terminal Launcher, Web, Windows
<b>SecureLogin Version:</b>	2.5
<b>Type:</b>	Action
<b>Usage:</b>	Break
<b>Arguments:</b>	None
<b>Description:</b>	Used within the Repeat/EndRepeat commands to break out of a repeat loop.
<b>Example 1:</b>	Dialog
<b>Windows Script</b>	Class #32770
SecureLogin reads the screen and searches for "Login". If "Login" is found, the Repeat loop is broken and the script continues. If it isn't found, the script checks again.	Title "Login" EndDialog Repeat ReadText #301 "?Text" If ?Text Eq "Login" Break EndIf Delay 100 EndRepeat

Item	Description
<b>Example 2:</b>	# Initial System Login
<b>Terminal Script</b>	WaitForText "ogin:"
The terminal emulator screen is read and the content is searched for a successful login. (In this case, the application main menu appears.) After the user has logged in, the Repeat loop is broken and the script continues. If the login isn't successful, the script checks again. Terminal emulators use Repeat loops for error handling and Break to break out of the loop as appropriate.	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 or Password stored. If -Text "Login Failed" DisplayVariables "The username or password stored by SecureLogin is invalid. Verify your credentials and try again. IT x453." Type \$Username Type @E Delay 500 WaitForText "assword:" Type \$Password Type @E Delay 500 EndIf  # Account is locked for some reason, possibly inactive. If -Text "Account Locked" MessageBox "Your account has been locked, possibly because of inactivity for 40 days. Contact the administrator at x453." EndIf  # Main Menu, user has logged in successfully. If -Text "Application Selection" Break EndIf  Delay 100 EndRepeat

## Call

Item	Description
<b>Use with:</b>	Startup scripts, Terminal Launcher, Web, Windows

Item	Description
<b>SecureLogin Version:</b>	2.5
<b>Type:</b>	Flow control
<b>Usage:</b>	Call <i>Subroutine</i>
<b>Arguments:</b>	
<i>Subroutine</i>	The name of the subroutine to be called. This name must be identical to the name specified in the Sub command.
<b>Description:</b>	<p>Calls and runs a subroutine.</p> <p>When a subroutine is called, the script begins executing from the first line of the subroutine. When the subroutine completes, the script resumes executing from the command immediately following the call command.</p>
<b>Example:</b>	
<b>Terminal Script</b>	<pre>Repeat   If -Text "Username"     Call "Login"   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. Verify your password and click OK to retry logging in. IT x453." \$Password   Call Login EndSub</pre>
If the word Username is found on the screen, the subroutine "Login" is launched. If "Wrong Password" is found, the subroutine "WrongPassword" is launched. Subroutines are particularly useful when you would otherwise need to repeat the same lines of script.	

## ChangePassword

Item	Description
<b>Use with:</b>	Startup scripts, Terminal Launcher, Web, Windows
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Usage:</b>	ChangePassword <i>Variable</i> [Text] [Random]



Item	Description
<b>Arguments:</b>	
<i>Variable</i>	A normal or runtime variable that the new password is stored in.
[Text]	The text you want displayed in the change password dialog box.
[Random]	Invokes the random password generator.
<b>Description:</b>	<p>Allows a single variable to be changed. Use this command in scenarios where password expiration is an issue. The <i>Variable</i> will be set to the new password.</p> <p>The flag for this command is Random. If Random is set, the new password will be generated automatically in compliance with the variable's password policy.</p> <p>If Random is 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. Also see <a href="#">"RestrictVariable" on page 79</a>.</p>
<b>Syntax Examples:</b>	<pre>ChangePassword \$NewPassword ChangePassword ?NewPassword "Enter a new password" ChangePassword ?NewPassword Random</pre>
<b>Example:</b>	
<b>Windows Script</b>	
The script detects the change password event. The application requires the current username and password, then the new password and confirmation of the new password. The script creates a backup of the old password in case the password change fails (which can be detected via the message that pops up). The script then generates and enters a new password.	<pre># Change Password Dialog Box Dialog     Class #32770     Title "Change Password" EndDialog  Set \$PasswordBackup \$Password Type \$Password #1015 ChangePassword \$Password Random Type \$Password #1005 Type \$Password #1006 Click #1  # Change Password Failed Dialog Box Dialog     Class #32770     Title "Change Password Failed" EndDialog  # Set the password back as the password change failed Set \$Password \$PasswordBackup MessageBox "The change password process failed. Retry the password change at your next login. IT x453."</pre>

## Class

Item	Description
<b>Use with:</b>	Startup scripts, Windows
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Dialog specifier
<b>Usage:</b>	Class <i>Window-Class</i>

Item	Description
<b>Arguments:</b>	
<i>Window-Class</i>	A string specifying the window class that this statement will match.
<b>Description:</b>	<p>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 <i>Window-Class</i> argument.</p> <p>If the window matches the <i>Window-Class</i> argument, the execution of the script continues to the next line. If the window doesn't match the <i>Window-Class</i> argument, execution continues at the next dialog statement.</p> <p>You can determine the class by using Window Finder. See <a href="#">"Finding Control IDs" on page 28</a>.</p> <p>In any Dialog statement, place the Class command before the Title command.</p>
<b>Example:</b>	
<b>Windows Script</b>	<pre># Login Dialog Box Dialog   Class #32770   Title "Login" EndDialog Type \$Username #1001 Type \$Password #1002 Click #1</pre>
The dialog box generated by the application is checked to determine if the Window Class is #32770. If True and its title is "Login", that section of the script executes. If False, the script checks the next Dialog block.	

## ClearPlat

For each dialog block in a script, the chosen User ID is reset and must therefore be reselected, either via a SetPlat command or by having the user reselect it from a list.

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

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

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

Item	Description
<b>Use with:</b>	Startup scripts, Terminal Launcher, Web, Windows
<b>SecureLogin Version:</b>	3.51.2
<b>Type:</b>	Action

Item	Description
<b>Usage:</b>	<p>To use the ClearPlat command, add code to three places:</p> <p><b>Application Startup.</b> When an application first starts up, the previously chosen platform should be cleared via ClearPlat. (In a Windows application, add an extra dialog statement for the main window).</p> <p><b>Change Credentials Canceled.</b> If the user decides not to modify the chosen platform's credentials, call ClearPlat. This gives the user a chance to choose a different platform next time.</p> <p><b>Successful Login.</b> Call ClearPlat to allow the user to relogin with a different User ID at a later stage.</p>
<b>Arguments:</b>	None
<b>Description:</b>	Resets the last-chosen User ID, causing subsequent calls to ReLoadPlat to do nothing.
<b>Example: Windows Script</b>	<pre># ==== BeginSection: Application startup ==== Dialog   Class "#32770"   Title "Password Test Application" EndDialog  ClearPlat # ==== EndSection: Application startup ====  # ==== BeginSection: Login ==== Dialog   Class "#32770"   Title "Login"   Ctrl #1001 EndDialog  ReLoadPlat SetPrompt "Username =====&gt;" Type \$Username #1001 SetPrompt "Password =====&gt;" Type \$Password #1002 SetPrompt "Domain =====&gt;" Type \$Domain #1003 Click #1 # ==== EndSection: Login ====  # ==== BeginSection: Login Successful ==== Dialog   Class "#32770"   Title "Login Successful" EndDialog  ClearPlat  Click #2 # ==== EndSection: Login Successful =====</pre>

Item	Description
	<pre># ==== BeginSection: Login Failure ==== Dialog   Class "#32770"   Title "Login Failure" EndDialog  Click #2 ReLoadPlat OnException ChangePasswordCancelled Call ChangeCancelled   ChangePassword \$password ClearException ChangePasswordCancelled Type -Raw \Alt+F Type -Raw L # ==== EndSection: Login Failure ====  # ==== BeginSection: Change Credentials Canceled ==== Sub ChangeCancelled   ClearPlat   EndScript EndSub # ==== EndSection: Change Credentials Canceled ==</pre>

## Click

Item	Description
<b>Use with:</b>	Java, Web, Windows
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Windows Usage:</b>	
Usage 1	Click <i>#Ctrl-ID</i> [-Raw] [-Right]
Usage 2	Click <i>#Ctrl-ID</i> [-Raw] [-x <i>X Coordinate</i> -y <i>Y Coordinate</i> ]
<b>Web Usage:</b>	Click <i>#Number</i>
<b>Arguments:</b>	
<i>#Ctrl-ID</i>	The ID number of the control to be pressed.
[-Raw]	Eliminates the mouse and sends a direct click.
[-Right]	Sends a right-mouse click. Use this argument with the -Raw flag only.
<i>X_Coordinate</i>	Represents the horizontal coordinate relative to the client area of the application (not the screen).
<i>Y_Coordinate</i>	Represents the vertical coordinate relative to the client area of the application (not the screen).

Item	Description
<i>#Number</i>	<p>The pound or hash symbol followed by the sequential number or Control ID of the button to be pressed.</p> <p><b>Web-Specific:</b> The number of the button is determined by the Web page layout. See <a href="#">"DumpPage" on page 51</a>.</p> <p><b>Windows-Specific:</b> The control ID. Use the Window Finder to discover the number.</p> <p><b>Java-Specific:</b> The index to use is put in an example script that the Java wizard creates.</p>
<b>Description:</b>	<p>When used with Windows applications, the Click command sends a click instruction to the specified <i>#Ctrl-ID</i>. If the button to be clicked doesn't have a control ID, the Type \N command will often click the default button in a Windows application.</p> <p>The -Raw flag can be set if the button or control doesn't respond to the Click command. The -Raw flag causes SecureLogin to emulate the mouse and send a direct click message to the control. Using the -Right flag with the -Raw flag sends a right-click to the control.</p> <p>Setting the <i>#Ctrl-ID</i> to 0 (zero) sends the Click command to the window that the script is running on.</p> <p>If -Raw is specified, you can also set the X and Y coordinates. These coordinates are relative to the client area of the application, not the screen.</p> <p>When used with Web pages, the Click command takes a single argument, which is the sequential number on the page of the button to be pressed. "Click #3" clicks the third button on the page. Keep in mind that, because of Web page layout and design, the sequential order of the buttons might not be obvious. In this case, you might need to use the DumpPage command to discover the field layout.</p>
<b>Syntax Examples:</b>	<pre>Click #1 Click #1 -Raw -Right Click -X 12 -Y 24</pre>
<b>Example 1:</b> <b>Windows Script</b> The Login dialog is detected, the username and password are entered, and button number 1 (in this case the Login button) is clicked.	<pre># Login Dialog Box Dialog   Class #32770   Title "Login" EndDialog Type \$Username #1001 Type \$Password #1002 Click #1</pre>
<b>Example 2:</b> <b>Web Script</b> The username and password are entered, then the Login button is clicked.	<pre>Type \$Username Type \$Password Password Click #1</pre>

Item	Description
<b>Example 3:</b>	#Login Dialog Box
<b>Windows Script</b>	Dialog
In this example, the application is Java.	Class #32770
Therefore, there is no Control ID. Instead, the click command is told to click a particular place on the window.	Title "Login"
	End Dialog
	Type \$Username
	Type \$Password
	Click -X 12 -Y 24

## ConvertTime

Item	Description
<b>Use with:</b>	Startup scripts, Terminal Launcher, Web, Windows
<b>SecureLogin Version:</b>	3.0.4
<b>Type:</b>	Variable manipulator
<b>Usage:</b>	ConvertTime <i>time String Time</i>
<b>Arguments:</b>	
<i>String Time</i>	The output variable.
<b>Description:</b>	Converts a numeric time value, for example, ?CurrTime(system) into a legible format and stores it in <i>String Time</i> .
<b>Example:</b>	# Login Dialog Box
<b>Windows Script</b>	Dialog
Converts the time to a readable format and displays it in a dialog box.	Class #32770
	Title "Login"
	End Dialog
	ConvertTime ?CurrTime ?Time(system) ?Time
	MessageBox ?Time

## Ctrl

Item	Description
<b>Use with:</b>	Java, startup scripts, Windows
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Dialog specifier
<b>Usage:</b>	Ctrl #Ctrl-ID [ <i>Regular Expression</i> ]

Item	Description
<b>Arguments:</b>	
<code>#Ctrl-ID</code>	The ID number of the control to be checked.
<code>[RegEx]</code>	The regular expression.
<b>Description:</b>	<p>Determines if a window contains the control expressed in the <code>#Ctrl-ID</code> argument. The control ID number is a constant that is established at the time a program is compiled.</p> <p><b>NOTE:</b> Third-party software control ID numbers might not be consistent from one version to the next.</p> <p>You can use the Window Finder tool to determine the control ID number. See <a href="#">"Finding Control IDs" on page 28</a>.</p> <p>Using the <code>[RegEx]</code> argument adds a further check that allows the script to skip to the next command. If the text on the specified <code>#Ctrl-ID</code> doesn't conform to the <code>[RegEx]</code>, the script skips to the next dialog statement as though the <code>#Ctrl-ID</code> didn't exist.</p>
<b>Syntax Examples:</b>	<pre>Ctrl #1 Ctrl #1 "OK"</pre>
<b>Example:</b>	<pre># Login Dialog Box Dialog   Title "Login"   Ctrl #1 "OK"   Ctrl #2 "Cancel"   Ctrl #3 "Help" EndDialog</pre>
<b>Windows Script</b>	
The dialog box is tested to see if it contains the correct Control IDs with the correct values. If any of the Control IDs are missing, or the text doesn't match, the script passes on to the next dialog block.	<pre>Type \$Username Type \T Click #1</pre>

## Delay

Item	Description
<b>Use with:</b>	All
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Usage:</b>	<i>Delay Time Period</i>
<b>Arguments:</b>	
<i>Time Period</i>	A period of time, expressed in milliseconds (1/1000 of a second), to pause execution of the script.

Item	Description
<b>Description:</b>	<p>Delays the action of the script for the time specified in the <i>Time Period</i> argument. The time specified in the <i>Time Period</i> argument is noted in milliseconds. For example, Delay 5000 creates a 5-second pause.</p> <p>Use the Delay command to accommodate an introduction screen or some other custom feature.</p> <p>To optimize SecureLogin's performance, use the Delay command in all Repeat loops.</p>
<p><b>Example:</b>  <b>Windows Script</b>  The login box is detected. However, the script waits half a second before acting upon it to ensure that the box is complete.</p>	<pre># Login Dialog Box Dialog     Class #32770     Title "Login" EndDialog  Delay 500 Type \$Username #1001 Type \$Password #1002 Click #1</pre>

## Dialog / EndDialog

Item	Description
<b>Use with:</b>	Java, Windows
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Dialog specifier
<b>Usage:</b>	Dialog EndDialog
<b>Arguments:</b>	None
<b>Description:</b>	<p>Identifies the beginning and end of a dialog specification block. Use these commands to construct a dialog specification block, which consists of a series of dialog specification statements (for example, Ctrl, Title).</p> <p>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 execution proceeds to the next dialog block, if any. You need to specify enough information in the dialog block to make the dialog box unique (for example, Log In or Change Password).</p> <p>The part of the script that follows the EndDialog command is called the script body. Another dialog block, or the end of the script, terminates the script body.</p>



Item	Description
<b>Example:</b>	# Login Dialog Box
<b>Windows Script</b>	Dialog
The dialog box is	Title "Login"
tested to determine its	Ctrl #1 "OK"
identity. If it is	Parent
determined to be the	Title "Application 1"
login box, the script	EndParent
parses the Type and	EndDialog
Click commands to	Type \$Username #1001
complete the login	Type \$Password #1002
process.	Click #1

## DisplayVariables

Item	Description
<b>Use with:</b>	All
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Usage:</b>	DisplayVariables [ <i>User Prompt</i> ] [ <i>Variable [Variable]...</i> ]
<b>Arguments:</b>	
[ <i>User Prompt</i> ]	Optional, customized text to be displayed in the Enter SecureLogin Variables dialog box.
[ <i>Variables</i> ]	The name of the variables you want the user to be prompted for. If you don't specify the name of the variables, SecureLogin prompts for all variables that the script uses.

Item	Description
<b>Description:</b>	<p>Displays a dialog box that lists the user's stored variables (for example, \$Username and \$Password) for the current application. The user can edit the variables from this dialog box.</p> <p>For example, if the login is unsuccessful because of an incorrect username or password, the DisplayVariables command prompts the user to edit the stored username or password values. From that point, the login process proceeds as usual.</p> <p>You can specify a particular variable to display. If the <i>variables</i> parameter is specified, DisplayVariables prompts only for the variables specified.</p> <p>To replace the default prompt text in the Enter SecureLogin Variables dialog box, enter the replacement text in quotation marks after the DisplayVariables command. Limit the text to 90 characters.</p> <p>If no variables are stored for the user the first time SecureLogin attempts to apply single sign-on to the application, the prompt will not be customized.</p> <p>After variables are stored for the user, the prompt is customized when the script is run.</p> <p>You can also customize the text in the prompt by using the SetPrompt command. See <a href="#">"SetPrompt" on page 89</a>.</p> <p><b>TIP:</b> You can use the OnException EnterVariablesCancelled command to prevent a user from canceling the DisplayVariables prompt.</p>
<b>Syntax Examples:</b>	<pre>DisplayVariables DisplayVariables "Enter your details" DisplayVariables "Enter a new password" \$Password DisplayVariables "Enter your username and password" \$Username \$Password DisplayVariables "" \$Username \$Password</pre>
<b>Example:</b>	
<b>Windows script</b>	
The Wrong Password dialog box is detected. SecureLogin prompts the user to enter a new username and password for it to use. After these have been specified, SecureLogin enters them into the dialog box and clicks OK.	<pre># 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</pre>

## Divide

Item	Description
<b>Use with:</b>	Startup scripts, Terminal Launcher, Web, Windows
<b>SecureLogin Version:</b>	3.0
<b>Type:</b>	Variable manipulator

Item	Description
<b>Usage:</b>	Divide <i>Variable1 Variable2</i> [? <i>Result</i> ]
<b>Arguments:</b>	
<i>Variable1</i>	The dividend. The first argument. The number that will be divided by the second argument. This argument will contain the result if the optional [? <i>Result</i> ] argument is not passed in. If you use the <i>Variable1</i> argument without the [? <i>Result</i> ] argument, <i>Variable1</i> must be a SecureLogin variable (either ? <i>Variable1</i> or \$ <i>Variable1</i> ). Otherwise, <i>Variable1</i> can be any numeric value.
<i>Variable2</i>	The divisor. The second argument. The number that the first argument is divided by. The <i>Variable2</i> argument can be a SecureLogin variable or a numeric value.
[? <i>Result</i> ]	The quotient or result of the equation.
<b>Description:</b>	Divides one whole number by another. (Doesn't divide fractions or give results in fractions.) The numbers can be hard-coded into the script, or they can be variables. The result can either be output to another variable or to one of the original numbers.  <b>NOTE:</b> This is an integer arithmetic that is 5/2, not 2.5.
<b>Syntax Examples</b>	Divide "1" "2" ? <i>Result</i> Divide ?LoginAttempts ?LoginFailures Divide ?LoginAttempts ?LoginFailures ? <i>Result</i> Divide ?LoginAttempts "3" Divide ?LoginAttempts "3" ? <i>Result</i>
<b>Example:</b> <b>Windows Script</b> The values of Control IDs 103 and 104 are read into variables. From there they are divided and typed into Control ID 1.	<pre>ReadText #103 ?Number1 ReadText #104 ?Number2 Divide ?Number1 ?Number2 ?Result Type ?Result #1</pre>

## DumpPage

Item	Description
<b>Use with:</b>	Advanced Web Script
<b>SecureLogin Version:</b>	3.5
<b>Type:</b>	Action
<b>Usage:</b>	DumpPage <i>Variable</i>
<b>Arguments:</b>	
<i>Variable</i>	The string variable to receive the page information.
<b>Description:</b>	Provides information about the current Web page. This information can be useful for debugging scripts for a Web page.

Item	Description
<b>Example:</b>	DumpPage ?dump MessageBox ?dump

## EndScript

Item	Description
<b>Use with:</b>	Startup scripts, Terminal Launcher, Web, Windows
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Usage:</b>	EndScript
<b>Arguments:</b>	None
<b>Description:</b>	Immediately terminates execution of the script.
<b>Example:</b>	Dialog
<b>Windows Script</b>	Title "Login Failure"
The login box is	Ctrl #1
detected. SecureLogin	EndDialog
enters the username	ReadText #65535 ?ErrorMsg
and password and	If "Incorrect Password" -In ?ErrorMsg
clicks OK. If the	MessageBox "You have entered an incorrect password"
"Incorrect Password"	EndScript
message is detected,	EndIf
SecureLogin tells the	
user that the password	
was incorrect and	
terminates the script.	

## Event

Item	Description
<b>Use with:</b>	Windows
<b>SecureLogin Version:</b>	3.5
<b>Type:</b>	Dialog specifier
<b>Usage:</b>	Event <i>Event</i>
<b>Arguments:</b>	
<i>Event</i>	The application event to monitor. This corresponds to a Windows event, which usually begins with WM_. For a list of events that you can specify, see <a href="#">Appendix E, "Event Specifiers," on page 137</a> . For information on Windows events, see <a href="http://msdn.microsoft.com">http://msdn.microsoft.com</a> .

Item	Description
<b>Description:</b>	<p>Scripts generally execute when an application window is created. This timing corresponds to the WM_CREATE message that is received from an application window at startup.</p> <p>By adding the Event specifier to a dialog block, you can override this behavior, so that a script now executes when (and only when) the specified message is generated. If no Event specifier is given, it is equivalent to "Event WM_CREATE".</p> <p>You can apply the Event specifier only within a Dialog and EndDialog statement block. Also, specify only one Event per Dialog block. If there is a requirement to monitor for multiple events, each must be specified within its own Dialog block. For further information, refer to MSDN or other documentation on the Win32 messaging system.</p> <p>Microsoft's Spy++ or similar Windows message spy tools are useful for trapping event names in specific windows.</p>
<b>Syntax Examples:</b>	<pre>Dialog   Class "someclass"   Event WM_ACTIVATE EndDialog MessageBox "Caught the WM_ACTIVATE message"</pre>

## GetCheckBoxState

Item	Description
<b>Use with:</b>	Advanced Web Script
<b>SecureLogin Version:</b>	3.5
<b>Type:</b>	Action
<b>Usage:</b>	GetCheckBoxState <i>Item Number Variable</i>
<b>Arguments:</b>	
<i>Item Number</i>	The Windows control ID of the check box.
<i>Variable</i>	The target variable for the status of the specified check box. The value returned will be Checked or Unchecked. The variable can be either a ? or a \$ variable.
<b>Description:</b>	Returns the current state of the specified check box.
<b>Example.:</b>	<pre>GetCheckBoxState #25 ?state1 GetCheckBoxState #26 ?state2 MessageBox ?state1 MessageBox ?state2</pre>

## GetCommandline

Item	Description
<b>Use with:</b>	Startup scripts, Windows
<b>SecureLogin Version:</b>	3.0.4
<b>Type:</b>	Action
<b>Usage:</b>	GetCommandline <i>Variable</i>
<b>Arguments:</b>	
<i>Variable</i>	Defines where the captured command line will be stored.
<b>Description:</b>	<p>Captures the full command line of the program that is loaded and saves it to the specified variable.</p> <p><b>TIP:</b> You can use GetCommandLine to detect and differentiate back-end systems or databases for use with multiple logins in the SAP application.</p> <p>This command isn't supported under Windows 95/98.</p>
<b>Example:</b>	<pre>GetCommandline ?Text</pre>
<b>Windows script</b>	<pre>If ?Text Eq "C:\Winnt\notepad.exe"     Killapp Notepad.exe EndIf</pre>
	<p>The command line of the application is read, then tested to see if it is Notepad.exe. If it is, Notepad is closed. If it isn't, the script ends.</p>

## GetEnv

Item	Description
<b>Use with:</b>	All
<b>SecureLogin Version:</b>	3.5
<b>Type:</b>	Action
<b>Usage:</b>	GetEnv <i>envvar variable</i>
<b>Arguments:</b>	
<i>Envvar</i>	The environment variable name that you want to retrieve.
<i>Variable</i>	Defines where the retrieved environment variable data will be stored.
<b>Description:</b>	Reads the value of an environment variable and saves it in the specified <i>variable</i> .

Item	Description
<b>Example: Windows Script</b>	<pre>GetEnv "SESSIONNAME" ?SessionName If ?SessionName eq "console"     MessageBox "Running from Citrix Server Console" EndIf</pre>

## GetIni

Item	Description
<b>Use with:</b>	Java, Terminal Launcher, Web, Windows
<b>SecureLogin Version:</b>	3.5
<b>Type:</b>	Action
<b>Usage:</b>	GetIni <i>ini file section key variable</i>
<b>Arguments:</b>	
<i>Ini file</i>	The filename that you want to read the section or key from.
<i>section</i>	The name of the section that contains the key name.
<i>key</i>	The name of the key that is to be read.
<i>Variable</i>	Defines where the retrieved environment variable data is stored.
<b>Description:</b>	Allows reading data from the .ini file.
<b>Example: Windows Script</b>	<pre>GetIni "c:\program files\lotus\notes\notes.ini" "Notes". "KeyFileName ?NotesDefaultIDFile SetPlat ?NotesDefaultIDFile</pre>

## GetReg

Item	Description
<b>Use with:</b>	All
<b>SecureLogin Version:</b>	3.5
<b>Type:</b>	Action
<b>Usage:</b>	GetReg <i>registry variable</i>
<b>Arguments:</b>	
<i>Registry</i>	The registry entry that you want to read.
<i>Variable</i>	Defines where the retrieved environment variable data will be stored.

Item	Description
<b>Description:</b>	<p>Reads data from the registry and saves it in the specified variable. The registry entry input needs to be in the following format: HIVE\KEY\Value</p> <p>Valid hives are:            "HKCR" HKEY_CLASSES_ROOT            "HKCC" HKEY_CURRENT_CONFIG            "HKCU" HKEY_CURRENT_USER            "HKLM" HKEY_LOCAL_MACHINE            "HKU" HKEY_USERS</p>
<b>Example:</b>	<pre>GetReg "HKLM\Software\ABCCorp\ProductID" ?ProductID</pre>
<b>Windows Script</b>	<pre>If ?ProductID noteq "xxxxxxxxxx"     #Not corporate desktop EndScript EndIf</pre>

## GetSessionName

Item	Description
<b>Use with:</b>	Terminal Emulator
<b>SecureLogin Version:</b>	3.5
<b>Type:</b>	Action
<b>Usage:</b>	GetSessionName <i>?variable</i>
<b>Arguments:</b>	
<i>Variable</i>	The target variable that the session name is copied into.
<b>Description:</b>	Finds the current HLLAPI session name that is being used to connect and returns it to the specified variable.
<b>Example:</b>	GetSessionName ?Session_name
<b>Windows Script</b>	

## GetText

Item	Description
<b>Use with:</b>	Terminal Launcher, Web
<b>SecureLogin Version:</b>	3.0
<b>Type:</b>	Action
<b>Usage:</b>	GetText <i>Variable</i>



Item	Description
<b>Arguments:</b>	
<i>Variable</i>	Defines where the captured text will be stored.
<b>Description:</b>	<p>Gets all of the text from the screen and saves it to the specified variable. GetText is used in a large Web script that might contain several If -Text statements.</p> <p>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, if you use GetText (for example, If ?Text -In ?FromGetText), the script can contain multiple If-Text commands, with only one scan of the screen.</p>
<b>Example:</b>	<pre>GetText ?Text</pre>
<b>Web Script</b>	<pre>If "Login" -In ?Text   Type \$Username   Type \$Password Password EndIf</pre>
The text content of the Web page is copied into the ?Text variable. SecureLogin tests for the presence of "Login." If it exists, SecureLogin enters the credentials and submits them automatically.	

## GetURL

Item	Description
<b>Use with:</b>	Web
<b>SecureLogin Version:</b>	3.0
<b>Type:</b>	Action
<b>Usage:</b>	GetURL <i>Variable</i>
<b>Arguments:</b>	
<i>Variable</i>	Defines where the captured URL will be stored.
<b>Description:</b>	Captures the URL of the site that is loaded and saves it to the specified variable.
<b>Example:</b>	<pre>GetURL ?URL</pre>
<b>Web Script</b>	<pre>If "Logout" -In ?URL   MessageBox "You have chosen to log out of the applications.   You will now be redirected to the Intranet home page."   GoToURL "http://Intranet" EndIf</pre>
The URL of the Web site is copied into the ?URL variable and tested 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.	

# GotoURL

---

Item	Description
<b>Use with:</b>	Web
<b>SecureLogin Version:</b>	2.5
<b>Type:</b>	Action
<b>Usage:</b>	GotoURL <i>URL</i> [- <i>frame</i> ]
<b>Arguments:</b>	
<i>URL</i>	The URL that the browser will navigate to.
<i>-frame</i>	Opens the URL in the frame that started the script.
<b>Description:</b>	Makes the browser navigate to the specified <i>URL</i> . By default, the command opens the new Web page in the main window, rather than the frame that started the script. When you use the <i>-frame</i> option on a framed Web page, the URL redirect occurs only in the current frame rather than in the parent window.  You must specify <i>http://</i> before the URL.
<b>Example:</b> <b>Web Script</b> SecureLogin detects an incorrect password message, displays a message box informing the user, then browses to the Novell Web site.	<pre>If -text "Incorrect Password"     MessageBox "You have entered an incorrect password"     GotoURL "http://www.novell.com" EndIf</pre>

---

## If / Else / EndIf

---

Item	Description
<b>Use with:</b>	Startup scripts, Terminal Launcher, Web, Windows
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Flow control
<b>Usage 1:</b>	If <i>Value1 Gt Lt Value2</i> #Do This [Else] #Do This Endif
<b>Usage 2:</b>	If <i>Value1 Eq NEq Value2 [-I]-S</i> #Do This [Else] #Do This Endif

---

Item	Description
<b>Usage 3:</b>	If <i>Value1</i> [-In]-NotIn <i>Value2</i> [-I]-S] #Do This Else #Do This EndIf
<b>Usage 4:</b>	If -text [-frame] <i>Text</i> #Do This [Else] #Do This EndIf
<b>Usage 5:</b>	If -exists <i>Variable</i> #Do This [Else] #Do This EndIf
<b>Arguments:</b>	
<i>Value1</i>	The left hand side of the expression to be evaluated.
<i>Value2</i>	The right hand side of the expression to be evaluated.
<i>Text</i>	The text that will be searched for.

Item	Description
<b>Description:</b>	<p>Establishes a block to be executed if the expression supplied is found to be true.</p> <p>The Else command works inside an If block. This command is executed if the operator in the If block is false.</p> <p>The EndIf command terminates the If block.</p> <p>The If command supports the following text comparison operators:</p> <ul style="list-style-type: none"> <li>◆ Eq/SEq Evaluates to true if the left-hand side is equal to the right-hand side.</li> <li>◆ NEq/SNEq Evaluates to true if the left-hand side is not equal to the right-hand side.</li> <li>◆ -In/-SIn Evaluates to true if the left-hand side is a substring of the right-hand side.</li> <li>◆ -NotIn/-SNotIn Evaluates to true if the left-hand side is not a substring of the right-hand side.</li> </ul> <p>When using these text comparison operators, you can optionally specify whether the comparison is to take into account the case of the strings being compared. If -I is specified, the comparison will be case insensitive. If -S is specified, the comparison will be case sensitive. By default the Eq and NEq operators are not case sensitive, while the -In and -NotIn operators are case sensitive.</p> <p>The If command supports two numerical comparison operators:</p> <ul style="list-style-type: none"> <li>◆ Gt/Lt Evaluates to true if the left-hand side is greater than /less than the right-hand side. Because this is a numerical comparison, the right-hand side and left-hand side must be numbers.</li> </ul> <p>The operator -exists checks for a stored variable. If the specified variable exists, the comparison evaluates to true.</p> <p>The operator -Text is supplied to directly query the application being scripted for a particular string. -Text evaluates to true if the specified text is found in the application windows of the application being scripted. For IE scripts, an optional argument -Frame can be supplied. -Frame restricts the command to looking for the specified text in the current frame.</p>
<b>Syntax Examples:</b>	<pre>If ?Value1 Gt ?Value2 If -Text "Login" If -Exists \$RunBefore If "Login" -In ?Text</pre>

Item	Description
<p><b>Example 1:</b>  <b>Web Script</b>  SecureLogin tests for "Incorrect Password". If it is found, an incorrect password message box is displayed. If the error message isn't found, SecureLogin logs in as normal.</p>	<pre>If -Text "Incorrect Password"     DisplayVariables "You have entered the incorrect password. Verify it and try logging in again." Else     Type \$Username     Type \$Password Password EndIf</pre>
<p><b>Example 2:</b>  <b>Windows Script</b>  Each time the script is run, a variable is incremented. This is used to count the number of times the dialog box has been displayed. If it is displayed more than three times, the application is closed. If the login is successful, the count is reset.</p>	<pre># Login Dialog Box Dialog     Class #32770     Title "Login" EndDialog  ReadText #1001 ?Username  If -Exists \$Username Else     Set \$Username ?Username EndIf  Increment ?RunCount If ?RunCount Gt "3"     MessageBox "Login has been attempted too many times. The application will be closed."     KillApp "app.exe" Else     Type \$Username #1001     Type \$Password #1002     Click #1 EndIf  # Login Successful Dialog Box Dialog     Title "Login Successful"     Ctrl #1 EndDialog  Set ?RunCount "0"</pre>
<p><b>Example 3:</b>  <b>Web Script</b>  The text content of the Web page is copied to ?WebText. The variable is then tested to see if "Login" is present. If it is, SecureLogin performs the login process. If it isn't, the script is terminated.</p>	<pre>GetText ?WebText If "Login" -In ?WebText     Type \$Username     Type \$Password Password Else     EndScript EndIf</pre>

Item	Description
<b>Example 4:</b> <b>Startup Script</b> When SecureLogin loads, it tests to see whether the user has run SecureLogin before. If the user hasn't, SecureLogin sets the variable so that the message is displayed only once. SecureLogin then displays a welcome message along with the option for further details on SecureLogin.	<pre> If -Exists \$LoadedBefore     EndScript Else     MsgBox "Welcome to SecureLogin, 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?" -YesNo ?Result     Set \$LoadedBefore "Yes"     If ?Result Eq "Yes"         GoToURL "http://www.company.com/SecureLoginDetails.htm"     EndIf EndIf </pre>

## Include

Item	Description
<b>Use with:</b>	All
<b>SecureLogin Version:</b>	3.0
<b>Type:</b>	Flow control
<b>Usage:</b>	Include <i>Platform-Name</i>
<b>Arguments:</b>	
<i>Platform-Name</i>	The name of the script to be included.
<b>Description:</b>	Allows commonly-used application script code to be shared by multiple applications. The script identified by <i>Platform-Name</i> is included at execution time into the calling application script. The script included with the Include command must comprise commands supported by the calling application.
<b>Example:</b> <b>Windows Script</b> SecureLogin detects the login dialog, executes the notepad.exe script, then enters the user's credentials.	<pre> # Login Dialog Box Dialog     Class #32770     Title "Login" EndDialog  Include Notepad.exe Type \$Username #1001 Type \$Password #1002 Click #1 </pre>

# Increment / Decrement

Item	Description
<b>Use with:</b>	All
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Variable manipulator
<b>Usage:</b>	Increment <i>Variable</i> Decrement <i>Variable</i>
<b>Arguments:</b>	
<i>Variable</i>	The name of the variable to increase or decrease in value.
<b>Description:</b>	<p>Adds or subtracts from a specified variable. You can use Increment and Decrement to count the number of passes a particular script has made. After the number of instances is equal to the specified number, you can instruct the script to run another task or end the script.</p> <p>This instruction can be particularly useful in the following situations:</p> <ul style="list-style-type: none"> <li>• When you configure an application whose login panel is similar to other windows within the application.</li> <li>• To easily control the number of attempts a user can have to access an application.</li> </ul>
<b>Syntax Examples:</b>	<pre>Increment ?RunCount Decrement ?RunCount</pre>
<b>Example: Windows Script</b>	<pre># Login Dialog Box Dialog   Class #32770   Title "Login" EndDialog  Increment ?RunCount If ?RunCount Gt "3"   MessageBox "Login has been attempted too many times. The application will be closed."   KillApp "app.exe" Else   Type \$Username #1001   Type \$Password #1002   Click #1 EndIf  # Login Successful Message Dialog   Title "Login Successful"   Ctrl #1 EndDialog  Set ?RunCount "0"</pre>
Each time the script is run, a variable is incremented. This is used to count the number of times the dialog box has been displayed. If it is displayed more than three times, the application is closed. If the login is successful, the count is reset.	

# KillApp

Item	Description
<b>Use with:</b>	All
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Usage:</b>	KillApp <i>Process-Name</i>
<b>Arguments:</b>	
<i>Process-Name</i>	The name of the process that will be terminated.
<b>Description:</b>	Terminates an application.
<b>Example:</b>	<pre># Login Dialog Box Dialog   Class #32770   Title "Login" EndDialog  Increment ?RunCount If ?RunCount Gt "3"   MessageBox "Login has been attempted too many times. The application will be closed."   KillApp "app.exe" Else   Type \$Username #1001   Type \$Password #1002   Click #1 EndIf  # Login Successful Message Dialog   Title "Login Successful"   Ctrl #1 EndDialog  Set ?RunCount "0"</pre>
<b>Windows Script</b>	
Each time the script is run, a variable is incremented. This is used to count the number of times the dialog box has been displayed. If it is displayed more than three times, the application is closed. If the login is successful, the count is reset.	

# Local

Item	Description
<b>Use with:</b>	All
<b>SecureLogin Version:</b>	3.0
<b>Type:</b>	Variable manipulator
<b>Usage:</b>	Local <i>?Variable</i>



Item	Description
<b>Arguments:</b>	
<i>?Variable</i>	The runtime variable that will be declared as local.
<b>Description:</b>	<p>Declares that a runtime variable will only exist for the lifetime of the script. Use local runtime variables the same way as normal runtime variables, and still write local runtime variables as <i>?Variable</i>.</p> <p>Declare local runtime variables to be local by using the Local command, followed by the variable name. When runtime variables are declared local, they cannot be set back again. You can declare a runtime variable to be local at any time in a script.</p> <p>Using local runtime variables slightly increases the performance of SecureLogin. Use local runtime variables to run scripts multiple times and not have the runtime variables stored between each run of the script.</p> <p>Also use local runtime variables to prevent runtime variables from overwriting each other. Overwriting could happen if two instances of a script 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 script but attached to different emulator sessions.</p>
<b>Example:</b>	
<b>Windows Script</b>	
A variable is declared local, then used to count the number of times a dialog box has been displayed. If the box has been displayed too many times, SecureLogin alerts the user, then closes the application.	<pre># Invalid Login Message Dialog     Class #32770     Title "Login Failure" EndDialog  Local ?RunCount Increment ?RunCount If ?RunCount Gt "5"     MessageBox "Closing Application"     KillApp "PasswordText.exe" EndIf Type \$Username Type \$Password</pre>

## MessageBox

Item	Description
<b>Use with:</b>	Startup scripts, Terminal Launcher, Web, Windows
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Usage:</b>	<pre>MessageBox <i>Data</i> [-Background] [-DefaultNo] [-YesNo <i>Variable</i>] [-YesNoCancel <i>?Variable</i>]</pre>

Item	Description
<b>Arguments:</b>	
<i>Data</i>	The text to be displayed to the user. can be several strings, variables or a combination of the two.
[-Background]	When specified, allows the user to open an application and work in that application, without having to respond to the MessageBox.  If this parameter isn't used, the MessageBox remains the topmost window and the user must respond to the MessageBox before continuing with any other work.
[-DefaultNo]	An optional parameter, used only with the -YesNo and -YesNoCancel flags. When the -DefaultNo parameter is set, default focus goes to the No button instead of to the Yes button.
[-YesNo]	Allows the user to select either Yes or No within the message box rather than being limited to an OK button only.
? <i>Variable</i>	Required with the -YesNo or -YesNoCancel flag to store the result of the user action.
[-YesNoCancel]	Allows the user to select either Yes, No, or Cancel when a message box is presented.
<b>Description:</b>	Displays a dialog box that contains the text specified in the <i>Data</i> variable. The script is suspended until the user reacts to this message. As the following line illustrates, MessageBox can take any number of text arguments, including variables:  <pre>MessageBox "The User "\$Username" has just been logged into the system"</pre> 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.  You can capture the result of the MessageBox immediately after the flag by using a runtime ? <i>Variable</i> . The variable value is set to Yes, No, or Cancel.
<b>Syntax Examples:</b>	<pre>MessageBox "Script completed successfully" MessageBox "Do you want to continue?" -YesNo ?Result MessageBox "Do you want to continue?" -YesNoCancel ?Result -Background -DefaultNo</pre>

Item	Description
<p><b>Example 1:</b>  <b>Windows Script</b>  SecureLogin detects the change password dialog box, asks the user whether the user wants to change the password, and informs the user that the change was successful.</p>	<pre># Change Password Dialog Box Dialog   Class #32770   Title "Change Password" EndDialog  MessageBox "Your password has expired. Would you like to change it now?" -YesNo ?Result  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</pre>
<p><b>Example 2:</b>  <b>Terminal Launcher Script</b>  Message boxes can be useful when troubleshooting scripts. You can display a box before each step in the script, to enable you to see where the script fails to execute.</p> <p><b>NOTE:</b> The WaitForText command cuts off the first character because it will find both Password and password and respond to all password entry points.</p>	<pre>MessageBox "Beginning wait for Login prompt" WaitForText "ogin:" MessageBox "Login detected. Now entering Username." Type \$Username MessageBox "Username entered. Now simulating Enter." Type @E MessageBox "Enter has been simulated. Now waiting for Password" WaitForText "assword:" MessageBox "Password detected. Now entering Password." Type \$Password MessageBox "Password entered. Now simulating Enter." Type @E MessageBox "Sequence completed. The user should now be logged in"</pre>

## Multiply

Item	Description
<b>Use with:</b>	All
<b>SecureLogin Version:</b>	3.0
<b>Type:</b>	Variable manipulator
<b>Usage:</b>	Multiply <i>Variable1 Variable2</i> [? <i>Result</i> ]

Item	Description
<b>Arguments:</b>	
<i>Variable1</i>	The multiplicand. The first argument. The whole number (but not a fraction) that will be multiplied by the second argument. This argument will contain the result if the optional [?Result] argument is not passed in. If you use the <i>Variable1</i> argument without the [?Result] argument, <i>Variable1</i> must be a SecureLogin variable (either ?Variable1 or \$Variable1). Otherwise, <i>Variable1</i> can be any numeric value.
<i>Variable2</i>	The multiplier. The second argument. The number that the first number will be multiplied by. <i>Variable2</i> can be a SecureLogin variable or a numeric value.
[?Result]	Optional. The product or result of the equation.
<b>Description:</b>	Multiplies one whole number by another. (Doesn't multiply fractions.) The numbers can be hard-coded into the script, or they can be variables. The result can be output to another variable or to one of the original numbers.
<b>Syntax Examples:</b>	<pre>Multiply "1" "2" ?Result Multiply ?LoginAttempts ?LoginFailures Multiply ?LoginAttempts ?LoginFailures ?Result Multiply ?LoginAttempts "3" Multiply ?LoginAttempts "3" ?Result</pre>
<b>Example:</b>	<pre>ReadText #103 ?Number1 ReadText #104 ?Number2 Multiply ?Number1 ?Number2 ?Result Type ?Result #1</pre>
<b>Windows Script</b>	<p>The values of control IDs 103 and 104 are read into variables. From there they are multiplied, then typed into control ID 1.</p>

## OnException/ClearException

Item	Description
<b>Use with:</b>	All
<b>SecureLogin Version:</b>	3.0.4
<b>Type:</b>	Flow control
<b>Usage:</b>	<pre>OnException <i>Exception_Name</i> Call <i>SubRoutine</i> ClearException <i>Exception_Name</i></pre>

Item	Description
<b>Arguments:</b>	
<i>Exception Name</i>	<p>The name of the exception that you want to act on. Two exceptions are supported:</p> <ul style="list-style-type: none"> <li>♦ <code>ChangePasswordCancelled</code> When a user clicks Cancel in the Change Password dialog box.</li> <li>♦ <code>EnterVariablesCancelled</code> When a user clicks Cancel in the automatic variable prompt dialog box.</li> </ul>
<i>Subroutine</i>	The name of the subroutine you want to run when the exception condition is found to be true.
<b>Description:</b>	<p>Detects when certain conditions are met. This is when the Cancel button is clicked in either of two dialog boxes. When the condition is met, a subroutine is run.</p> <p>Use the <code>ClearException</code> command to reset the exceptions value.</p>
<b>Syntax Examples:</b>	<pre>OnException ChangePasswordCancelled Call DisplayError ClearException ChangePasswordCancelled</pre>
<b>Example 1:</b>	# Login Failed Dialog Box
<b>Windows Script</b>	Dialog
The login has failed	Class #32770
because the user has	Title "Login Failed"
invalid credentials	EndDialog
stored. Provide the	OnException EnterVariablesCancelled Call VariablesCancelled
user with an	DisplayVariables "Verify your Username and Password and try
opportunity to verify the	again. Helpdesk x5555."
username and	ClearException EnterVariablesCancelled
password. If the user	Type \$Username #1001
clicks Cancel, the	Type \$Password #1002
exception is executed.	Click #1
The user must then	Sub VariablesCancelled
enter credentials.	OnException EnterVariablesCancelled Call
	VariablesCancelled
	DisplayVariables "You can't cancel this verification
	dialog box. Verify your username and password when prompted.
	Then click OK to retry logging in."
	ClearException EnterVariablesCancelled
	EndSub

Item	Description
<b>Example 2:</b>	# Change Password Dialog Box
<b>Windows Script</b>	Dialog
The user has been prompted to change the password.	Class #32770
SecureLogin must handle password changes so that the password is updated both in the application and in the user's 3DES encrypted store (in the directory against the User object).	Title "Change Password"
	EndDialog
	Type \$Username #1005
	Type \$Password #1006
	OnException ChangePasswordCancelled Call ForceChangePwd
	ChangePassword \$Password "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.
	You can't cancel. IT x5555"
	Type \$Password #1007
	Type \$Password #1008
	ClearException ChangePasswordCancelled
	EndSub

## Parent / EndParent

Item	Description
<b>Use with:</b>	Windows
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Dialog specifier
<b>Usage:</b>	Parent EndParent
<b>Arguments:</b>	None

Item	Description
<b>Description:</b>	<p>The Parent command begins 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, the script ends.</p> <p>For example, the command Title in a Parent block returns False if the title of the Parent doesn't match the one specified in the command.</p> <p>However, if a command in a Parent block returns a False result, the execution doesn't skip to the next Parent block, as it would in a dialog block. Instead, the Parent block proceeds to the next dialog box, or the script terminates if no further dialog block exists.</p> <p>The EndParent command terminates a Parent block and sets the subject of the script back to the original window. You can nest the Parent command, allowing the parent block to act on the parent of the parent.</p> <p>The Parent command is particularly useful in applications where the dialog box (for example, Login Dialog Box) is the child of an open window, typically in the background. If you are unable to single sign-on to an application after enabling it with the Wizard, you typically need to specify Parent blocks.</p> <p>Also, you can use the Parent command to execute commands on a dialog's parent. For example, you can get a script to click a button on the parent window, as illustrated in Example 2.</p> <p><b>TIP:</b> If you use the wizard or try to enable an application and it doesn't seem to be working, try using the Parent command. It is able to handle windows that are within windows.</p>
<p><b>Example 1:</b>  <b>Windows Script</b>  The Parent command is used to further specify the dialog box that is used for logging in. In this case, the parent of the login box has a Class of "Centura:MDIFrame".</p>	<pre># Login Dialog Box Dialog   Class "Centura:Dialog"   Title "Login"   Ctrl #4098   Ctrl #4100   Parent     Class "Centura:MDIFrame"   EndParent EndDialog  Type \$Username #4098 Type \$Password #4100 Click #4101</pre>
<p><b>Example 2:</b>  <b>Windows Script</b>  The Parent command is used to click a button on the Login window's parent.</p>	<pre># Login Dialog Box Dialog   Class #32770   Title "Login" EndDialog  Type \$Username #1001 Type \$Password #1002 Parent   Click #1 EndParent</pre>

# PickListAdd

Item	Description
<b>Use with:</b>	All
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Usage:</b>	PickListAdd <i>Display-Text</i> [ <i>Return-Value</i> ]
<b>Arguments:</b>	
<i>Display-Text</i>	The text that will be displayed in the pick list for the specified option.
[ <i>Return-Value</i> ]	The value returned from the pick list. If you don't specify this parameter, the return will be the display text.
<b>Description:</b>	<p>Allows users who have multiple accounts for a particular system to choose the account that they will log in to. Also, you can use this command set to choose from multiple sessions on one mainframe account.</p> <p>In fact, you can use PickList to build a list of databases, phone numbers, or any list you need your user to choose from. You can then set Variables or take action accordingly.</p> <p>Always use PickListAdd with the PickListDisplay command. Also, you typically use PickListAdd with the SetPlat command.</p>
<b>Example:</b>	
<b>Windows Script</b>	<pre># Login Dialog Box Dialog   Class #32770   Title "Login" EndDialog PickListAdd "Account One" "One" PickListAdd "Account Two" "Two" PickListAdd "Account Three" "Three" PickListDisplay ?Account "Select the account to use." -NoEdit SetPlat ?Account Type \$Username #1001 Type \$Password #1002 Click #1</pre>
The user has three accounts for this application and wants to be able to pick which one to use. The user picks an account to use, and SecureLogin (using the SetPlat command) switches to that set of credentials.	

# PickListDisplay

Item	Description
<b>Use with:</b>	Startup scripts, Terminal Launcher, Web, Windows
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Usage:</b>	PickListDisplay <i>?Variable Display-Text</i> [-NoEdit]



Item	Description
<b>Arguments:</b>	
?Variable	The output variable for the selected option.
Display-Text	The description text for the pick list box.
-NoEdit	Prevents users from adding entries to the pick list.
<b>Description:</b>	Displays 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, users can enter their own data into an edit field at the bottom of the pick list. You can turn off this feature by setting the -NoEdit flag.
<b>Syntax Examples:</b>	PickListDisplay ?Choice PickListDisplay ?Choice "Select the account you want to use" PickListDisplay ?Choice "Select the account you want to use" -NoEdit
<b>Example:</b>	# Login Dialog Box
<b>Windows Script</b>	Dialog
The user has three	Class #32770
accounts for this	Title "Login"
application and wants	EndDialog
to be able to pick which	PickListAdd "Account One" "One"
one to use. The user	PickListAdd "Account Two" "Two"
picks an account and	PickListAdd "Account Three" "Three"
SecureLogin (using the	PickListDisplay ?Account "Select the account to use." -NoEdit
SetPlat command)	SetPlat ?Account
switches to that set of	Type \$Username #1001
credentials.	Type \$Password #1002
	Click #1

## PositionCharacter

Item	Description
<b>Use with:</b>	Password Policy Scripts
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Usage:</b>	PositionCharacter [Numeral] [Uppercase] [Lowercase] [Punctuation] <i>position</i> , [ <i>position</i> ]...
<b>Arguments:</b>	
[numeral]	The character at <i>position</i> must be a numeral.
[uppercase]	The character at <i>position</i> must be an uppercase character.
[lowercase]	The character at <i>position</i> must be a lowercase character.
[punctuation]	The character at <i>position</i> must be a punctuation character.
<i>position</i>	The character <i>position</i> in the password.

Item	Description
<b>Description:</b>	Use this command in a password policy script to enforce that a certain character in the password be a numeral, uppercase, lowercase, or punctuation character.  You can specify multiple positions.
<b>Example:</b> The password won't be valid unless the first, sixth and seventh characters are in uppercase.	<code>PositionCharacter Uppercase 1,6,7</code>

## ReadText

Item	Description
<b>Use with:</b>	Terminal Launcher, Windows
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Windows Usage:</b>	<code>ReadText #Ctrl-ID ?Variable</code>
<b>Terminal Launcher Usage:</b>	<code>ReadText ?Variable Character-Number Row-Number</code>
<b>Arguments:</b>	
<i>#Ctrl-ID</i>	The control ID number of the text to be read.
<i>?Variable</i>	The variable that will receive the text that is read.
<i>Character-Number</i>	The number of characters to be read.
<i>Row Number</i>	The horizontal position number of the first character to be read (for example, row).
<b>Description:</b>	<p>Runs in both Windows and Terminal Launcher scripts. Although the usage and arguments for the use of ReadText with Windows and Terminal Launcher are different, the results of each command are the same.</p> <p>In a Windows script, the ReadText command reads the text from any given <i>#Ctrl-ID</i> and sends it to the specified variable. For this command to function correctly, the <i>#Ctrl-ID</i> must be valid.</p> <p>In a Terminal Launcher script, the ReadText command reads a specified number of characters, starting at the <i>Row-Number</i>, and sends those characters to the specified <i>Variable</i>.</p> <p>The ReadText command won't 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.</p>
<b>Syntax Examples:</b>	<code>ReadText #301 ?Text</code> <code>ReadText ?Text 4 6</code>

Item	Description
<b>Example 1: Windows Script</b>	# Login Failed Message
The same Title and Class appear in the error message dialog box when a user fails to log in. To distinguish among errors and provide users with more specific information (rather than a general message stating that the username and password are incorrect, or the account is locked), SecureLogin can read the actual error message, clear it by clicking OK, and prompt the user with a customized message.	Dialog Class #32770 Title "Login Failed" EndDialog ReadText #65535 ?ErrorMsg Click #1 If "Invalid Username" -In ?ErrorMsg DisplayVariables "Verify your username and try again." \$Username Type \$Username #1001 Type \$Password #1002 Click #1 EndIf If "Invalid Password" -In ?ErrorMsg DisplayVariables "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. Contact the Helpdesk at x3849." Endscript EndIf
<b>Example 2: Windows Script</b>	# Login Dialog Box
Read the text from a control ID and set the Database variable so the user isn't prompted to.	Dialog Class #32770 Title "Login" EndDialog ReadText #15 ?Database If -Exists \$Database Else Set \$Database ?Database EndIf Type \$Username #1001 Type \$Password #1002 Type \$Database #1003 Click #1
<b>Example 3: Terminal Launcher Script</b>	ReadText ?Message 30 24 2 MessageBox ?Message
Read a message in a Terminal Emulator and display it in a user-friendly format.	

# RegSplit

Item	Description
<b>Use with:</b>	All
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Usage:</b>	RegSplit <i>RegEx</i> <i>Input-String</i> [ <i>Output-String1</i> <i>Output-String2</i> ]...
<b>Arguments:</b>	
<i>RegEx</i>	The regular expression.
<i>Input-String</i>	The string that will be split.
<i>Output-String1</i>	The first subexpression.
<i>Output-String2</i>	The second subexpression.
<b>Description:</b>	Enables you to split a string by using a regular expression. <i>Output-String1</i> contains the first subexpression. <i>Output-String2</i> contains the second subexpression.  For information on regular expressions, see <a href="http://msdn.microsoft.com/library/en-us/vcsample98/html/vcsmpspy.asp?frame=truehttp://etext.lib.virginia.edu/helpsheets/regex.html">http://msdn.microsoft.com/library/en-us/vcsample98/html/vcsmpspy.asp?frame=truehttp://etext.lib.virginia.edu/helpsheets/regex.html</a> .
<b>Example:</b>	<pre># Login Dialog Box Dialog   Class #32770   Title "Login" EndDialog ReadText #65535 ?Text RegSplit "Enter the password for (.*?) account" ?Text ?User SetPlat ?User Type \$Username #1001 Type \$Password #1002 Click #1</pre>
<b>Windows Script</b>	
The text from control ID #301 is copied to the ?Text variable. The RegSplit command then strips the username details out of the text that was read. The user ID is set to that username, and SecureLogin enters the correct password	
<b>Open-Text Example:</b>	<pre>##?InputString: "This is a long string with a few components in it"</pre>
<b>Command:</b>	<pre>RegSplit "This (.*?) a long (.*?) with (.*?) components (.*?)" ?InputString ?First ?Second ?Third ?Fourth</pre>
<b>Result:</b>	<pre>?First = "is", ?Second = "string", ?Third = "a few", ?Fourth = "in it"</pre>

# ReloadPlat

For each dialog block in a script, the chosen User ID is reset and must therefore be reselected, either via a SetPlat command or by having the user reselect it from a list.

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

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

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

---

Item	Description
<b>Use with:</b>	Startup scripts, Terminal Launcher, Web, Windows
<b>SecureLogin Version:</b>	3.51.2
<b>Type:</b>	Action
<b>Usage:</b>	Use ReLoadPlat at:  <b>Login.</b> Before the user first logs into the application, call ReLoadPlat. This prevents the user from needing to reselect a user ID after a failed login.  <b>Failed Login.</b> Call ReLoadPlat to reselect the user ID that contained the incorrect credentials. Then give the user an opportunity to change the credentials by using either a ChangePassword or a DisplayVariables command.
<b>Arguments:</b>	None
<b>Description:</b>	Sets the current user ID to the last one chosen by the script, or leaves the user ID unset if an ID hasn't been selected.
<b>Example</b>	For an example script that uses ReloadPlat, see the example script in <a href="#">"ClearPlat" on page 42.</a>

---

## Repeat / EndRepeat

---

Item	Description
<b>Use with:</b>	All
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Usage:</b>	Repeat [Loop#] EndRepeat
<b>Arguments:</b>	
[Loop#]	The number of times the repeat script block is repeated. If you don't specify a number, the repeat continues indefinitely unless you use other commands to break the loop.

---

Item	Description
<b>Description:</b>	The Repeat command establishes a script block similar to the If command. The EndRepeat command terminates the repeat block. To break out of a repeat block, use the Break or EndScript command.
<b>Syntax Examples:</b>	Repeat Repeat 3
<b>Example:</b>	# Initial System Login
<b>Terminal Script</b>	WaitForText "ogin:"
The Repeat command	Type \$Username
watches the screen for	Type @E
messages and	WaitForText "assword:"
responds accordingly.	Type \$Password
The Break command	Type @E
jumps to the next	Delay 500
repeat loop in the	
script.	# Repeat loop for error handling
	Repeat
	#Check to see if the 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 or password (or both) stored.
	If -Text "Login Failed"
	DisplayVariables "The username or password (or both)
	stored by SecureLogin is invalid. Verify your credentials and
	try again. IT x453."
	Type \$Username
	Type @E
	Delay 500
	WaitForText "assword:"
	Type \$Password
	Type @E
	Delay 500
	EndIf
	# Account is locked for some reason, possibly inactive.
	If -Text "Account Locked"
	MessageBox "Your account has been locked, possibly
	because of inactivity for 40 days. Contact the administrator
	at x453."
	EndIf
	# Main Menu, user has logged in successfully.
	If -Text "Application Selection"
	Break
	EndIf
	Delay 100
	EndRepeat

# RestrictVariable

---

Item	Description
<b>Use with:</b>	All
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Usage:</b>	RestrictVariable <i>Variable-Name Password-Policy</i>
<b>Arguments:</b>	
<i>Variable-Name</i>	The name of the variable to restrict.
<i>Password-Policy</i>	The name of the policy to enforce on the variable.
<b>Description:</b>	<p>Monitors a <i>Variable</i> and enforces a specified <i>Password Policy</i> on the <i>Variable</i>. Any variable specified must match the policy or it won't be saved.</p> <p>When restricting variables to policies, be aware of the following information if you are making a tighter policy than is already in place. If you restrict a variable that doesn't match the policy today, the user won't be able to save it the first time. (When SecureLogin detects that there is no saved credential, a user who has a password of 6 characters today won't be able to save it if the policy restricts the \$Password variable to eight characters and two numbers.)</p> <p>Example 2 tells how to work around this issue. Instead of restricting the \$Password variable, restrict a new password variable (?NewPwd). The User will be able to store an existing password the first time that SecureLogin prompts for the credentials. Also, SecureLogin enforces the stronger password policy when the password expires in x days.</p> <p>You can restrict any variable by 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, the \$Username might need to be lowercase, or \$Database might need to be six characters with no numbers.</p>

---

Item	Description
<p><b>Example 1:</b>  <b>Windows Script</b>  The script restricts the \$Password variable to the Finance password policy. When the user first saves login credentials, the user's password must match the policy. When the password requires changing, the script randomly generates a new password based on that policy. No user intervention is required.</p>	<pre># Set the Password to use the Finance Password Policy RestrictVariable \$Password FinancePwdPolicy  # Login Dialog Box Dialog     Class #32770     Title "Login" 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</pre>
<p><b>Example 2:</b>  <b>Windows Script</b>  The script restricts the ?NewPwd variable to the Finance password policy. When the application starts for the first time and prompts the user to enter credentials, the user's current password (\$Password) is saved and used. When the password expires, the password policy is enforced on any new password. If you can't guarantee that all existing passwords meet the new policy, this is a graceful way to enforce tougher password policies than are currently in place.</p>	<pre># Set the Password to use the Finance Password Policy RestrictVariable ?NewPwd FinancePwdPolicy  # Login Dialog Box Dialog     Class #32770     Title "Login" 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</pre>

## Run

Item	Description
<b>Use with:</b>	Startup scripts, Terminal Launcher, Web, Windows



Item	Description
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Usage:</b>	Run <i>Command</i> [ <i>Arg1</i> [ <i>Arg2</i> ] ...]
<b>Arguments:</b>	
<i>Command</i>	The full path of the program to be executed.
[ <i>Arg1</i> [ <i>Arg2</i> ]...]	An optional list of arguments or switches for the command.
<b>Description:</b>	Launches the program specified in the <i>Command</i> with the specified optional [ <i>Arg1</i> [ <i>Arg2</i> ] ...] arguments. The script doesn't wait for the launched program to complete.
<b>Example:</b>	<pre> 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 </pre>
<b>Startup Script</b>	
The user is prompted to start the Finance System. If the user clicks Yes, the Run command (with the necessary switches) starts the application. If the user clicks No, a message box is displayed, and the application isn't started.	

## SelectListBoxItem

Item	Description
<b>Use with:</b>	Advanced Web Scripts
<b>SecureLogin Version:</b>	3.5
<b>Type:</b>	Action
<b>Usage:</b>	SelectListBoxItem <i>Text of Item to set to</i> [ <i>item Number</i> ] [- <i>multiselect</i> ]
<b>Arguments:</b>	
<i>Text of Item to set to</i>	The text item that you want SecureLogin to select in the list box.
[ <i>Item Number</i> ]	When multiple list boxes are found, this specifies which list box to address.
[- <i>multiselect</i> ]	Used to select multiple list box entries by using a subsequent SelectListBoxItem command.
<b>Description:</b>	Selects entries from a list box. For information on determining item numbers, see <a href="#">"DumpPage" on page 51</a> .

Item	Description
<b>Example:</b>	SelectListBoxItem "Remember Defects" #2 -multiselect SelectListBoxItem "Remember Enhancements" #2 -multiselect

## SendKey

Item	Description
<b>Use with:</b>	Terminal Launcher
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Usage:</b>	SendKey <i>Text</i>
<b>Arguments:</b>	
<i>Text</i>	The text to be typed into the emulator screen.
<b>Description:</b>	<p>Works only with Generic and Advanced Generic emulators. Use SendKey in the same manner as the Type command.</p> <p>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.</p> <p>Variables don't work with the SendKey command. If you want to use variables, use the Type command.</p> <p>The Type command has many special functions, some of which can also be used with the SendKey command. For information on these functions, see <a href="#">"Type" on page 98</a>.</p>
<b>Example:</b>	#Send Username
<b>Terminal Launcher Script</b>	SendKey mkurz
	SendKey "\N"
The SendKey command sends the username and password to the terminal emulator.	#Send Password
	SendKey "Hu7%f"
	SendKey "\N"

## Set

Item	Description
<b>Use with:</b>	All
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action

Item	Description
<b>Usage:</b>	<i>Set Variable Data</i>
<b>Arguments:</b>	
<i>Variable</i>	The variable that the data is being assigned to.
<i>Data</i>	The text or variable being read from and assigned to the variable.
<b>Description:</b>	Copies the value of <i>Data</i> into <i>Variable</i> . The <i>Data</i> can be any text or another variable. However, the <i>Variable</i> argument must be a ?Variable or \$Variable.
<b>Example 1: Windows Script</b> The script sets a ?RunCount variable to count the number of times the application is run.	<pre># Login Dialog Box Dialog     Class #32770     Title "Login" EndDialog  If ?RunCount Eq <i>NOTSET</i>     Set ?RunCount "1" Else     Increment ?RunCount EndIf  Type \$Username #10091 Type \$Password #1002 Click #1</pre>
<b>Example 2: Windows Script</b> The script sets the ?NewPwd to the stored variable, \$Password.	<pre># Login Dialog Box Dialog     Class #32770     Title "Login" 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</pre>

Item	Description
<b>Example 3:</b> <b>Windows Script</b> The script reads the value of Ctrl #15 and sets the \$Database variable so that the user doesn't need to.	<pre># Login Dialog Box Dialog   Class #32770   Title "Login" EndDialog  ReadText #15 ?Database If -Exists \$Database   Else     Set \$Database ?Database EndIf</pre>

## SetCheckBox

Item	Description
<b>Use with:</b>	Advanced Web Script
<b>SecureLogin Version:</b>	3.5
<b>Type:</b>	Action
<b>Usage 1:</b>	SetCheckBox <i>Item Number Option</i>
<b>Arguments:</b>	
<i>Item Number</i>	The check box in reference to the number of check boxes found.
<i>Option</i>	Specifies the status of the check box as Checked or Unchecked.
<b>Description:</b>	Allows the selection of a check box to be checked or unchecked.

Item	Description
<b>Example:</b>	<pre> MessageBox "Scroll down so you can see the 'Search Language' section and all the languages with the check boxes, then click OK on this message box." SetCheckBox #1 "checked" SetCheckBox #2 "checked" SetCheckBox #3 "checked" SetCheckBox #4 "checked" SetCheckBox #25 "checked" SetCheckBox #26 "checked" SetCheckBox #27 "checked"  MessageBox "Did it select the first four languages and Norwegian, Polish and Portuguese languages" -yesno ?advweb  If ?advweb Eq yes   Set ?cmd37 "SetCheckBox command worked" Else   Set ?cmd37 "SetCheckBox failed" EndIf  SetCheckBox #1 "unchecked" SetCheckBox #2 "unchecked" SetCheckBox #3 "unchecked" SetCheckBox #4 "unchecked" SetCheckBox #26 "unchecked" SetCheckBox #27 "unchecked"  MessageBox "Did it unselect all the languages except Norwegian" -yesno ?advweb2 If ?advweb2 Eq yes   Set ?cmd38 "SetCheckBox command worked" Else   Set ?cmd38 "SetCheckBox failed" EndIf </pre>

## SetCursor

Item	Description
<b>Use with:</b>	Terminal Launcher (Only available in HLLAPI and some DDE emulators)
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Usage 1:</b>	SetCursor <i>Screen-Position</i>
<b>Usage 2:</b>	SetCursor <i>X Coordinate Y Coordinate</i>
<b>Arguments:</b>	
<i>Screen-Position</i>	On the screen, the position that the cursor should be moved to.
<i>X Coordinate</i>	The horizontal coordinate. When <i>X-Coordinate</i> is specified, a row/column conversion is carried out before the cursor is set to the position.
<i>Y Coordinate</i>	The vertical coordinate. When <i>Y-Coordinate</i> is specified, a row/column conversion is carried out before the cursor is set to the position.

Item	Description
<b>Description:</b>	Sets the cursor to a specified <i>Screen Position</i> or <i>X Coordinate Y Coordinate</i> . The position will be noted by a number greater than 0 (for example, SetCursor 200). If the screen position is invalid, Terminal Launcher displays an error message.
<b>Syntax Examples:</b>	SetCursor 200 SetCursor 100 500
<b>Example:</b>	SetCursor 200
<b>Terminal Launcher Script</b>	Type \$Username Type @E The cursor is set to the correct position, then the credentials are entered.
	Type \$Password Type @E

## SetFocus

Item	Description
<b>Use with:</b>	Java, Web, Windows
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Usage:</b>	SetFocus #Ctrl-ID
<b>Arguments:</b>	
<i>#Ctrl-ID</i>	The ID number of the control that the keyboard focus will be directed to.
<b>Description:</b>	Hands the keyboard focus to a specified <i>#Ctrl-ID</i> .  For the SetFocus command to function correctly, the <i>#Ctrl-ID</i> must be valid.
<b>Example:</b>	# Login Dialog Box
<b>Windows Script</b>	Dialog
SecureLogin sets the focus to the username field (#1001), types the username, simulates the tab, types the password, then simulates Enter.	Class #32770 Title "Login" EndDialog  SetFocus #1001 Type \$Username Type \T Type \$Password Type \N

## SetPlat

Item	Description
<b>Use with:</b>	All

Item	Description
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Usage 1:</b>	SetPlat <i>Application-Name</i>
<b>Usage 2:</b>	SetPlat <i>RegEx Variable #Ctrl-ID</i>
<b>Arguments:</b>	
<i>Application-Name</i>	The application name that the variables are read from.
<i>RegEx</i>	A regular expression to be used as the application name.
<i>Variable</i>	Must be a ?Variable previously set (for example, using a pick list).
<i>#Ctrl-ID</i>	The control ID of the regular expression to be used.
	For information on regular expressions, see <a href="http://msdn.microsoft.com/library/en-us/vcsample98/html/vcsmpspy.asp?frame=truehttp://etext.lib.virginia.edu/helpsheets/regex.html">http://msdn.microsoft.com/library/en-us/vcsample98/html/vcsmpspy.asp?frame=truehttp://etext.lib.virginia.edu/helpsheets/regex.html</a> .
<b>Description:</b>	<p>By default, variables are stored directly against the platform or application that you have enabled for single sign-on. For example, if you enable Groupwise.exe, the Groupwise® credentials are stored against the Groupwise.exe application.</p> <p>You might have multiple accounts (for example, your own login and an Admin login) accessing the same application. Or you might have multiple applications using a common set of credentials. In these cases, SetPlat sets the application that the variables are read to and saved from.</p> <p>You can also use SetPlat to do the following:</p> <ul style="list-style-type: none"> <li>♦ Script application1 to read its \$Username and \$Password from application2. This saves a user entering the credentials twice and having to remember to update them in both locations when the credentials change.</li> <li>♦ Script application1, application2, and application3 to read their credentials from Platform "Common." This means that you have a single store of common information that only needs to be updated once.</li> <li>♦ Create new applications, depending on what a user selected in a pick list. If the <i>Application-Name</i> doesn't exist, it will be created.</li> </ul> <p>SetPlat can also read from a <i>#Ctrl-ID</i> and support regular expressions.</p>

Item	Description
<b>Example 1:</b>	# Login Dialog Box
<b>Windows Script</b>	Dialog
SecureLogin displays a pick list and sets a new user ID so that multiple users can log in to the application. In this case, SetPlat creates a new use ID called Default User, Global Administrator, or Regional Administrator. The respective \$Username and Password are saved there.	Class #32770 Title "Login" EndDialog  PickListAdd "Default User" PickListAdd "Global Administrator" PickListAdd "Regional Administrator" PickListDisplay ?Choice "Select the account to use." -NoEdit  SetPlat ?Choice Type \$Username #1001 Type \$Password #1002 Click #3

### Example 2: Web Script

The following figure illustrates a standard dialog box for accessing a password-protected site using Netscape Navigator.



When you specify the Title, Class, Username, and Password fields of this dialog box, they will always be the same. If you stored the Username and Password against this application without using the SetPlat command, the Username and Password for www.novell.com would be entered to log in to any site (and are invalid for any other site).

However, the dialog box pictured above always contains the name of the Web site to log in to. This can be used as the unique identifier, to set a new SecureLogin platform, and save login credentials to.

The solution to this problem is to use a dialog block with a SetPlat statement similar to the following:

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



```
Type $Username #214
Type $Password #330
Click #1
```

The power of this script is in the following line:

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

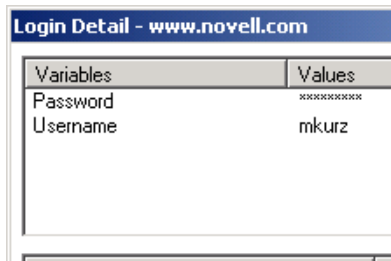
The script first reads the following line from dialog control ID 331:

```
Enter username for Control Panel at www.novell.com:
```

The script then applies the regular expression to this text. Regular expressions are a powerful way to manipulate text strings. However, for most purposes you can use the basic commands listed in the following table:

Basic Command	Action
* (an asterisk)	Matches any character
. (a period)	Matches zero or more of the preceding character
( ) (parentheses)	Makes the contents of the parentheses a subexpression

After running the script, the user sees the username and password saved as www.novell.com.



The text that is matched inside the parentheses then becomes the symbol application. If a dialog *#Ctrl-ID* is not specified, the symbol application is unconditionally changed to the application specified in the *RegEx*. An unconditional *SetPlat* command is only valid if specified before *Dialog/EndDialog* statements.

## SetPrompt

Item	Description
<b>Use with:</b>	All
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Usage:</b>	SetPrompt <i>Prompt-Text</i>

Item	Description
<b>Arguments:</b>	
<i>Prompt-Text</i>	The customized text prompt to be displayed in the Enter SecureLogin Variables dialog box.
<b>Description:</b>	<p>Customizes the text in the Enter SecureLogin Variables dialog boxes that are used to prompt the user for new variables. For Variables that have been stored previously, you can also use the DisplayVariables command to customize the prompt text in the dialog box.</p> <p>Positioning of the setprompt command is crucial. SetPrompt must be before the first usage of each variable to name that variable, and the final SetPrompt is applied to the text displayed at the top of the prompt screen).</p>
<b>Example 1: Windows Script</b>	<pre># Login Dialog Box Dialog   Class #32770   Title "Login" EndDialog  Type \$Username #1001 Type \$Password #1002 Click #1  SetPrompt "Enter your username and password for accessing the Human Resources system. In the future, SecureLogin will remember these credentials and automatically log you in. IT Helpdesk x4564"</pre>
To replace the default text prompt in the Enter SecureLogin Variables dialog box, place the SetPrompt command at the bottom of the script	
<b>Example 2: Windows Script</b>	<pre># Login Dialog Box Dialog   Class #32770   Title "Login" EndDialog  SetPrompt "Enter Username==&gt;" Type \$Username #1001  SetPrompt "Enter Password==&gt;" Type \$Password #1002 Click #1  SetPrompt "Enter your username and password for accessing the Human Resources system. In the future, SecureLogin will remember these credentials and automatically log you in. IT Helpdesk x4564"</pre>
To replace the text prompt next to any variable entry field in the Enter SecureLogin Variables box, place the SetPrompt command immediately before the variable in the script.	

## Strcat

Item	Description
<b>Use with:</b>	All
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action

Item	Description
<b>Usage:</b>	Strcat <i>Variable Input-String1 Input-String2</i>
<b>Arguments:</b>	
<i>Variable</i>	The variable that you want the result saved to.
<i>Input-String1</i>	The first data string or variable.
<i>Input-String2</i>	The second data string or variable.
<b>Description:</b>	Appends the second data string to the first data string.
	<b>Scenario:</b> You include the following line in a script:
	<code>StrCat ?Result "SecureRemote" "\$Username"</code>
	When \$Username is Tim, the ?Result variable contains the value "SecureRemote Tim."
<b>Example:</b>	
<b>Windows Script</b>	<code># Login Dialog Box</code>
The username is read from #1001 into ?Username. The StrCat command joins the username and the password. The result is a LoginID that SecureLogin uses to log the user in to the system.	<code>Dialog   Class #32770   Title "Login" EndDialog  Readtext #1001 ?Username StrCat ?LoginID ?Username \$Password Type ?LoginID #1002 Click #1</code>

## StrLength

Item	Description
<b>Use with:</b>	All
<b>SecureLogin Version:</b>	3.0.4
<b>Type:</b>	Variable manipulator
<b>Usage:</b>	StrLength <i>Destination String</i>
<b>Arguments:</b>	
<i>Destination</i>	The output variable. Also the input variable if no source is specified.
<i>String</i>	The string whose length is to be measured.
<b>Description:</b>	Counts the number of characters in a variable and outputs that value to the destination variable.

Item	Description
<b>Example:</b>	# Login Dialog Box
<b>Windows Script</b>	Dialog
The password is read from #301. StrLength is then used to count the number of characters. If the number is less than 4, an error message is displayed.	Class #32770 Title "Login" EndDialog  Readtext #301 ?Password StrLength ?Length ?Password If ?Length Lt "4" MessageBox "Password is too short" EndIf

## StrLower

Item	Description
<b>Use with:</b>	All
<b>SecureLogin Version:</b>	3.0.4
<b>Type:</b>	Variable manipulator
<b>Usage:</b>	StrLower <i>Destination</i> [ <i>Source</i> ]
<b>Arguments:</b>	
<i>Destination</i>	The output variable. Also the input variable if no source is specified.
[ <i>Source</i> ]	The input variable. If not specified, SecureLogin reads the destination variable, makes the necessary changes, and writes over it.
<b>Description:</b>	Modifies a variable so that all the characters are lowercase.  If only a destination variable is specified, the string is read from the destination, then stored back to it. If a 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.
<b>Example:</b>	#Login Dialog Box
<b>Windows Script</b>	Dialog
SecureLogin reads the username from #1001 and copies it into ?Username. The StrLower command is then used to ensure that the username is in all lowercase.	Class #32770 Title "Login" EndDialog  Readtext #1001 ?Username StrLower ?LowerCaseUsername ?Username Type ?LowerCaseUsername #1002 Click #1

# StrUpper

---

Item	Description
<b>Use with:</b>	All
<b>SecureLogin Version:</b>	3.0.4
<b>Type:</b>	Variable manipulator
<b>Usage:</b>	StrUpper <i>Destination</i> [ <i>Source</i> ]
<b>Arguments:</b>	
<i>Destination</i>	The output variable. Also the input variable if no source is specified.
[ <i>Source</i> ]	The input variable. If you don't specify a source, SecureLogin reads the destination variable, makes the necessary changes, and writes over the variable.
<b>Description:</b>	Modifies a variable so that all the characters are uppercase.  If only a destination variable is specified, the string is read from the destination, then stored back to it. If a 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.
<b>Example:</b>	<pre># Login Dialog Box Dialog   Class #32770   Title "Login" EndDialog Readtext #1001 ?Username StrUpper ?UpperCaseUsername ?Username Type ?UpperCaseUsername #1002 Click #1</pre>
<b>Windows Script</b>	SecureLogin reads the username from #1001 and copies it into ?Username. The StrLower command is then used to ensure that the username is in all uppercase.

---

# Sub / EndSub

---

Item	Description
<b>Use with:</b>	Startup scripts, Terminal Launcher, Web, Windows
<b>SecureLogin Version:</b>	2.5
<b>Type:</b>	Flow control
<b>Usage:</b>	Sub <i>Name</i> EndSub
<b>Arguments:</b>	
<i>Name</i>	Any name entered to identify the subroutine.
<b>Description:</b>	Used around a block of lines within a script to denote a subroutine. You can call a subroutine by using the Call command.

---

Item	Description
<b>Example:</b>	If -Text "Login"
<b>Terminal Launcher</b>	Call Login
<b>Script</b>	EndIf
The emulator screen is checked for the text "Login" or "Wrong Password." If either of these is found, the appropriate subroutine is called and run before the next part of the script.	If -Text "Wrong Password" Call "WrongPassword" EndIf
	Sub Login Type \$Username Type @E Type \$Password Type @E EndSub
	Sub WrongPassword DisplayVariables "Enter correct password" \$Password Call Login EndSub

## Submit

Item	Description
<b>Use with:</b>	Web
<b>SecureLogin Version:</b>	3.0
<b>Type:</b>	Action
<b>Usage:</b>	Submit
<b>Arguments:</b>	None

Item	Description
<b>Description:</b>	<p>Use the Submit command only in Web scripts 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 that the first password field is found in.</p> <p>When used with Netscape, the Submit command is ignored.</p> <p>By default, the function performed by the Submit command is automatically performed by a Web script. For example, the following script types the username, types the password, and submits the form:</p> <pre>Type \$Username Type \$Password Password</pre> <p>Submits don't automatically occur if any of the following commands are in the script. If any one of these commands is used, you must use the Submit command:</p> <ul style="list-style-type: none"> <li>◆ Type \N</li> <li>◆ Type \T</li> <li>◆ Submit</li> <li>◆ Click</li> </ul> <p>If one of these commands is in the script, you must use the Submit command or some other means to cause the form to be submitted.</p> <p>Furthermore, an automatic submit won't occur if you type text into a specific text entry field. For example, in the following script segment the Submit command must follow the Type command for the script to work properly:</p> <pre>Type \$Username #1001  Submit</pre>
<b>Example: Web Script</b>	<pre>Type \$Username #1 Type \$Password Password #2 Submit</pre>
The script enters the username and password, then performs a manual Submit.	

## Subtract

Item	Description
<b>Use with:</b>	Startup scripts, Terminal Launcher, Web, Windows
<b>SecureLogin Version:</b>	3.0
<b>Type:</b>	Variable manipulator
<b>Usage:</b>	Subtract <i>Start-Value Subtract-Value</i> [? <i>Result</i> ]

Item	Description
<b>Arguments:</b>	
<i>Start-Value</i>	The start number that the second argument will be subtracted from. This argument will contain the result if the optional [?Result] argument is not passed in.  If you use the <i>Start-Value</i> argument without the [?Result] argument, <i>Start-Value</i> must be a SecureLogin variable (for example, ? <i>Start-Value</i> or \$ <i>Start-Value</i> ). If the [?Result] argument is provided, <i>Start-Value</i> can be a SecureLogin variable or a numeric value.
<i>Subtract-Value</i>	The number that will be subtracted from the first argument. <i>Subtract-Value</i> can be a SecureLogin variable or a numeric value.
[?Result]	Optional. The result of the equation. If you use this argument, set it to Start-Value - Subtract-Value. The [?Result] must be a SecureLogin variable (for example, \$Result or ?Result).
<b>Description:</b>	Subtracts one value from another. This can be useful if you are implementing periodic password change functionality for an application. The subtract command (in conjunction with the Divide function and the slinac .dll file) can be used to determine the number of days that have elapsed since the last password change.  The Subtract command correctly subtracts when <i>Start-Value</i> , <i>Subtract-Value</i> , and <i>Result</i> are between -2147483648 and +2147483647.  Subtract doesn't work with fractions.
<b>Syntax Examples:</b>	Subtract "1" "2" ?Result Subtract ?LoginAttempts ?LoginFailures Subtract ?LoginAttempts ?LoginFailures ?Result Subtract ?LoginAttempts "3" Subtract ?LoginAttempts "3" ?Result
<b>Example:</b> <b>Windows Script</b> The values of Control IDs 103 and 104 are read 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

## Tag/EndTag

Item	Description
<b>Use with:</b>	Advanced Web Script
<b>SecureLogin Version:</b>	3.5
<b>Type:</b>	Tag specifier
<b>Usage:</b>	Tag EndTag



Item	Description
<b>Arguments:</b>	None
<b>Description:</b>	Used to find HTML tags.
<b>Example:</b> SecureLogin finds the form that has an attribute of "name" with a value of "login".	<pre> Tag "Form"   Attribute "Name" "Login" EndTag </pre>

## Title

Item	Description
<b>Use with:</b>	Java, Windows
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Dialog specifier
<b>Usage:</b>	Title <i>Window-Title</i>
<b>Arguments:</b>	
<i>Window-Title</i>	The text to test against the window title.
<b>Description:</b>	<p>Retrieves the title of the window and compares it against the string specified in the <i>Window-Title</i> argument. For this block of the script to run, the retrieved window title and the <i>Window-Title</i> argument must match exactly.</p> <p>Title is one of the main commands that you can use to identify a window. However, just using the Title command alone may not be enough. If an application has more than one window with the specified title, the SecureLogin script will run every time that window is detected.</p> <p>To uniquely identify a window, typically use the Title command with the Class or Ctrl command. In any Dialog statement, place the Class command before the Title command.</p> <p>You can use Window Finder to locate the window title. See <a href="#">"Finding Control IDs" on page 28</a>.</p>
<b>Example:</b> <b>Windows Script</b> The dialog box is tested to see whether it has the correct title. If the title isn't correct, the script passes on to the next Dialog block.	<pre> # Login Dialog Box Dialog   Class #32770   Title "Login" EndDialog  Type \$Username #1001 Type \$Password #1002 Click #1 </pre>

# Type

Item	Description
<b>Use with:</b>	Java, Terminal Launcher, Web, Windows
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Action
<b>Terminal Launcher Usage:</b>	Type [-Raw] <i>Text</i>
<b>Windows Usage:</b>	Type <i>Text</i> [#Ctrl-ID] Type [-Raw] <i>Text</i>
<b>Web Usage:</b>	Type <i>Text</i> [#Field-ID] Type <i>Text</i> [Password]
<b>Arguments:</b>	
[-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,. 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.
<i>Text</i>	The text to type into this area. This text can be either static text, such as ABC, or any SecureLogin variable, such as \$Username.
[#Ctrl-ID]	For Windows scripts, this optional argument specifies the control that the text will be typed in to. You can use Window Finder to extract these control IDs. See <a href="#">“Windows-Specific Information” on page 99</a> .
[Field-ID]	For Web scripts, this optional argument specifies the text field that the text will be typed in to. See <a href="#">“Web-Specific Information” on page 99</a> .
[Password]	For Web scripts, this optional argument specifies that this type should be performed into the password field on this form. If you use [Password], that application's script cannot use a #Ctrl-ID argument. See <a href="#">“Web-Specific Information” on page 99</a> .

Item	Description
<b>Description:</b>	<p data-bbox="617 159 1449 185">Used to enter data (for example, usernames and passwords) into applications.</p> <p data-bbox="617 211 1449 294">Some character sequences are reserved. These character sequences are used to type special characters such as Tab and Enter. See <a href="#">Chapter 7, “Keystrokes and Functions,”</a> on page 115.</p> <p data-bbox="617 320 1449 374">If you can’t determine control IDs in a Windows application, and the Type command is not working, you can use the SendKey command.</p> <p data-bbox="617 401 960 423"><b>Windows-Specific Information</b></p> <p data-bbox="617 449 1449 562">In Windows, if the <i>#Ctrl-ID</i> argument is provided, it must be a number that refers to a control ID as identified by Window Finder. SecureLogin then sends the contents of the <i>Text</i> argument directly to the window and to the specific control that matches the <i>#Ctrl-ID</i> argument</p> <p data-bbox="617 588 1449 643">If the <i>Ctrl-ID</i> is not specified, SecureLogin sends keystrokes to whichever control has focus. In the Windows environment, the -Raw option is often useful when</p> <ul data-bbox="617 669 1449 764" style="list-style-type: none"> <li>♦ Window Finder is unable to determine control IDs for the text entry areas of an application</li> <li>♦ These control IDs are changing.</li> </ul> <p data-bbox="617 790 1299 812">When you use the -Raw option, don’t use the <i>#Ctrl-ID</i> argument.</p> <p data-bbox="617 838 907 860"><b>Web-Specific Information</b></p> <p data-bbox="617 887 1449 1030">You can use either of two methods to specify which field receives <i>Text</i>. The first method uses absolute positioning through the <i>#Field-ID</i> argument. The <i>#Field-ID</i> 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, and #2 refers to the second text entry field.</p> <p data-bbox="617 1056 1449 1199">The second method uses relative position via the [Password] argument. In this method, the SecureLogin agent first locates the text field that is a password field within the HTML form. Then the <i>Text</i> is typed into that field. Other Type commands send their <i>Text</i> parameters to fields that are relative to the first password field.</p> <p data-bbox="617 1225 1449 1308">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 the example Web script.</p> <p data-bbox="346 1340 560 1483"><b>Example 1: Windows Script</b> A typical use of the Type command in a Windows script.</p> <pre data-bbox="606 1340 894 1620"> # Login Dialog Box Dialog     Class #32770     Title "Login" EndDialog  Type \$Username #1001 Type \$Password #1002 Type "DB2" #1003 Click #1 </pre>

Item	Description
<p><b>Example 2:</b>  <b>Windows Script</b>            This typical example shows the use of the -Raw switch.</p>	<pre># Calculator Is Active Dialog   Class #SciCalc   Title "Calculator" EndDialog</pre>
<p>(The -Raw switch is not actually required in this instance. It is only an example.)</p>	<pre>Type -Raw "15" Type -Raw "+" Type -Raw "20" Type -Raw "="</pre>
<p><b>Example 3:</b>  <b>Web Script</b>            The SecureLogin agent automatically generated this script for the mail.yahoo.com site. This example shows the use of Password as the [Password] argument.</p>	<p>In the following script, 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.</p> <pre>Type \$Username Type \$Password Password</pre> <p>The same script could be rewritten using absolute placement, as shown below. In this example, the Submit command is also used to automatically submit the page.</p> <pre>Type \$Username #1 Type \$Password #2 Submit</pre>

## Sending Keyboard Commands by Using Type

### Type Commands Used with Windows

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 script is working in. These special commands include the ability to select Menu items, send Alt, and send other keyboard combinations.

#### Special Key Commands

Commands	Description
Type \Alt+key	Simulates pressing the Alt key plus the desired key.
Type \Shift+key	Simulates pressing the Shift key plus the desired key.
Type \Ctrl+key	Simulates pressing the Ctrl key plus the desired key.
Type \LWin+key	Simulates pressing the left Windows key plus the desired key.
Type \RWin+key	Simulates pressing the right Windows key plus the desired key.

Commands	Description
Type \Apps+key	Simulates pressing the Application key plus the desired <i>key</i> .

### Raw Key Commands

You can also use the Type command to send a combination of raw key commands.

Commands	Description
Type \ xxx	The format for sending a raw key command, where <i>xxx</i> represents the keyboard code. See <a href="#">Appendix D, “Keyboard Functions and Codes,”</a> on page 131.
Type \18+65	Simulates 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 table in “[@ Commands Used with Emulators](#)” on page 116 lists the @ commands that you can use in the SecureLogin script 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 script language in the following format:

- ◆ Type @ *command*
- ◆ WaitForText "Login:"
- ◆ Type \$Username
- ◆ Type @T
- ◆ Type \$Password
- ◆ Type @E

## WaitForFocus

Item	Description
<b>Use with:</b>	Windows
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Flow control
<b>Usage:</b>	WaitForFocus #Ctrl-ID [ <i>Repeat-Loops</i> ]

Item	Description
<b>Arguments:</b>	
<i>#Ctrl-ID</i>	The ID number of the control that will have the focus.
<i>[Repeat-Loops]</i>	The number of repeat loops that will run.
<b>Description:</b>	<p>Suspends running the script until the <i>#Ctrl-ID</i> has received keyboard focus or until the <i>Repeat-Loops</i> argument expires. The <i>Repeat-Loops</i> argument is an optional value. It defines the number of loop cycles that will run.</p> <p>The <i>Repeat-Loops</i> value defaults to 3000 loops if nothing is set. As soon as focus is received, the script continues.</p> <p>As the following line illustrates, by setting the figure to a negative number, you can set <i>Repeat-Loops</i> to never expire:</p> <pre>WaitForFocus "#1065" "-1"</pre> <p>If the <i>#Ctrl-ID</i> is set to 0 (zero), it loops until the window defined in the Dialog / EndDialog statement is given keyboard focus.</p> <p><b>NOTE:</b> Don't place WaitForFocus commands within Dialog / EndDialog statements.</p>
<b>Syntax Examples:</b>	<pre>WaitForFocus #301 WaitForFocus #301 "2000" WaitForFocus #301 "0" WaitForFocus #301 "-1"</pre>
<b>Example:</b>	<pre># Login Dialog Box Dialog     Class #32770     Title "Login" EndDialog WaitForFocus #301 "-1" Type \$Username Type \T Type \$Password Type \N</pre>
<b>Windows Script</b>	
After the Login dialog box has been detected, SecureLogin waits indefinitely for window #301 to get focus before entering the user credentials.	

## WaitForText

Item	Description
<b>Use with:</b>	Terminal Launcher
<b>SecureLogin Version:</b>	All
<b>Type:</b>	Flow control
<b>Usage:</b>	WaitForText <i>Text</i>
<b>Arguments:</b>	
<i>Text</i>	The text that the script is waiting for.

Item	Description
<p><b>Description:</b></p> <p><b>Example:</b></p> <p><b>Terminal Launcher</b> SecureLogin waits for the text "ogin:" to appear on the emulator screen before entering the username. SecureLogin then waits for "assword:" to be displayed before entering the password</p>	<p>Causes Terminal Launcher to wait for the specified <i>Text</i> to be displayed before continuing. This command is important because the user often wants to wait for particular text to be displayed on the screen before continuing. For example, it is important to wait for a username field to be displayed before attempting typing an actual username into it.</p> <p>The <i>Text</i> can appear anywhere on the terminal screen. The <i>Text</i> is usually case-sensitive, depending on the terminal emulator you are using. If the <i>Text</i> is written in the wrong case, Terminal Launcher pauses and tries to find the correct <i>Text</i> in the correct case, pausing until the terminal screen times out.</p> <p>If WaitForText isn't working, try leaving the initial letter off the <i>Text</i>, so that you avoid any conflict with case. For example, the following line will work regardless of whether "login" is presented on the terminal screen as "Login" or "login".</p> <pre>WaitForText "ogin"</pre> <p>However, WaitForText "Login" will only work if "login" is presented on the screen as "Login".</p> <p>Also, some terminal emulators won't correctly match text that is against the left margin of the window. Again, if you encounter this situation, try to match text without the leading character.</p> <pre>WaitForText "ogin:" Type \$Username Type @E WaitForText "assword:" Type \$Password Type @E</pre>





# 6

## Practicing Your Scripting Skills

This section enables you to practice your scripting skills by using the Password Test Application (PasswordTest.exe) and SecureLogin. Password Test Application replicates an application login panel. The script is a typical example of scripting for a Windows application.

### Using the Wizard to Create a Script

In this section, you will use the Add Application Wizard to create a script for Password Test Application.

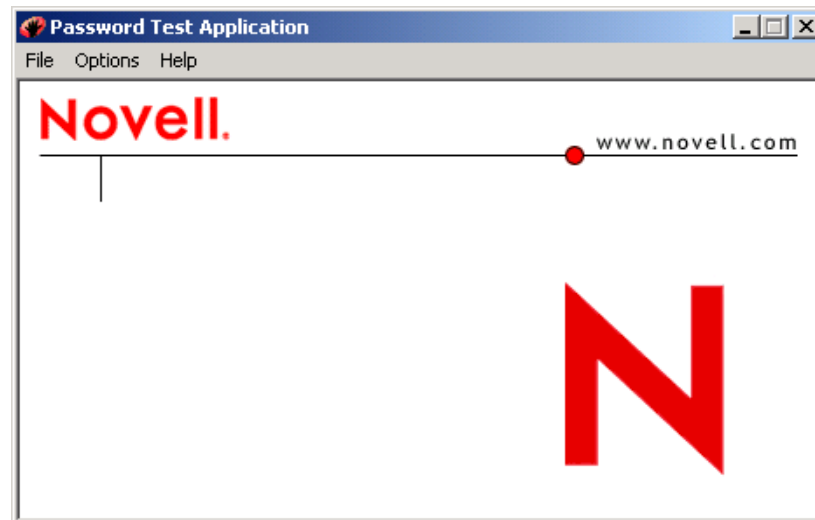
**1** Run SecureLogin.

Click Start > Programs > Novell SecureLogin > Novell SecureLogin. The SecureLogin icon is active on the system tray.

**2** Run PasswordTest.exe.

The file is in the \securelogin\tools directory.

The following figure illustrates Password Test Application's main window.



**3** Create login credentials for Password Test Application.

**3a** In the Password Test Application window, click File > Log In.

Because SecureLogin is active on the system tray, SecureLogin prompts you to use the Add Applications Wizard (SSO wizard) to add a login for Password Test.

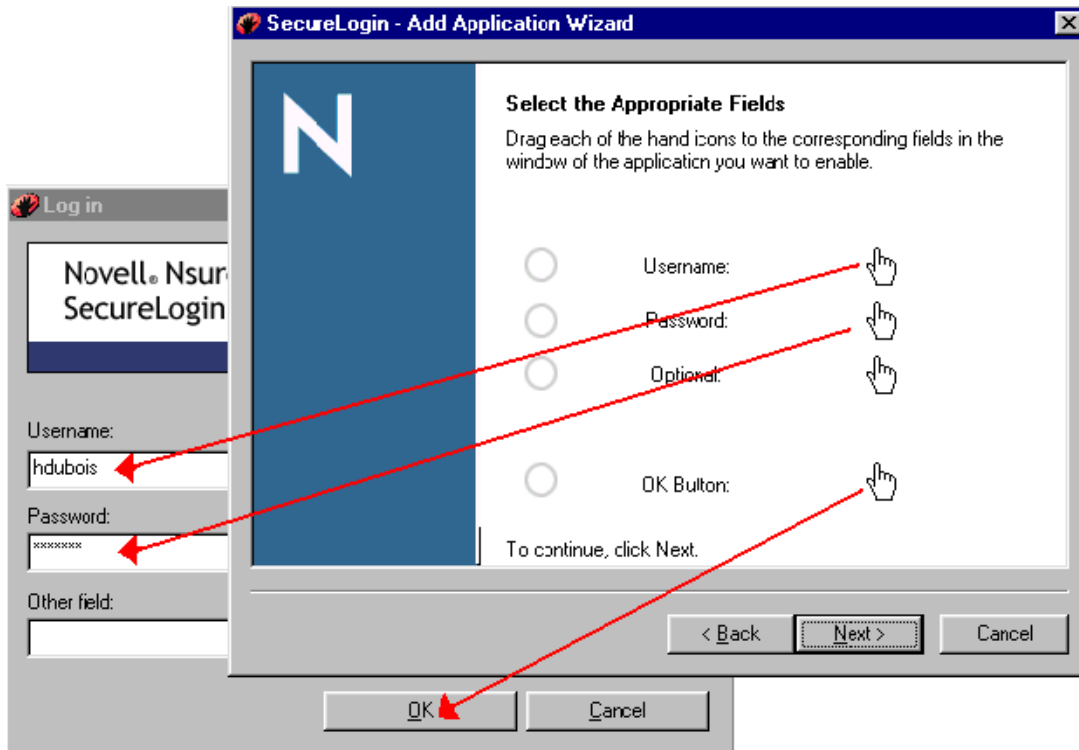


- 3b** Launch the Add Application Wizard by clicking Yes.
- 3c** In the Select Window Function dialog box, select Login Window, then click Next.
- 3d** Click and drag the Add Application Wizard to one side of your screen, so that you can also view and work with the PasswordTest.exe dialog box.
- 3e** Enter your username and password (novell) in the Password Text Application text boxes, but don't click OK.

**IMPORTANT:** The password for Password Test is "novell" until you change it. When you change the password, the new password is written to c:\passwordtest.txt. If you forget the current password, refer to this file or use Options > Display Password.

For this exercise, leave the Other Field text box blank.

- 3f** Drag hands from the Add Application Wizard to the corresponding Username, Password, and OK fields in the Password Test Application.



For this exercise, you don't need to drag a hand to the Other Field text box.

For additional details about the prompts and adding a login, see [“Enabling a Windows Application”](#) in the *Nsure SecureLogin 3.51.2 User Guide*.

- 3g** In the Add Application Wizard, click Next.
- 3h** In the Confirm Login Details dialog box, click Next.
- 3i** In the Name the Script window, select a descriptive name (for example, Password Test Application), then click Finish.
- 4** Finish logging in to the Password Test Application by clicking OK at the Log In dialog box. The following window indicates a successful login.



If the login isn't successful, you probably entered your customary password instead of "novell." Right-click the SecureLogin icon on the system tray, click Manage Logins, click Applications, select the PasswordTest application, delete it, click OK to save the changes, then start over.

- 5** Verify that the script is working.

In the Password Test Application dialog box, click File > Log in. SecureLogin quickly types your stored username and password, clicks OK, logs you in, and again displays the Login Successful window.

## Viewing the Wizard's Script

When you used the Add Application Wizard to create a login for Password Test, SecureLogin created a script for the application. You can view the script.

- 1** Right-click the SecureLogin icon on the system tray, click Manage Logins, then click Applications.
- 2** Click PasswordTest.exe > Edit > Script.

While you were using the Add Application Wizard earlier, SecureLogin created and saved this script for you. When you click File > Log in (in the Password Test Application), this script automatically logs you in to the Password Test Application. To gain experience with basic SecureLogin script commands, you will replace this script with one that you create.

## Experimenting with a Script

### Creating a Password Policy

Create a password policy named PwdTestPolicy.

- 1** Right-click the SecureLogin icon on the system tray, then click Manage Logins.
- 2** Click Password Policies > New.
- 3** Type PwdTestPolicy in the New Password Policy text box, then click OK.

The example script you will be working with in the Creating Your Own Script section requires a password policy called PwdTestPolicy.

- 4** Click PwdTestPolicy > Edit.
- 5** Click Minimum Password Length > Edit.
- 6** Type 6 in the Value text box, then click OK.

So that you can use "novell" as a password, the policy must require a minimum of 6 characters but no complex rules.

- 7** Click OK to save the policy, then click OK to save the data.

For additional information on creating a password policy, see [“Creating or Editing a Password Policy”](#) in the *Nsure SecureLogin 3.51.2 Administration Guide*.

## Creating Your Own Script

During this exercise, you will create a script for the Password Test Application.

- 1** Delete PasswordTest.exe from the applications enabled for single sign-on in SecureLogin.

**1a** Right-click the SecureLogin icon on the system tray, then select Manage Logins.

**1b** Click Applications, then select PasswordText.exe.

**1c** Click Delete, click Yes, then click OK to save the changes to SecureLogin.

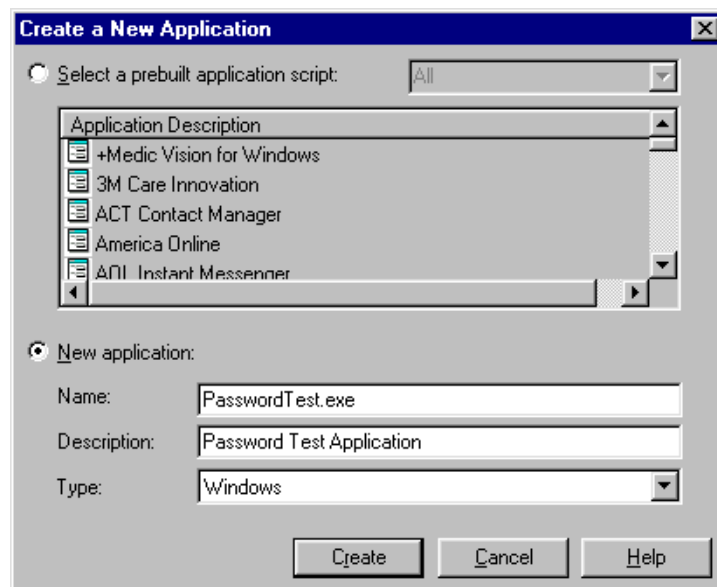
- 2** Create a new PasswordTest.exe application.

You will enable this application for single sign-on by creating a script.

**2a** Right-click the SecureLogin icon on the system tray, then select Manage Logins.

**2b** Click Applications > New.

**2c** In the Create a New Application dialog box, select New Application.



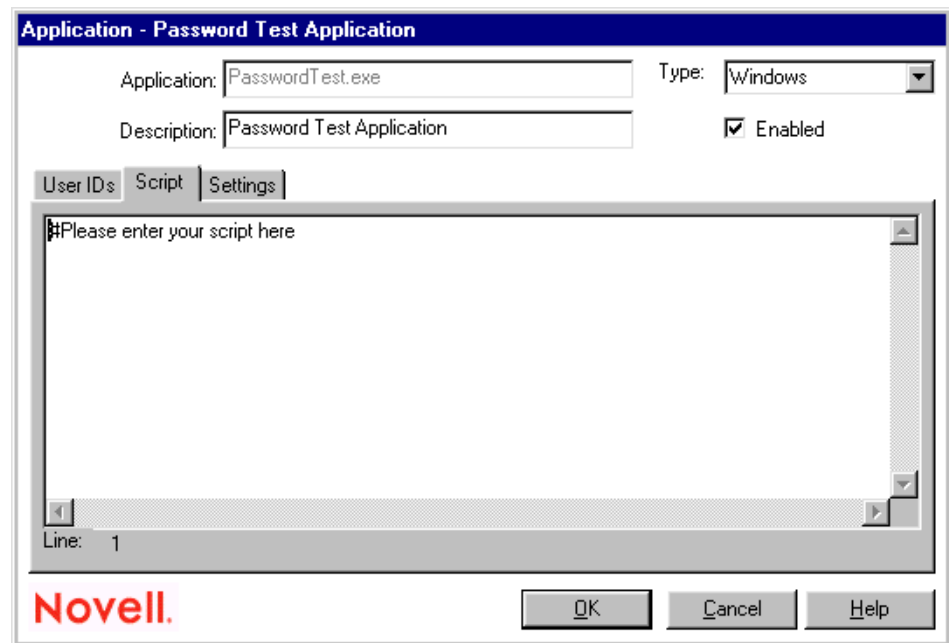
**2d** In the Name edit box, type PasswordTest.exe.

**2e** In the Description edit box, type Password Test Application.

**2f** Leave the default (Windows) as the Type, then click Create.

### 3 Click Script.

As the following figure illustrates, you are ready to type a script.



### 4 Type the Password Policy section.

Replace “#Please enter your script here” with the following lines:

```
# Set the password policy
RestrictVariable $Password PwdTestPolicy
```

**RestrictVariable:** This command restricts the \$Password variable according to the restrictions you set in PwdTestPolicy.

For more information on this command, see [“RestrictVariable” on page 79](#).

### 5 Type the Dialog/EndDialog block.

```
# ===== BeginSection: Login Window =====
Dialog
    Class "#32770"
    Title "Log In"
EndDialog
```

**Dialog/EndDialog:** This block defines a Windows dialog box (a dialog box that pops up on the screen). When the dialog box appears, SecureLogin detects this dialog box based on the information found within the dialog block.

When passwordtest.exe runs, SecureLogin watches for dialog boxes that appear and matches the information defined between the Dialog/EndDialog commands.

Beginning script writers commonly ask how much information is required in the Dialog/EndDialog block. The block must have enough information for the block to be unique. Otherwise, the script runs when other dialog boxes owned by the same executable with the same information appear.

When SecureLogin detects that all the information between Dialog/EndDialog is contained in the dialog box on the screen (for example, an application login box, change password box, or

failed logon box), SecureLogin runs the script commands until it sees the next dialog statement or the end of the script, whichever is applicable.

The order doesn't matter in Windows scripts. SecureLogin watches for all dialog boxes while the executable is running. For troubleshooting purposes, you'll most likely want to use a logical order.

For more information on this command, see [“Dialog / EndDialog” on page 48](#).

**Title:** The Title command in the script identifies the title of the window that SecureLogin needs to enter information into. In this case, the title is "Log In".

For information on this command, see [“Title” on page 97](#).

**Class:** The class of the Login window is #32770.

For information on this command, see [“Class” on page 41](#).

## 6 Type information for the login window.

```
SetPrompt "Username:"
Type $Username #1001
SetPrompt "Password:"
Type $Password #1002
SetPrompt "Other:"
Type $Other #1003
Click #1
SetPrompt "Type missing information. SecureLogin remembers what you type
and automatically logs you in. IT Helpdesk x4546."

# ==== EndSection: Login Window ====
```

**TIP:** When entering long strings after SetPrompt, type the text continuously on one line. It doesn't wrap. If you press Enter to break the line, the SecureLogin script parser will report an error at Step 6.

**SetPrompt:** This command customizes the window that the user sees when the user has no credentials stored. When the user first runs a newly single-sign-on-enabled application, SecureLogin prompts the user for login credentials, stores those credentials, and remembers them for future login attempts.

For information on this command, see [“SetPrompt” on page 89](#).

**Type:** The Username field is Control ID #1001, and the Password field is Control ID #1002. The script types the stored \$Username variable into Control ID #1001 and types the stored \$Password variable into Control ID #1002.

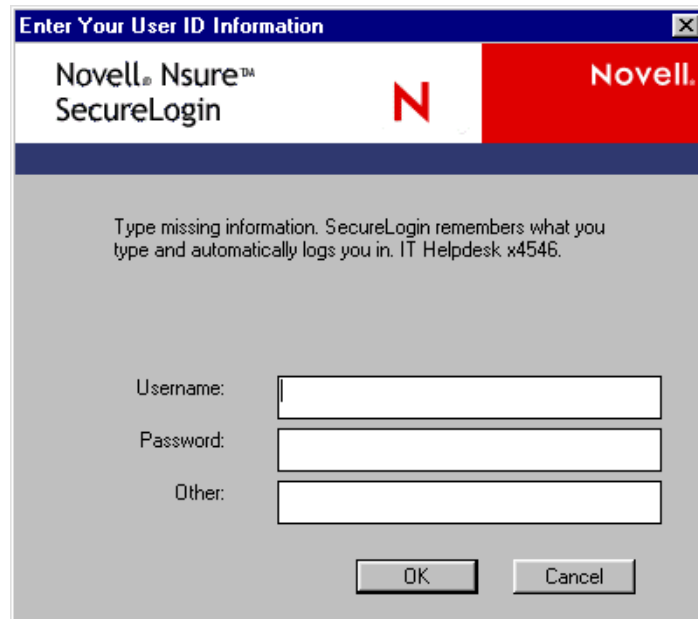
For information on this command, see [“Type” on page 98](#).

**Click: The OK button is Control ID #1.** The Click command sends a click instruction to this Control ID.

For information on this command, see [“Click” on page 44](#).

## 7 Close and save the script by clicking OK twice.

## 8 View the results of the script by clicking File > Log In in the Password Test Application.



This window appears because you haven't yet entered a username, password, or text for the Other field. SecureLogin will store the information and then automatically enter it as the new script calls for it.

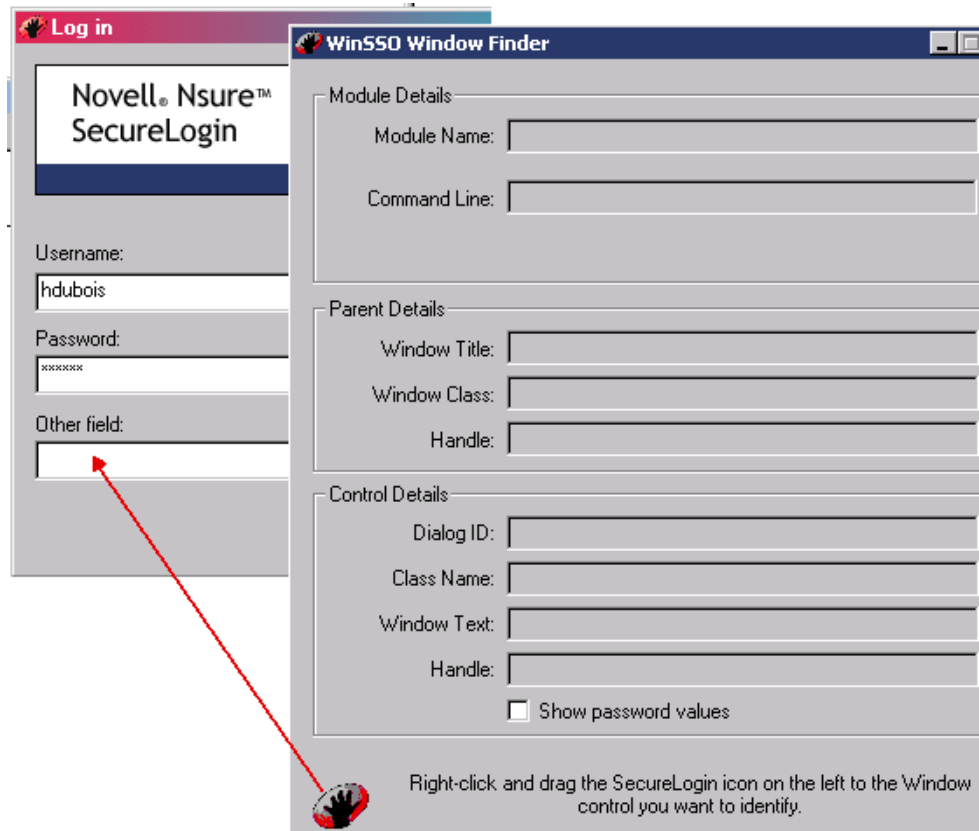
If the script contains an error, the Enter SecureLogin Variables dialog box doesn't appear. Instead, SecureLogin displays an error message, specifying the script line that contains the error. Return to the script and resolve the problem.

Before you enter a domain name and complete the login, use Window Finder to identify the Control ID of the Domain edit box.

## Experiment: Using Window Finder

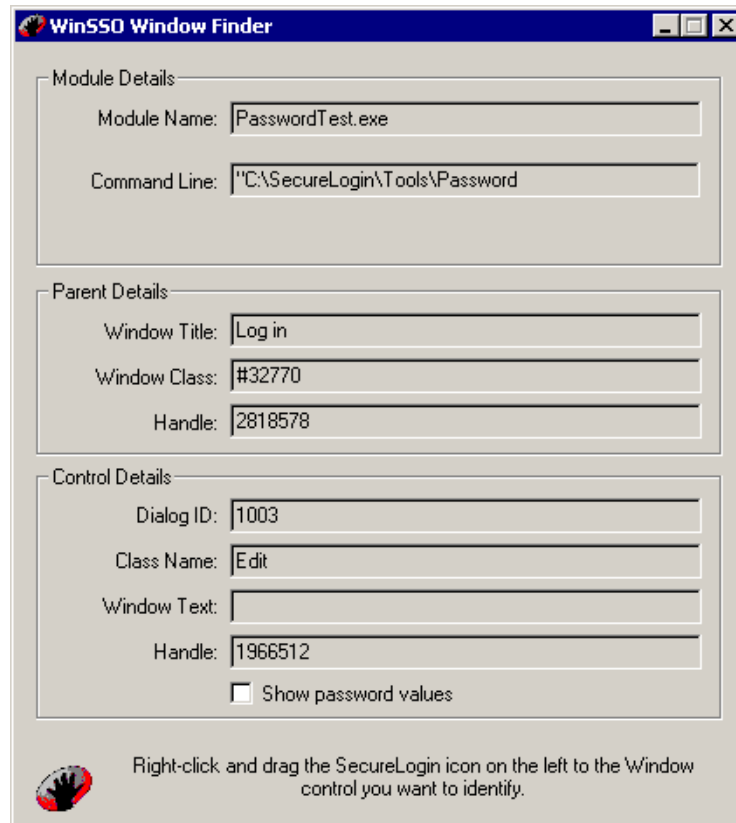
Verify that the Other Field in the Login window is control ID 1003, which appears as #1003 in the script that you created.

- 1 Click Start > Programs > Novell SecureLogin > Window Finder.
- 2 Right-click the hand icon in Window Finder and drag it to the Other Field's edit box.



As the following figure illustrates, Window Finder displays the control ID (1003). After discovering a control ID number, you can type it in a script.





## Adding a MessageBox

The MessageBox command helps you troubleshoot scripts. You can pinpoint problem lines in your script.

- 1 Add a MessageBox command after the # ==== EndSection: Login ==== line.

```
MessageBox "Completed the Login section. Ready for the Change Password section."
```

- 2 Save and close the script by clicking OK twice.
- 3 Observe the feedback by logging in again to the Password Test Application.

If the message box doesn't appear, close and restart SecureLogin, then click File > Log In in the Password Test Application.

- 4 Exit the MessageBox feedback window by clicking OK.

Also close the Login Successful window.

For additional information on this command, see [“MessageBox” on page 65](#).

## Changing Passwords

- 1 Add a comment and the Dialog/EndDialog block for the Change Password section.

```
# ==== BeginSection: Change Password ====
# The Change-Password Dialog Box
Dialog
    Title "Change Password"
```

```
Class "#32770"  
EndDialog
```

**2** Add a backup and ChangePassword block.

```
# Back up password, fill in the Old Username and Password, then start the  
change password routine.  
Set ?PwdBackup $Password  
Type $Username #1015  
Type $Password #1004  
ChangePassword ?NewPwd "Enter a new password for Password Test."  
Type ?NewPwd #1005  
Type ?NewPwd #1006  
Click #1
```

**Set:** For information on this command, see [“Set” on page 82](#).

**ChangePassword:** For information on this command, see [“ChangePassword” on page 40](#).

**3** Add a message block.

```
# Change Password Successful message  
Dialog  
  Title "Change Successful"  
  Class "#32770"  
  Ctrl #65535 "You have changed the password successfully."  
EndDialog
```

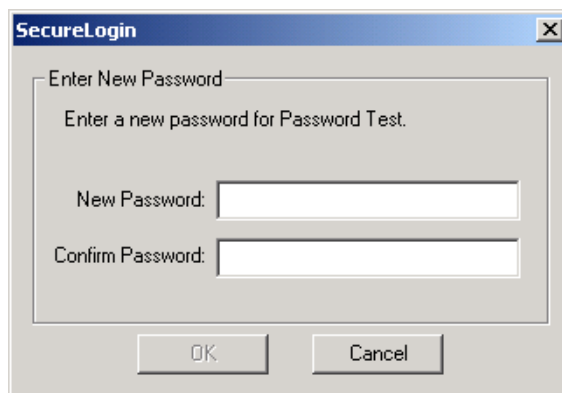
**4** Add the remainder of the script.

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

**5** Save and close the script by clicking OK twice.

**6** Click File > Change Password.

The following figure illustrates the Enter New Password dialog box that you specified in the script:



Move the Enter New Password window to one side of the screen so that you can observe behavior in the Change Password window.

**7** Type and confirm the new password, then click OK.

Provide a password that meets the criteria specified in PwdTestPolicy.

# 7

## Keystrokes and Functions

This section provides information on the following:

- ♦ “Sending Special Keystrokes” on page 115
- ♦ “@ Commands Used with Emulators” on page 116

### Sending Special Keystrokes

SecureLogin can send special keyboard keystrokes to Windows and Internet applications that emulate the user’s keyboard entry. These special commands include the ability to select Menu items and send Alt and other keyboard combinations.

To select a menu item within an application, you could use the following sequence:

Desired Result	Sequence
Select a file	Type \Alt+F
Select tools	Type T
Select Change Password	Type C

The following table illustrates keyboard sequences that you can use:

Desired Result	Sequence
Select a given option, where x is any key	Type \Alt+x Type \Ctr+x Type \Shift+x
Send the Backspace key	Type \B
Send the Delete key	Type \D
Send the End key	Type \E
Send the Home key	Type \H
Send the Enter key	Type \N
Send the Print Screen key	Type \P
Send the Space key	Type \S

Desired Result	Sequence
Send the Tab key	Type \T
Send the Shift-Tab keys	Type \-T
Send the Space bar	Type \ 32 (The   keystroke is the pipe character.)
Send the End key	Type \ 35
Send the Home key	Type \ 36
Send the Left-arrow key	Type \ 37
Send the Up-arrow key	Type \ 38
Send the Right-arrow key	Type \ 39
Send the Down-arrow key	Type \ 40

## @ Commands Used with Emulators

The following table lists the @ commands that you can use in the SecureLogin script Type. These commands perform specific emulator and mainframe functions. For example, you can send an Enter key, Tab key, or cursor, or issue a mainframe emulator print screen or reset function.

The Type Command	Meaning	The Type Command	Meaning
@B	Left Tab	@A@C	Test
@C	Clear	@A@D	Word Delete
@D	Delete	@A@E	Field Exit
@E	Enter	@A@F	Erase Input
@F	Erase EOF	@A@H	System Request
@H	Help	@A@I	Insert Toggle
@I	Insert	@A@J	Cursor Select
@J	Jump (Set Focus)	@A@L	Cursor Left Fast
@L	Cursor Left	@A@Q	Attention
@N	New Line	@A@R	Device Cancel (Cancels Print Presentation Space)
@O	Space	@A@T	Print Presentation Space
@P	Print	@A@U	Cursor Up Fast
@R	Reset	@A@V	Cursor Down Fast
@T	Right Tab	@A@Z	Cursor Right Fast

<b>The Type Command</b>	<b>Meaning</b>	<b>The Type Command</b>	<b>Meaning</b>
@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 Colors
@5	PF5/F5	@A@j	White
@6	PF6/F6	@A@t	Print (Personal Computer)
@7	PF7/F7	@A@y	Forward Word Tab
@8	PF8/F8	@A@z	Backward Word Tab
@9	PF9/F9	@A@-	Field -
@a	PF10/F10	@A@<	Record Backspace
@b	PF11/F11	@A@+	Field +
@c	PF12/F12	@S@x	Dup
@d	PF13	@S@E	Print Presentation Space or Host
@e	PF14	@S@y	Field Mark
@f	PF15	@X@c	Split Vertical Bar ( )
@g	PF16	@X@7	Forward Character
@h	PF17	@X@6	Display Attribute
@i	PF18	@X@5	Generate SO/SI
@j	PF19	@X@1	Display SO/SI
@k	PF20	@M@0	VT Numeric Pad 0
@l	PF21	@M@1	VT Numeric Pad 1
@m	PF22	@M@2	VT Numeric Pad 2
@n	PF23	@m@3	VT Numeric Pad 3
@o	PF24	@M@4	VT Numeric Pad 4

<b>The Type Command</b>	<b>Meaning</b>	<b>The Type Command</b>	<b>Meaning</b>
@q	End	@M@5	VT Numeric Pad 5
@s	ScrLk (No action)	@M@6	VT Numeric Pad 6
@t	Num Lock (No action)	@M@7	VT Numeric Pad 7
@u	Page Up	@M@8	VT Numeric Pad 8
@v	Page Down	@M@9	VT Numeric Pad 9
@x	PA1	@M@-	VT Numeric Pad
@y	PA2	@M@,	VT Numeric Pad
@z	PA3	@M@.	VT Numeric Pad
@M@h	VT Hold Screen	@M@e	VT Numeric Pad Enter
@M@N	Control Code SO	@M@f	VT Edit Find
@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	Control Code S1
@M@D	Control Code EOT	@M@Q	Control Code DC1
@M@C	Control Code ETX	@M@P	Control Code DLE
@M@B	Control Code STX	@M@A	Control Code SOH

# 8

## Troubleshooting Scripts

This section provides information on the following;

- ◆ “Logging In to Web Sites” on page 119
- ◆ “Deriving Application Names from Strings” on page 120
- ◆ “Using the -NoEdit Switch” on page 121
- ◆ “Using Microsoft JVM and Internet Explorer” on page 121

### Logging In to Web Sites

**What’s the best way to log in to Web sites?**

**Answer:** Because SecureLogin recognizes a login panel on a Web page, the easiest method to create scripts for Web sites is to use the pop-up wizard. The second option is to run the wizard manually.

If for some reason you need to examine or modify scripts, you can use the following scripts to enable most HTML Web sites to use SecureLogin. Script One works for more than 95% of HTML Web pages.

#### Script One

```
Type $Username  
Type $Password password
```

The `password` flag always follows the variable that contains the password.

If the first eight letters of a variable are `password`, the password is masked. If the first eight letters of a variable are not `Password`, the entry is displayed normally, unless the Web page masks the entry with asterisks.

The following table illustrates uses of the `$password` variable:

Command	Variable	Result
Type	<code>\$password password</code>	Enters the value of the variable <code>\$password</code> and displays asterisks because the first eight letters of the variable are <code>password</code> .
Type	<code>\$juanspassword password</code>	Enters the value of the variable <code>\$juanspassword</code> , but not as asterisks, unless the Web page masks the entry with asterisks.
Type	<code>\$password4juan password</code>	Enters the value of the variable <code>\$password</code> and displays asterisks because the first eight letters of the variable are <code>password</code> .

## Script Two

```
Type $Username #1
Type $Password #2
Click #1
```

This script is also successful for Web sites. The parameter #1 instructs SecureLogin to enter the value of the variable \$password into the first (from top to bottom) entry field on the page.

**TIP:** If a Web page uses frames, “top to bottom” might not be obvious. In this case, try different numbers until one works.

The parameter #2 instructs SecureLogin to enter the value of the variable \$password into the second entry field on the page.

Using the #1 parameter with the click command instructs SecureLogin which button on the page to click.

The script submits automatically. If a problem occurs, use the following commands:

- ◆ Type \N  
This option presses Enter.
- ◆ Type \n *control position* (for example, Type \N #1)  
This option presses Enter for the specified button for field number. You can also try changing the #1 to #2, #3, and so on to make sure that SecureLogin presses the correct button.
- ◆ Click *control position* (for example, Click #2)
- ◆ “Submit”  
This option forces a submit.

## Deriving Application Names from Strings

### Why do I get error -217 when logging in to a Web site?

**Answer:** The application name is derived from text strings in the login screen (for example, VERDE CENTRAL VMP or Clarify LODGE lodge).

When loaded into a temporary variable, these values work as expected in SetPlat statements and If-Exists statements (for example, SetPlat ?Clarify). However, if the literal value is used in an If-Exists statement in a Web script, error -217 occurs.

The following script shows the problem:

```
If-Exists $Username(Clarify LODGE lodge)
MessageBox a
Else
MessageBox b
EndIf
```

If you modify the first line as follows, the script works. You won't receive an error.

```
If-Exists "$Username(Clarify LODGE lodge)"
```

**IMPORTANT:** Use quotation marks around the string that follows If-Exists.



## Using the -NoEdit Switch

In the SecureLogin 3.51 release, the -NoEdit switch required you to use NoEdit.

However, for the SecureLogin 3.51.2 release, use -NoEdit (with a hyphen).

## Using Microsoft JVM and Internet Explorer

If you are running Microsoft JVM and Internet Explorer, place a Class or Ctrl command (or both) at the beginning of each Dialog/EndDialog statement for any Java application scripts and the iexplore.exe script.

For example, type

```
Dialog
  Class #32770
  Title "Login"
  Ctrl #65535 "Enter Login information"
EndDialog
```

Failure to follow this procedure when using the Microsoft JVM instead of the SUN JVM will result in the following symptoms:

After SecureLogin has logged into a Java-based application or Web page, SecureLogin no longer interacts with any Win32-based applications. SecureLogin appears to stop working because the Microsoft JVM is not replying to the calls that SecureLogin makes to gather title information. Therefore, SecureLogin waits for the JVM to reply. Because the JVM doesn't reply, SecureLogin continues to wait, and will wait endlessly.

Closing the Java Web page or application causes SecureLogin to stop waiting and start responding to Win32-based applications again. In a future update, SecureLogin will timeout on the request if it receives no response in a specified time.



# A

## Quick-Reference Chart

This section provides a quick-reference chart of commands used in SecureLogin scripts. The chart lists the following:

- ♦ The platform that the command is used with (Startup scripts, Terminal Launcher, Web, Windows, or Java)
- ♦ The type of command (action, dialog specifier, flow control, or variable manipulator)
- ♦ The SecureLogin version that the command is used with (All, 2.5 and later, 3.0, or 3.0.4, 3.5, )

Command	Use With	Type	SecureLogin Version
AAVerify	SS, TL, Web, Win	Action	All
Add	SS, TL, Web, Win	Variable manipulator	3.0
Attribute	Advanced Web Script	Specifier	3.5
BeginSplashScreen	TL	Action	3.0.4
Break	SS, TL, Web, Win	Action	2.5
Call	SS, TL, Web, Win	Flow control	2.5
ChangePassword	SS, TL, Web, Win	Action	All
Class	SS, Win	Dialog specifier	All
ClearPlat	SS, TL, Web, Win	Action	3.51.2
Click	Java, Web, Win	Action	All
ConvertTime	SS, TL, Web, Win	Variable manipulator	3.0.4
Ctrl	Java, SS, Win	Dialog specifier	All
Delay	All	Action	All
Dialog/EndDialog	Java, Win	Dialog specifier	All
DisplayVariables	All	Action	All
Divide	SS, TL, Web, Win	Variable manipulator	3.0
DumpPage	Advanced Web Script	Action	3.5
EndScript	SS, TL, Web, Win	Action	All
Event	Win	Dialog specifier	3.5

Command	Use With	Type	SecureLogin Version
GetCheckBoxState	Advanced Web Script	Action	3.5
GetCommandLine	SS, Win	Action	3.0.4
GetEnv	All	Action	3.5
GetIni	Java, TL, Web, Win	Action	3.5
GetReg	All	Action	3.5
GetSessionName	Terminal Emulator	Action	3.5
GetText	TL, Web	Action	3.0
GetURL	Web	Action	3.0
GoToURL	Web	Action	2.5
If/Else/EndIF	SS, TL, Web, Win	Flow control	All
Include	All	Flow control	3.0
Increment/Decrement	All	Variable manipulator	All
KillApp	All	Action	All
Local	All	Variable manipulator	3.0
MessageBox	SS, TL, Web, Win	Action	All
Multiply	All	Variable manipulator	3.0
OnException	All	Flow control	3.0.4
Parent/EndParent	Win	Dialog specifier	All
PickListAdd	All	Action	All
PickListDisplay	SS, TL, Web, Win	Action	All
PositionCharacter	Password Policy Scripts	Action	All
ReadText	TL, Win	Action	All
RegSplit	All	Action	All
ReloadPlat	SS, TL, Web, Win	Action	3.51.2
Repeat/EndRepeat	All	Flow control	All
RestrictVariable	All	Action	All
Run	SS, TL, Web, Win	Action	All
SelectListBoxItem	Advanced Web Script	Action	3.5
SendKey	TL	Action	All
Set	All	Variable manipulator	All

<b>Command</b>	<b>Use With</b>	<b>Type</b>	<b>SecureLogin Version</b>
SetCheckBox	Advanced Web Script	Action	3.5
SetCursor	TL (HLLAPI, some DDE)	Action	All
SetFocus	Java, Web, Win	Action	All
SetPlat	All	Action	All
SetPrompt	All	Action	All
StrCat	All	Variable manipulator	All
StrLength	All	Variable manipulator	3.0.4
StrLower	All	Variable manipulator	3.0.4
StrUpper	All	Variable manipulator	3.0.4
Sub/EndSub	SS, TL, Web, Win	Flow control	2.5
Submit	Web	Action	3.0
Subtract	SS, TL, Web, Win	Variable manipulator	3.0
Tag/EndTag	Advanced Web Script	Action	3.5
Title	Java, Win	Dialog specifier	All
Type	Java, TL, Web, Win	Action	All
WaitForFocus	Win	Flow control	All
WaitForText	TL	Flow control	All



# B

## FAQs on Scripting

This section provides information on the following:

- ♦ “One Script, Two Sets of Credentials” on page 127
- ♦ “Cache All Passwords, Prompt for Each Application” on page 127

### One Script, Two Sets of Credentials

**Question:** How do you use the same script to use one set of credentials for login to a Web URL, and then when it redirects to another page without changing the URL address, have it present another set of credentials?

**Answer:** Use the `setplat` command as part of the script. When you recognize where you are on the correct Web page, issue a `setplat` command to set the user ID and Password to the field that you require. Then use the `Type` command to enter that detail.

### Cache All Passwords, Prompt for Each Application

**Question:** My company would like SecureLogin to cache all passwords in Novell® eDirectory™, but we would like to have SecureLogin prompt for either the Passphrase or the eDirectory password each time someone opens an application.

In essence this would give users the impression that they have one password for all their applications. Is this feasible, or would it cause too much overhead in authenticating to the network?

**Answer:** Try one of the following:

- ♦ Store and compare a variable.
  1. Instead of inputting the application password, prompt the user for input.
  2. Store the input as a new variable (for example, `$Inputpw`).
  3. Compare this variable with `?Syspassword`.
  4. If the variable is true, input `$Password` into the correct field.
- ♦ Use `AAVERIFY` to prompt for the eDirectory password.





# C

## Trapping SNMP Alerts

SecureLogin is able to produce SNMP alerts so that network monitoring software can trap them. A simple script command sends the alerts upon any event you want.

**NOTE:** For SNMP support to work, you might have to copy the `libsnp.dll` file to the `Windows\System32` directory.

### Producing an Alert

To produce an SNMP alert, place the following command in the script where you want the alert to be created:

```
Run c:\Progra~1\Novell\Secure~1\slnsnp.exe Community Name Host IP Address Text
```

- ♦ *Community Name* is the case-sensitive community name that this computer will send trap messages to.
- ♦ *Host IP Address* is the IP address of the SNMP host.
- ♦ *Text* is the text to be displayed as the message at the host.

### Example Script

```
Dialog
  Class #32770
  Title "Incorrect Password"
EndDialog

Run C:\Progra~1\Novell\Secure~1\Slnsnp.exe SNMPCommunity1 192.168.156.23
"PSL - Incorrect password in finance system."
MessageBox "You have entered an incorrect password. The administrator has been
notified. Restart the application and try again."
KillApp "PasswordText.exe"
```



# D

## Keyboard Functions and Codes

The following table is a reference on keyboard functions from within Windows. You can use these functions with the Type command.

Function	Decimal	Comment/Information
Left mouse button	1	
Right mouse button	2	
Ctrl-Break	3	
Middle mouse button	4	
X1 mouse button	5	
X2 mouse button	6	
Backspace	8	
Tab	8	
Clear	12	5 on the keypad
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	

---

<b>Function</b>	<b>Decimal</b>	<b>Comment/Information</b>
Right-arrow	39	
Down-arrow	40	
Select	41	
Execute	43	
Print Screen	44	
Insert	45	
Delete	46	
Help key	47	
0	48	
1	49	
2	50	
3	51	
4	52	
5	53	
6	54	
7	55	
8	56	
9	57	
A	65	
B	66	
C	67	
D	68	
E	69	
F	70	
G	71	
H	72	
I	73	
J	74	
K	75	
L	76	

---

Function	Decimal	Comment/Information
M	77	
N	78	
O	79	
P	80	
Q	81	
R	82	
S	83	
T	84	
U	85	
V	86	
W	87	
X	88	
Y	89	
Z	90	
Left Windows key	91	
Right Windows key	92	
Application key	93	
Sleep key	94	
Keypad 0	96	
Keypad 1	97	
Keypad 2	98	
Keypad 3	99	
Keypad 4	100	
Keypad 5	101	
Keypad 6	102	
Keypad 7	103	
Keypad 8	104	
Keypad 9	105	
Keypad Asterisk (*)	106	
Keypad plus sign (+)	107	

<b>Function</b>	<b>Decimal</b>	<b>Comment/Information</b>
Keypad Separator	108	
Keypad minus sign (-)	109	
Keypad period (.)	110	
Keypad backward slash (/)	111	
F1 key	112	
F2 key	113	
F3 key	114	
F4 key	115	
F5 key	116	
F6 key	117	
F7 key	118	
F8 key	119	
F9 key	120	
F10 key	121	
F11 key	122	
F12 key	123	
F13 key	124	
F14 key	125	
F15 key	126	
F16 key	127	
F17 key	128	
F18 key	129	
F19 key	130	
F20 key	131	
F21 key	132	
F22 key	133	
F23 key	134	
F24 key	135	
Num Lock	144	
Scroll Lock	145	

<b>Function</b>	<b>Decimal</b>	<b>Comment/Information</b>
Left Shift	160	
Right Shift	161	
Left Ctrl	162	
Right Ctrl	162	
Left menu	164	
Right menu	<b>165</b>	
Browser Back key	166	Windows 2000 or later required
Browser Forward key	167	Windows 2000 or later required
Browser Refresh key	168	Windows 2000 or later required
Browser Stop key	169	Windows 2000 or later required
Browser Search key	170	Windows 2000 or later required
Browser Favorites key	171	Windows 2000 or later required
Browser Start and Home key	172	Windows 2000 or later required
Volume Mute key	173	Windows 2000 or later required
Volume Down key	174	Windows 2000 or later required
Volume Up key	175	Windows 2000 or later required
CD Next Track key	176	Windows 2000 or later required
CD Previous Track key	177	Windows 2000 or later required
CD Stop Media key	178	Windows 2000 or later required
CD Play/Pause key	179	Windows 2000 or later required
Launch Mail key	180	Windows 2000 or later required
Media Select key	181	Windows 2000 or later required
Start Application 1 key	182	Windows 2000 or later required
Start Application 2 key	183	Windows 2000 or later required
;	186	Semicolon/colon
=	187	Equals/plus
,	188	Comma/less than
-	189	Minus/underscore
.	190	Period/greater than
/	191	Slash/question mark

Function	Decimal	Comment/Information
'	192	Single open quote/tilde
[	219	Left square/brace
\	220	Back slash/pipe
]	221	Right square/brace
'	222	Single close quote/double quote
Play key	250	
Zoom key	251	



# E

## Event Specifiers

The following table illustrates Windows application events that you can monitor by using the Event command.

Event	Event	Event
BM_CLICK	EM_GETTHUMB	SBM_GETSCROLLINFO
BM_GETCHECK	EM_GETWORDBREAKPROC	SBM_SETSCROLLINFO
BM_GETIMAGE	EM_LIMITTEXT	STM_GETICON
BM_GETSTATE	EM_LINEFROMCHAR	STM_GETIMAGE
BM_SETCHECK	EM_LINEINDEX	STM_MSGMAX
BM_SETIMAGE	EM_LINELENGTH	STM_SETICON
BM_SETSTATE	EM_LINESCROLL	STM_SETIMAGE
BM_SETSTYLE	EM_POSFROMCHAR	WM_ACTIVATE
EM_CANUNDO	EM_REPLACESEL	WM_ACTIVATEAPP
EM_CHARFROMPOS	EM_SCROLL	WM_AFXFIRST
EM_EMPTYUNDOBUFFER	EM_SCROLLCARET	WM_AFXLAST
EM_FMTLINES	EM_SETHANDLE	WM_APP
EM_GETFIRSTVISIBLELINE	EM_SETIMESTATUS	WM_ASKCBFORMATNAME
EM_GETHANDLE	EM_SETMARGINS	WM_CANCELJOURNAL
EM_GETIMESTATUS	EM_SETMODIFY	WM_CANCELMODE
EM_GETLIMITTEXT	EM_SETPASSWORDCHAR	WM_CAPTURECHANGED
EM_GETLINE	EM_SETREADONLY	WM_CHANGECHAIN
EM_GETLINECOUNT	EM_SETRECT	WM_CHAR
EM_GETMARGINS	EM_SETRECTNP	WM_CHARTOITEM
EM_GETMODIFY	EM_SETSEL	WM_CHILDACTIVATE
EM_GETPASSWORDCHAR	EM_SETTABSTOPS	WM_CLEAR
EM_GETRECT	EM_SETWORDBREAKPROC	WM_CLOSE
EM_GETSEL	EM_UNDO	WM_COMMAND

Event	Event	Event
WM_COMPACTING	WM_EXITMENULOOP	WM_LBUTTONUP
WM_COMPAREITEM	WM_EXITSIZEMOVE	WM_MBUTTONDOWNBLCLK
WM_CONTEXTMENU	WM_FONTCHANGE	WM_MBUTTONDOWN
WM_COPY	WM_GETDLGCODE	WM_MBUTTONUP
WM_COPYDATA	WM_GETFONT	WM_MDIACTIVATE
WM_CREATE	WM_GETHOTKEY	WM_MDICASCADE
WM_CTLCOLORBTN	WM_GETICON	WM_MDICREATE
WM_CTLCOLORDLG	WM_GETMINMAXINFO	WM_MDIDESTROY
WM_CTLCOLOREDIT	WM_GETOBJECT	WM_MDIGETACTIVE
WM_CTLCOLORLISTBOX	WM_GETTEXT	WM_MDIICONARRANGE
WM_CTLCOLORMSGBOX	WM_GETTEXTLENGTH	WM_MDIMAXIMIZE
WM_CTLCOLORSCROLLBAR	WM_HANDHELDFIRST	WM_MDIRESTORE
WM_CTLCOLORSTATIC	WM_HANDHELDLAST	WM_MDISETMENU
WM_CUT	WM_HELP	WM_MDITILE
WM_DEADCHAR	WM_HOTKEY	WM_MEASUREITEM
WM_DELETEITEM	WM_HSCROLL	WM_MENUCHAR
WM_DESTROY	WM_HSCROLLCLIPBOARD	WM_MENUCOMMAND
WM_DESTROYCLIPBOARD	WM_ICONERASEBKGD	WM_MENUDRAG
WM_DEVICECHANGE	WM_INITDIALOG	WM_MENUGETOBJECT
WM_DEVMODECHANGE	WM_INITMENU	WM_MENURBUTTONUP
WM_DISPLAYCHANGE	WM_INITMENUPOPUP	WM_MENUSELECT
WM_DRAWCLIPBOARD	WM_INPUTLANGCHANGE	WM_MOVE
WM_DRAWITEM	WM_INPUTLANGCHANGEREQUEST	WM_MOVING
WM_DROPFILES	WM_KEYDOWN	WM_NCACTIVATE
WM_ENABLE	WM_KEYFIRST	WM_NCCALCSIZE
WM_ENDSESSION	WM_KEYLAST	WM_NCCREATE
WM_ENTERIDLE	WM_KEYUP	WM_NCDESTROY
WM_ENTERMENULOOP	WM_KILLFOCUS	WM_NCHITTEST
WM_ENTERSIZEMOVE	WM_LBUTTONDOWNBLCLK	WM_NCLBUTTONDOWNBLCLK
WM_ERASEBKGD	WM_LBUTTONDOWN	WM_NCLBUTTONDOWN

Event	Event	Event
WM_NCLBUTTONUP	WM_PRINT	WM_SPOOLERSTATUS
WM_NCMBUTTONDBLCLK	WM_PRINTCLIENT	WM_STYLECHANGED
WM_NCMBUTTONDOWN	WM_QUERYDRAGICON	WM_STYLECHANGING
WM_NCMBUTTONUP	WM_QUERYENDSESSION	WM_SYNCPAINT
WM_NCMOUSEMOVE	WM_QUERYNEWPALETTE	WM_SYSCHAR
WM_NCPAINT	WM_QUERYOPEN	WM_SYSCOLORCHANGE
WM_NCRBUTTONDBLCLK	WM_QUEUESYNC	WM_SYSCOMMAND
WM_NCRBUTTONDOWN	WM_QUIT	WM_SYSDEADCHAR
WM_NCRBUTTONUP	WM_RBUTTONDBLCLK	WM_SYSKEYDOWN
WM_NEXTDLGCTL	WM_RBUTTONDOWN	WM_SYSKEYUP
WM_NEXTMENU	WM_RBUTTONUP	WM_TCARD
WM_NOTIFY	WM_RENDERALLFORMATS	WM_TIMECHANGE
WM_NOTIFYFORMAT	WM_RENDERFORMAT	WM_TIMER
WM_NULL	WM_SETCURSOR	WM_UNDO
WM_PAINT	WM_SETFOCUS	WM_UNINITMENUPOPUP
WM_PAINTCLIPBOARD	WM_SETFONT	WM_USER
WM_PAINTICON	WM_SETHOTKEY	WM_USERCHANGED
WM_PALETTECHANGED	WM_SETICON	WM_VKEYTOITEM
WM_PALETTEISCHANGING	WM_SETREDRAW	WM_VSCROLL
WM_PARENTNOTIFY	WM_SETTEXT	WM_VSCROLLCLIPBOARD
WM_PASTE	WM_SHOWWINDOW	WM_WINDOWPOSCHANGED
WM_PENWINFIRST	WM_SIZE	WM_WINDOWPOSCHANGING
WM_PENWINLAST	WM_SIZECLIPBOARD	WM_WININICHANGE
WM_POWER	WM_SIZING	



# F

## Error Codes

This section contains error codes for Terminal Launcher.

For a full list of SecureLogin error codes, see “[Error Codes](#)” in the *Nsure SecureLogin 3.51.2 Administration Guide*.

### Error Codes with Tips

#### **-102 BROKER\_NO\_SUCH\_ENTRY**

**Possible Cause:** You tried to load a script or variable that doesn't exist. For example, you set up Terminal Launcher to run from a shortcut or to run a particular script, but the script doesn't exist.

**Action:** Check that the name of the script is actually defined in SecureLogin. Verify that the name is the same as specified in the script editor.

#### **-220 BROKER\_HLLAPI\_FUNCTION\_NOT\_FOUND**

**Possible Cause:** You used an incorrect function when you defined the emulator. In the Terminal Launcher configuration, you specified a HLLAPI.DLL and the name of the function in that DLL. The name of the function cannot be found in the DLL.

**Action:** Using the *Nsure SecureLogin 3.51.2 Guide for Terminal Emulation* on the [Novell Documentation Web site \(http://www.novell.com/documentation\)](http://www.novell.com/documentation), check the configuration for the emulator. Make sure that you typed the HLLAPI function correctly.

#### **-222 BROKER\_HLLAPI\_DLL\_LOAD\_FAILED**

**Possible Cause:** Terminal Launcher was unable to load the HLLAPI.DLL that you specified.

**Action:** Make sure that the path and file that you entered for the DLL are correct.

**Possible Cause:** The HLLAPI.DLL for that emulator is looking for other DLL files that don't exist or haven't been installed for that emulator.

**Action:** You have probably chosen the wrong .DLL file or have specified the wrong HLLAPI function (for example, HLLAPI or WinHLLAPI). Find the correct .dll and function. Check the vendor's documentation for information about that emulator.

To find control IDs, see “[Finding Control IDs and Offsets of an Emulator](#)” in the *Nsure SecureLogin 3.51.2 Administration Guide*.

#### **-224 BROKER\_ERROR\_DURING\_WINHLLAPICLEANUP**

**Possible Cause:** Terminal Launcher has called the WinHLLAPI cleanup function for a WinHLLAPI emulator.

Action: Check the vendor's documentation for information about that emulator.

#### **-225 BROKER\_CANNOT\_FIND\_WINHLLAPISTARTUP\_FUNCTION\_IN\_DLL**

Possible Cause: In the Terminal Launcher configuration, you incorrectly specified that the emulator is a WinHLLAPI emulator.

Action: Using the *Nsure SecureLogin 3.51.2 Guide for Terminal Emulation* on the [Novell Documentation Web site \(http://www.novell.com/documentation\)](http://www.novell.com/documentation), check the configuration for the emulator. Specify the correct emulator type.

#### **-226 BROKER\_ERROR\_DURING\_WINHLLAPISTARTUP**

Action: Check the vendor's documentation for information about that emulator.

#### **-227 BROKER\_CANNOT\_FIND\_WINHLLAPICLEANUP\_FUNCTION\_IN\_DLL**

Possible Cause: In the Terminal Launcher configuration, you incorrectly specified that the emulator is a WinHLLAPI emulator.

Action: Using the *Nsure SecureLogin 3.51.2 Guide for Terminal Emulation* on the [Novell Documentation Web site \(http://www.novell.com/documentation\)](http://www.novell.com/documentation), check the configuration for the emulator. Specify the correct emulator type.

#### **-264 BROKER\_DDE\_CONNECT\_FAILED**

Possible Cause: Terminal Launcher couldn't connect to a specified DDE emulator.

Action: Make sure that the emulator launched correctly and the emulator's DDE support is turned on.

#### **-273 BROKER\_MSTELNET\_OPERATION\_NOT\_SUPPORTED**

Possible Cause: The generic emulator can't support a particular operation (for example, SetCursor).

Action: For generic emulators, don't use the command.

#### **-279 BROKER\_EMULATOR\_LAUNCH\_FAILED**

Possible Cause: In Terminal Launcher, you can configure the path to the executable that will run. However, the specified executable is unable to run.

Action: Make sure that the path to the emulator is correct.

#### **-280 BROKER\_UNABLE\_TO\_CREATE\_EMULATOR**

Possible Cause: You have specified an invalid terminal type in TLAUNCH.INI (or the Terminal Launcher configuration).

Action: Specify the correct terminal type.

#### **-281 BROKER\_INVALID\_CHARACTER\_FOUND\_IN\_PASTE\_ID\_LIST**

Possible Cause: A comma doesn't separate decimal numbers for input and output control IDs.

Action: For generic emulators, you must specify a set of input and output control IDs. Use a comma to separate decimal numbers.

### **-282 BROKER\_INVALID\_CHARACTER\_FOUND\_IN\_COPY\_ID\_LIST**

Possible Cause: A comma doesn't separate decimal numbers for copy IDs

Action: For generic emulators, you must specify a set of copy control IDs. Use a comma to separate decimal numbers.

### **-283 BROKER\_UNABLE\_TO\_READ\_TLAUNCH\_INI**

Possible Cause: SecureLogin is unable to read the tlaunch.ini file because the file has been deleted.

Action: Create a blank tlaunch.ini file.

Action: Create a default tlaunch.ini file by reinstalling SecureLogin.

### **-284 BROKER\_NO\_TERMINAL\_TYPE\_DEFINED**

Possible Cause: The tlaunch.ini file contains an error. The terminal type for the emulator has not been defined.

Action: Using Terminal Launcher, specify a terminal type for the emulator.

### **-290 BROKER\_FILE\_LOAD\_FAILED**

Possible Cause: You don't have enough rights to convert an earlier tlaunch.ini file to a later format, read an earlier tlaunch.ini file, or create a new tlaunch.ini file.

Action: The network administrator must assign necessary rights.

### **-349 BROKER\_UNABLE\_TO\_FIND\_SESSION\_FILE**

Possible Cause: Terminal Launcher couldn't find a session file for an emulator.

Action: Configure Terminal Launcher to have the correct path to the file for the emulator session.

### **-356 BROKER\_INVALID\_CHARACTER\_FOUND\_IN\_STARTUP\_ID\_LIST**

Possible Cause: For generic emulators, you specify the startup control ID. A comma must separate a list of numbers. You have used a character other than a comma.

Action: Remove unacceptable characters.

### **-373 BROKER\_HLLAPI\_CONNECT\_FAILED**

Possible Cause: Terminal Launcher couldn't find the function name and was therefore unable to connect to the emulator. The function name is probably wrong.

Action: Make sure that the emulator has HLLAPI enabled.

### **-380 BROKER\_HLLAPI\_NOT\_CONNECTED\_TO\_PS**

Possible Cause: You haven't configured your emulator for an HLLAPI session. Terminal Launcher tried to use a HLLAPI function. However, the HLLAPI DLL is not connected to the emulator presentation space.

Action: Make sure that Terminal Launcher is set up correctly with the emulator.

### **-381 BROKER\_HLLAPI\_SPECIFYING\_PARAMETERS\_ERROR**

Possible Cause: Incorrect parameters were given to a command that uses a HLLAPI function.

Action: Contact Novell Technical Services.

### **-382 BROKER\_HLLAPI\_INVALID\_PS\_POSITION**

Possible Cause: Terminal Launcher was able to initialize the emulator but was unable to read the contents of the screen. An attempt was made to move the cursor or read text from an invalid (out of bounds) position on the emulator presentation space.

Action: Correct the positioning parameter in the script.

### **-383 BROKER\_HLLAPI\_SYSTEM\_ERROR**

Possible Cause: Terminal Launcher is not configured correctly for the emulator.

Action: Make sure that Terminal Launcher is set up correctly with the emulator and that the emulator correctly supports HLLAPI.

### **-384 BROKER\_HLLAPI\_PS\_BUSY\_ERROR**

Possible Cause: A HLLAPI function is being called while the emulator presentation space is unavailable.

Action: Make sure that the emulator is not being used by other HLLAPI applications.

### **-385 BROKER\_HLLAPI\_INPUT\_REJECTED**

Possible Cause: The emulator rejected an attempt to input data into the emulator presentation space.

Action: Make sure that the emulator presentation space is not locked.

### **-386 BROKER\_HLLAPI\_ERROR\_QUERYING\_SESSIONS**

Possible Cause: SecureLogin is unable to query available HLLAPI sessions.

Action: Make sure that Terminal Launcher is set up correctly with the emulator.