Novell Nsure™ SecureLogin

3.51

www.novell.com

SCRIPTING GUIDE

May 13, 2004

Novell®

Legal Notices

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

www.novell.com

Nsure SecureLogin 3.51 Scripting Guide
May 13, 2004

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

Third-Party Trademarks

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

Contents

About This Guide

This guide is for network administrators. The following sections provide information on scripting:

Additional Documentation

For documentation on understanding, managing, and troubleshooting SecureLogin, see the Nsure SecureLogin 3.51 Administration Guide.

For documentation on installing SecureLogin, see the Nsure SecureLogin 3.51 Installation Guide.

For documentation on terminals services, see the Nsure SecureLogin 3.51 Terminal Services Guide.

For documentation on terminal emulators, see the Nsure SecureLogin 3.51 Configuration Guide for Terminal Emulation.

For documentation on Novell® SecretStore®, see the Novell SecretStore 3.3.0 Administration Guide.

Documentation Updates

For the most recent version of this and other SecureLogin guides, see SecureLogin (http://www.novell.com/documentation) at the Novell Documentation Web page.

Documentation Conventions

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

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

1 Introduction to Scripting

This section provides information on the following:

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® technology to provide additional benefits:
  - Provides an additional level of security.
  - Enables you to share secrets with other applications (for example, Novell iChain® 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, If/Else/EndIf) you can skip, repeat or jump to certain parts of the script.

With the use of Dialog Specifier commands (for example, Call), you can skip, repeat, or jump to 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 23.

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.

The SecureLogin language has 52 different commands. Many of these, such as Repeat and Dialog, have one or two additional commands (for example, EndRepeat or EndDialog) that are used to close them. See Chapter 5, "SecureLogin Commands," on page 37.

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 eDirectory.

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 37.

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:

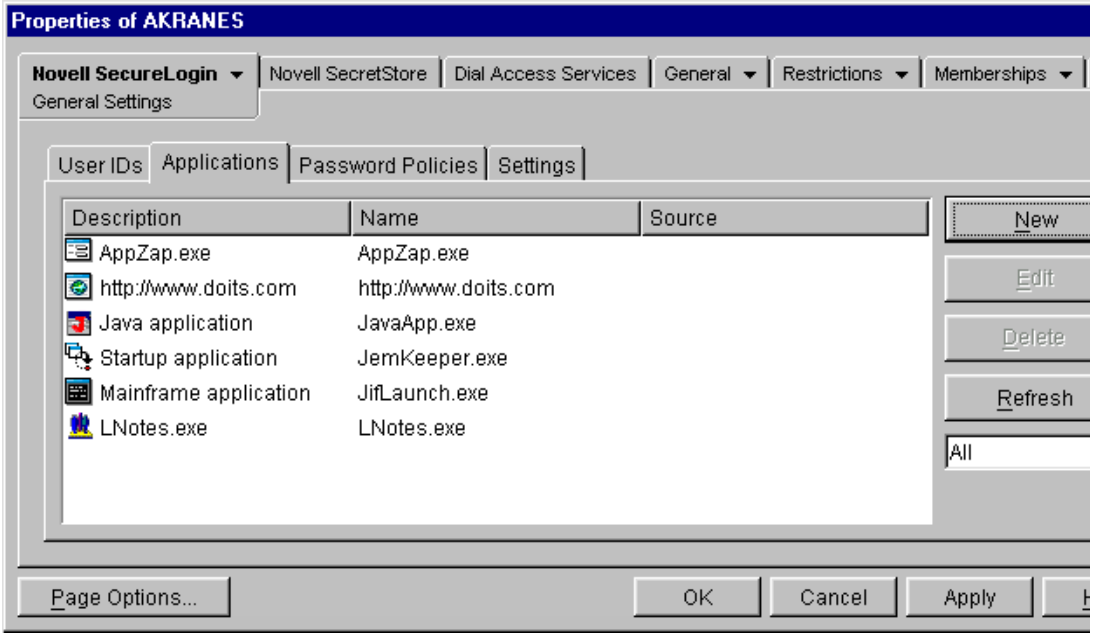
```
Window "login"
Delay 30
#SecureLogin ignores this line and the next two lines
#while executing the script.
```

```
#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.

Script Type	Description
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 red C in the upper left corner of the icon.

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.

Scripts for Predefined Applications

SecureLogin provides native script support for many popular applications so that you don't have to configure them manually.

Application	Application	Application
+Medic Vision for Windows	MeetingMaker	pcANYWHERE* 8.0
3M* Care Innovation	Microsoft* Front Page	PeopleSoft*
ACT Contact Manager	Microsoft Internet Gaming Zone (lobby.exe)	plusw33.exe
America Online*	Microsoft Internet Gaming Zone (zone.exe)	QuickBooks* Pro
AOL Instant Messenger*	Microsoft Money 98/99	Quicken*
Bloomberg	Microsoft Networking Client	Remedy* ARUSER
Clarify	MMIS	Remedy Notifier
Corporate Time	MMIS (NTVDM)	RiskMaster v3.6
Entrust* Client	Mobile UP v4.5	SAP* /R3 Login
Entrust Server	MS SQL	Siebel* Customer Tracking
Eudora* Email	MSN Messenger	Soft Front
GoldMine	MYOB Premier	STARS
GoldMine 5.5	Netscape*	Sunrise* Clinical Manager
ICQ	Novell BorderManager® VPN Client	Visual SourceSafe* Login
Informix* Connect for Win32*	Novell GroupWise® Client	Windows 9x Dialup Networking
Internet Explorer	Novell GroupWise Notify Client	Windows 9.*x* Login

Application	Application	Application
JPilot applet	Onebox email	Windows NT* Logon
Lotus* Notes*	Oracle* Generic Login	Yahoo!* Messenger
Lotus Organizer* 4 + 5	Oracle Financials	Zainetbar
Meditech Remote Workstation		

2 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 Capitals

Use capitalization where applicable.

Use This	Instead of This
`MessageBox "Some text" -YesNo ?Result`	`Messagebox "Some text" -yesno ?result`

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

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

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

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

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 38.

Use This	Instead of This

`#Written by M. Kurz June 7, 2002`
`#Modified by C. Bertrand July 3, 2003`
 	
`#Login Dialog Box`
`Dialog`
`  Title "Login"`
`  Class #32770`
`EndDialog`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`EndDialog`

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.

3 Using Symbols and Variables

This section contains information on the following:

Symbols Used in Scripts

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

The Pound Symbol (#)

Use the pound or hash symbol (#) to define a line of text as a comment field, so that you can annotate 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 37.

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 23.

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 24 and "Internal Variables" on page 25.

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 23.

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 25.

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 37.

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 114 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 37.

Understanding Script Variables

This section contains information on the following:

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”).

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 against a script in the Directory under the User object. The values are encrypted so that only the user can access them.

In general, variables are 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 85.

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 50 and “ChangePassword” on page 43.

Example: Stored Variables in Use

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

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

Runtime Variables

In general, use runtime variables to store calculations, process data, or date information. 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 63.

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.

SecureLogin has one preset runtime variable: ?CurrTime. This variable is constantly updated to be equal to the number of seconds that have passed from January 1970 until the present time. This variable can also be used with the ConvertTime command to convert it to read the current time and date. This feature is generally used to enforce password changes every *x* days.

Example: A Runtime Variable in Use

```
Dialog
  Title "ERROR"
  Class #32770
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 the user's details (for example, the Windows version, which Internet browser is in use, or the password that was used to access the Directory) from the Directory and the workstation. These details are accessed by using internal variables. Internal variables have set names, which vary with the Directory in use and are prefixed with a percentage sign (%). See "Internal Variables" on page 25.

The following are examples of when these variables are used:

- When it is necessary to use a different script for different browser types
- When it is necessary to use a different script for different versions of Windows
- When the password for an application is synchronized with the Directory password

Passticket Variables

Passticket variables are used to generate one-time passwords based on a DES key and offset, similar to how VASCO tokens work. They are used differently than the other variable types. 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, if you want 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 $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.

Internal 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 and can be used in the same manner within any application definition.

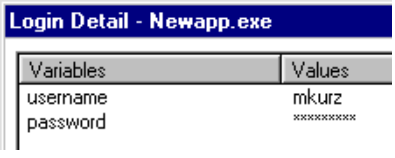
The following table lists the internal variables that SecureLogin supports.

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, 3050109 represents v3.5.1.9, in W.XX.YY.ZZ format.
?*BrowserType* (system)	Contains either Internet Explorer or Netscape and indicates which browser the script is running in.
?*SysUser* (system)	The name of the user that was last used in the GINA or Windows 9*x* login dialog box. This variable is available only when the SecureLogin login extension is installed.
?*SysPassword* (system)	The password that matches the username presented in the GINA dialog box. This variable is available only when the SecureLogin login extension is installed.
?*SysContext* (system)	Lists the Novell® eDirectory™ user context as entered in the GINA or Windows 9*x* login dialog box. This variable is available only when the SecureLogin login extension is installed.
?*SysTree* (system)	The NDS® or eDirectory tree name that the user entered. This variable is available only if the SecureLogin login extension is installed.
?*SysServer* (system)	The name of the server that was entered in the login GINA or Windows 9*x* login dialog box. This variable is available only when the SecureLogin 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.

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.



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.

4 Working with Scripts

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

Managing 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.

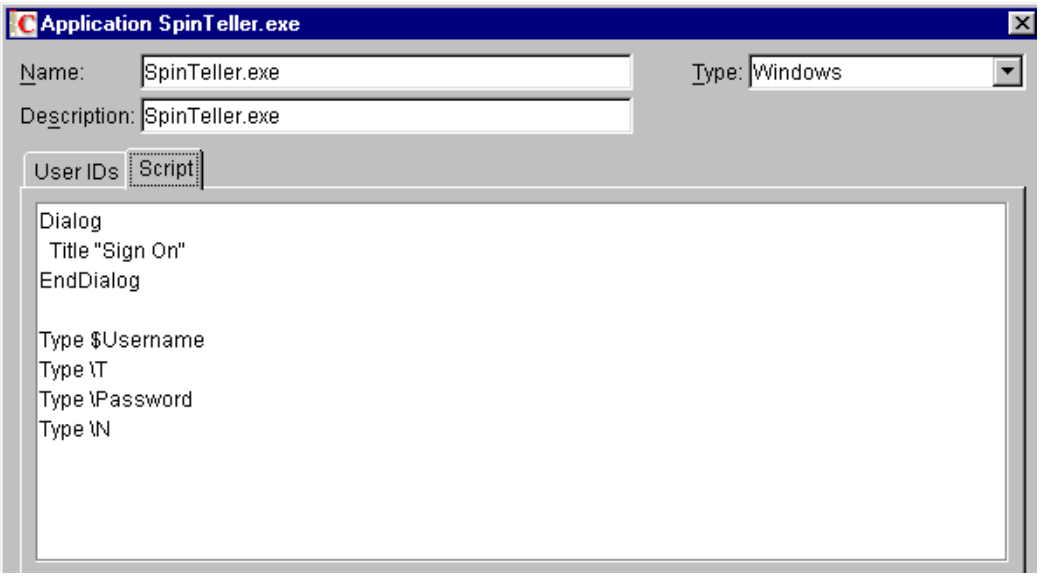
SecureLogin has a scripting wizard as well as a host of prebuilt scripts. These features enable you to easily enable a broad range of applications for single sign-on.

Manage scripts for applications by using ConsoleOne®, 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 37.

To experiment with a sample script and a test application, see Chapter 6, “Practicing Your Scripting Skills,” on page 103.

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 are normal scripts that are assigned to a Container object instead of to a User object. Corporate scripts differ from other scripts in two ways:

- The application is added at an Organization or Organizational Unit object instead of a User object.
- You use ConsoleOne, MMC, or SecureLogin Manager to add the application.

The differences are the location and inheritance only.

Because they are automatically rolled out to all User objects held in the Container object, corporate scripts simplify implementing and administering SecureLogin single sign-on. By using this method, you don’t have to configure applications for each individual user in your organization. All users read and use the same scripts.

Windows Application, Web, Startup, and Terminal Launcher scripts can all be implemented as corporate scripts.

Creating a Corporate Script: MMC or ConsoleOne

1 Log in as Admin or an Admin equivalent.

2 Navigate to the Container object where you want to create the corporate script.

3 Right-click the Container object, then click Properties.

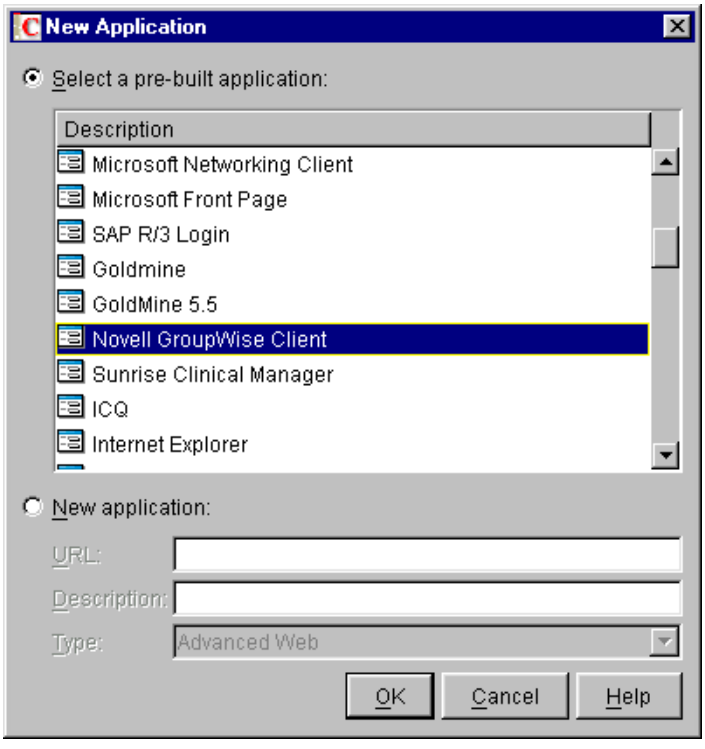
4 Click Novell SecureLogin > Applications > New.

To use a prebuilt script, go to Step 5.

To create a new script for an application, without using a prebuilt script, go to Step 6.

5 (Optional) Add a prebuilt script to the application list.

5a Click Select a Prebuilt Script, scroll to and select the desired application, then click OK.

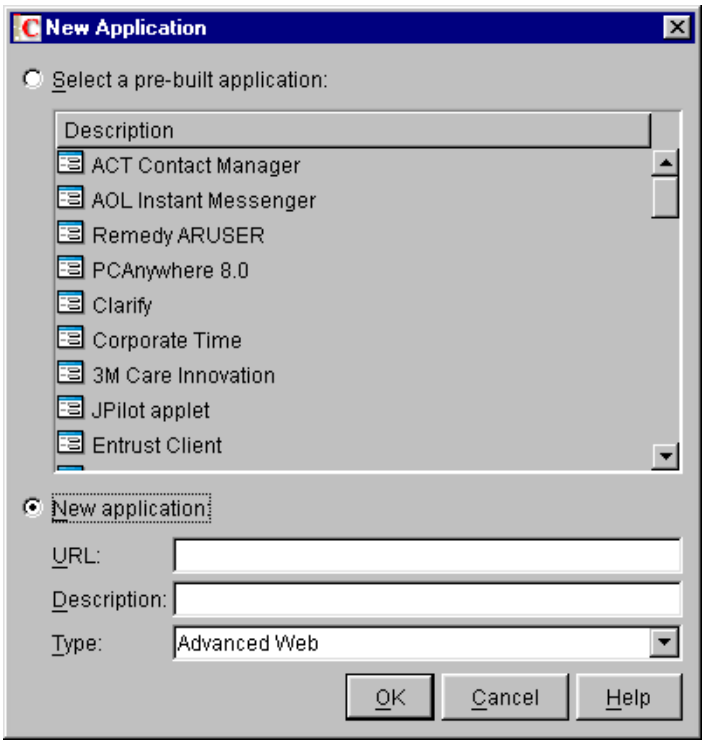


5b At the Applications tab, save the script by clicking Apply or OK.

The next time the selected application is launched, users will be prompted to enter their credentials.Whenever the application is subsequently launched, SecureLogin enters the users' credentials, as though the login process has been eliminated.

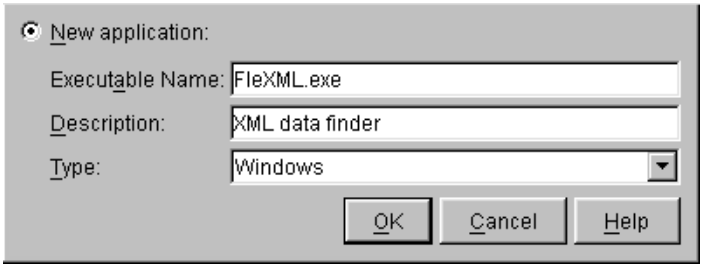
6 (Optional) Add an application that doesn't have a script.

6a From the New Application dialog box, click New Application.



6b Type a name in the first text field.

For a Windows application, type the executable filename. For a Web application, type the URL. This name will display in the Description column on the Applications tab.



7 Select a type (for example, Java, Startup, Windows) from the drop-down list, then click OK.

8 At the Applications tab, save the data by clicking Apply.

9 Click the newly added application, click Edit, then click Script.

10 Add a script.

For hands-on experience with basic scripting, work through the tutorial in Chapter 6, "Practicing Your Scripting Skills," on page 103.

For script commands, with accompanying example scripts and explanations, see Chapter 5, "SecureLogin Commands," on page 37.

Creating a Corporate Script: SecureLogin Manager

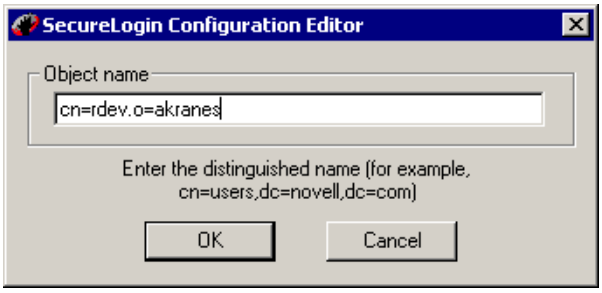
1 Log in to the workstation as Admin or equivalent.

2 Run SecureLogin.

3 Launch SecureLogin Manager.

Run slmanager.exe, found in the \securelogin\client\tools directory.

4 Type the distinguished name of the object where you want to create a corporate script.



You logged in to the workstation as Admin or equivalent, then accessed SecureLogin as that user. SecureLogin Manager uses the rights of the authenticated user to create the corporate script for the context or object that you specify.

For AD and LDAP, use LDAP naming conventions (for example, cn=admin,cd=akranes). For eDirectory, use eDirectory conventions (for example, cn=admin.o=akranes).

5 Click OK.

Exempting a User Object from a Corporate Script

Local scripts take precedence over corporate scripts. Occasionally, you might want a particular user to use a script other than the corporate script. To do this, create a local script for the application at the User object level.

If you have a corporate script for an application, and you have a user who should not have that application single sign-on enabled, create a blank local script for the application at the User object level.

You can also use this procedure to exempt a Container object from corporate scripts inherited from Container objects that are higher in the directory tree.

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.

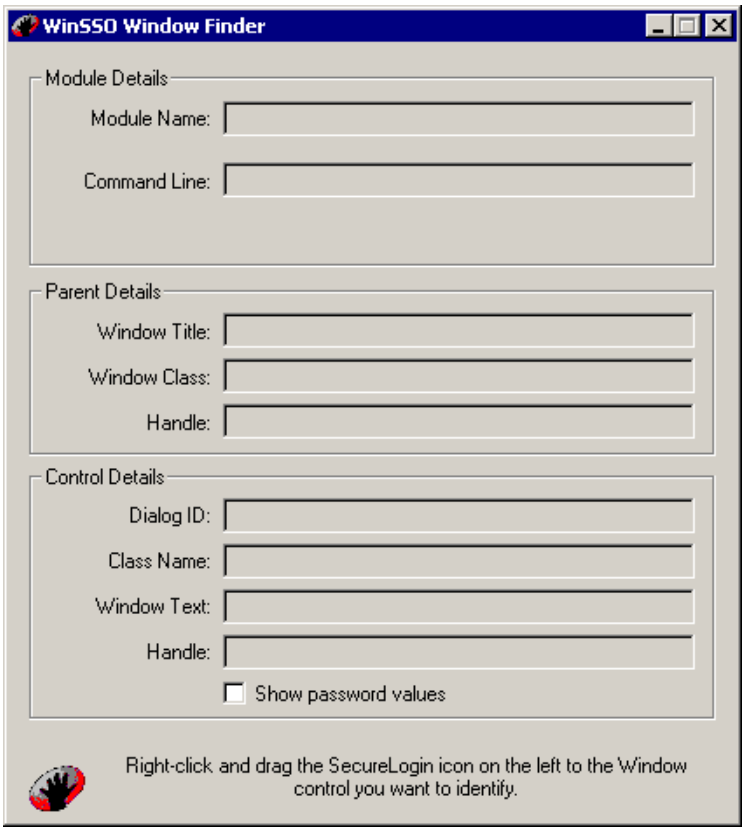
To inspect a control:

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 Chapter 5, "SecureLogin Commands," on page 37.
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.
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.

Using 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 fired 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 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 54.

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
Use with:	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.
SecureLogin Version:	All: You can use the command in all versions. Version number: The version that the command was introduced in.
Type:	Action: Use the command to perform an action, such as the way the Type command types information into an application. Dialog specifier: Use the command to define dialog boxes. For example, see “Parent / EndParent” on page 69 and “Class” on page 44. Flow control: Use the command to direct SecureLogin around the script. For example, see “Repeat / EndRepeat” on page 75 and “EndScript” on page 53. Variable manipulator: Use the command to modify variables. For example, see “Add” on page 39 and “Subtract” on page 94.
Usage:	The command argument / *variable*. Variables, values, text, and other items that you type are italicized in the tables. Optional items that you type are placed in brackets ([ ]).
Arguments:	Argument / *variable*: A brief explanation of the argument or variable.
Description:	An explanation of the command and how it is used.
Syntax Examples:	Examples of the various ways the command can be written in a script.
Example: Script type Script explanation	An example script.

AAVerify

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	All (Arguments were added in version 3.0.)
Type:	Action
Usage:	AAVerify -Method *NMAS sequence* -User *User object* -Tree *Tree name* [?Result]
Arguments:	
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 uses the current user that is authenticated.
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 will receive the result of the AAVerify command. The variable is set to either True for success or False for failure.
Description:	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. 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 fails, the [?Result] is set to False. If NMAS is not installed on the workstation, the script sends an error, or an error is returned via [?Result]. 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 Novell Modular Authentication Services (http://www.novell.com/documentation/lg/nmas21/index.html).
Syntax Examples:	AAVerify AAVerify -Method "Enhanced Password" ?Result AAVerify -Method "Enhanced Password" -User "mkurz" - Tree "Production" ?Result

Item	Description

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

Add

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	3.0
Type:	Variable manipulator
Usage:	Add *Variable1 Variable2* [?Result]
Arguments:	
Variable1	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 *Variable1* without the ?Result argument, *Variable1* must be a SecureLogin variable. Otherwise, *Variable1* can be any numeric value.
Variable2	The second argument, the number added to the first argument in the equation. V*ariable2* can be a SecureLogin variable or a numeric value.
[?Result]	Optional. The sum or result of the equation.
Description:	Adds one number 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.

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

Item	Description

Example Windows Script: 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.	`ReadText #103 ?Number1`
	`ReadText #104 ?Number2`
	`Add ?Number1 ?Number2 ?Result`
	`Type ?Result #1`

Attribute

Item	Description
Use with:	Advanced Web Script
SecureLogin Version:	3.5
Type:	Specifier
Usage:	Attribute *Variable Name Value Name*
Arguments:	
Variable Name	The name of the variable to discover.
Value Name	The value that the above variable must contain for the condition to be true.
Description:	The Attribute specifier works with the Tag/EndTag command and specifies which attributes must exist.

Example: SecureLogin finds the form that has an attribute of “name” with a value of “login.”	`Tag "Form"`
	`  Attribute "Name" "Login"`
	`EndTag`

BeginSplashScreen / EndSplashScreen

Item	Description
Use with:	Terminal Launcher (Generic and Advanced Generic only)
SecureLogin Version:	3.0.4
Type:	Action
Usage:	BeginSplashScreen EndSplashScreen
Arguments:	None

Item	Description
Description:	Displays a Novell splash screen across the whole 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.

Example: Terminal Launcher Script After launching the emulator, SecureLogin waits two seconds for it to connect. The splash screen displays to cover the flashing. A login is detected and a username is entered. The splash screen disappears.	`Delay 2000`
	`BeginSplashScreen`
	`  WaitForText "ogin:"`
	`  Type $Username`
	`EndSplashScreen`
	`Type @E`

Break

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	2.5
Type:	Action
Usage:	Break
Arguments:	None
Description:	Used within the Repeat/EndRepeat commands to break out of a repeat loop.

Example 1: Windows Script 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.	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`EndDialog`
	
	`Repeat`
	`  ReadText #301 "?Text"`
	`  If ?Text Eq "Login"`
	`  Break`
	`  EndIf`
	`Delay 100`
	`EndRepeat`

Item	Description

Example 2: Terminal Script
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 is not successful, the script checks again. Terminal emulators use Repeat loops for error handling and Break to break out of the loop as appropriate.
	`# Initial System Login`
	`WaitForText "ogin:"`
	`Type $Username`
	`Type @E`
	`WaitForText "assword:"`
	`Type $Password`
	`Type @E`
	`Delay 500`
	
	`# Repeat loop for error handling`
	`Repeat`
	
	`#Check to see if password has expired`
	`If -Text "EMS: The password has expired."`
	`ChangePassword #Password`
	`Type $Password`
	`Type @E`
	`Type $Password`
	`Type @e`
	`EndIf`
	
	`#User has an invalid Username and / or Password stored.`
	`If -Text "Login Failed"`
	`DisplayVariables "The username and / 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
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	2.5
Type:	Flow control
Usage:	Call *Subroutine*
Arguments:	
Subroutine	The name of the subroutine to be called. This name must be identical to the name specified in the Sub command.
Description:	Calls and runs a subroutine.

Example:
Terminal Script
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.
	`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 login. IT x453." $Password`
	`  Call Login`
	`EndSub`

ChangePassword

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	All
Type:	Action

Item	Description
Usage:	ChangePassword *Variable* [Text] [Random]
Arguments:	
Variable	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.
Description:	Allows a single variable to be changed. Use this command in scenarios where password expiration is an issue. The *Variable* will be set to the new password. 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. 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 "RestrictVariable" on page 78.
Syntax Examples:	ChangePassword ?NewPassword ChangePassword ?NewPassword "Enter a new password" ChangePassword ?NewPassword Random

Example:
Windows Script
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.
	`# Change Password Dialog Box`
	`Dialog`
	`  Title "Change Password"`
	`  Class #32770`
	`EndDialog`
	
	`Set $PasswordBackup $Password`
	`Type $Password #1015`
	`ChangePassword $Password Random`
	`Type $Password #1005`
	`Type $Password #1006`
	`Click #1`
	
	`# Change Password Failed Dialog Box`
	`Dialog`
	`  Title "Change Password Failed"`
	`  Class #32770`
	`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."`

Class

Item	Description
Use with:	Startup scripts, Windows
SecureLogin Version:	All
Type:	Dialog specifier

Item	Description
Usage:	Class *Window-Class*
Arguments:	
Window-Class	A string specifying the window class that this statement will match.
Description:	When a window is created, it is based on a template known as a window class. The Class command checks to see if the class of the newly created window matches its *Window-Class* argument. If the window matches the *Window-Class* argument, the execution of the script continues to the next line. If the window does not match the *Window-Class* argument, execution continues at the next dialog statement. You can determine the class by using Window Finder. See "Finding Control IDs" on page 33.

Example: Windows Script
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.
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`EndDialog`
	
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

Click

Item	Description
Use with:	Startup scripts, Web, Windows
SecureLogin Version:	All
Type:	Action
Windows Usage:	
Usage 1	Click *#Ctrl-ID* [-Raw] [-Right]
Usage 2	Click *X_Coordinate Y_Coordinate*
Web Usage:	Click *#Number*

Item	Description
Arguments:	
#Ctrl-ID	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.
X_Coordinate	Represents the horizontal coordinate relative to the client area of the application (not the screen).
Y_Coordinate	Represents the vertical coordinate relative to the client area of the application (not the screen).
#Number	The pound or hash symbol followed by the sequential number of the button to be pressed. The number of the button is determined by the Web page layout. The first button (top to bottom, left to right) on the page is number 1, the second button on the page is number 2, and so on. Because of Web page layout and design, the sequential order of the buttons might not be obvious.
number	The sequential number of the button to be pressed. The number of the button is determined by the Web page layout. The first button on the page is number 1, the second button on the page is number 2, and so on. Because of Web page layout and design, the sequential order of the buttons might not be obvious.
Description:	When used with Windows applications, the Click command sends a click command to the specified *#Ctrl-ID*. If the button to be clicked does not have a control ID, use the Type \N command. The -Raw flag causes SecureLogin to bypass the mouse by emulating the mouse and sending a direct click message to the control. Using the -Right flag with the -Raw flag sends a right-click to the control. If the button or control does not respond to the Click command, you can set the -Raw flag. Setting the *#Ctrl-ID* to 0 (zero) sends the Click command to the window that the script is running on. You can also set *X coordinate Y coordinate* coordinates. These coordinates are relative to the client area of the application, not the screen. 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.
Syntax Examples:	Click #1 Click #1 -Raw -Right Click -*X* 12 -*Y* 24

Example 1: Windows Script
The Login dialog is detected, the username and password are entered, and button number 1 (in this case the Login button) is clicked.
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Ctrl #32770`
	`EndDialog`
	
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

Item	Description

Example 2:
Web Script
The username and password are entered, then the Login button is clicked.
	`Type $Username`
	`Type $Password Password`
	`Click #1`

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

ConvertTime

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	3.0.4
Type:	Variable manipulator
Usage:	ConvertTime ?CurrTime *String Time*
Arguments:	
String Time	The output variable.
Description:	Converts ?Currtime system to a string representation of the date and time and stores it in *String Time*.

Example:
Windows Script
Converts the time to a readable format and displays it in a dialog box.
	# Login Dialog Box
	Dialog
	  Title "Login"
	  Class #32770
	End Dialog
	
	ConvertTime ?CurrTime ?Time
	MessageBox ?Time

Ctrl

Item	Description
Use with:	Startup scripts, Windows
SecureLogin Version:	All
Type:	Dialog specifier
Usage:	Ctrl *#Ctrl-ID* [*RegEx*]
Arguments:	
#Ctrl-ID	The ID number of the control to be checked.
[*RegEx*]	The regular expression.
Description:	Determines if a window contains the control expressed in the *#Ctrl-ID* argument. The control ID number is a constant that is established at the time a program is compiled. **NOTE:** Third-party software control ID numbers might not be constant from one version to the next. You can use the Window Finder tool to determine the control ID number. See "Finding Control IDs" on page 33. Using the [*RegEx*] argument adds a further check that allows the script to skip to the next command. If the text on the specified *#Ctrl-ID* does not conform to the [*RegEx*], the script skips to the next dialog statement as though the *#Ctrl-ID* did not exist.
Syntax Examples:	Ctrl #1 Ctrl #1 "OK"

Example:
Windows Script
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 does not match, the script passes on to the next dialog block.
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Ctrl #1 "OK"`
	`  Ctrl #2 "Cancel"`
	`  Ctrl #3 "Help"`
	`EndDialog`
	
	`Type $Username`
	`Type \T`
	`Click #1`

Delay

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	All
Type:	Action
Usage:	Delay *Time Period*
Arguments:	
Time Period	A period of time, expressed in milliseconds (1/1000 of a second), to pause execution of the script.
Description:	Delays the action of the script for the time specified in the *Time Period* argument. The time specified in the *Time Period* argument is noted in milliseconds. For example, Delay 5000 creates a 5-second pause. Use the Delay command to accommodate an introduction screen or some other custom feature. When troubleshooting new scripts, add Delay to the script as a first course of action. To optimize SecureLogin's performance, use the Delay command in all Repeat loops.

Example:
Windows Script
The login box is detected. However, the script waits half a second before acting upon it to ensure that the box is complete.
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`EndDialog`
	
	`Delay 500`
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

Dialog / EndDialog

Item	Description
Use with:	Startup scripts, Windows
SecureLogin Version:	All
Type:	Dialog specifier
Usage:	Dialog EndDialog
Arguments:	None

Item	Description
Description:	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). 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). 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.

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

DisplayVariables

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	All
Type:	Action
Usage:	DisplayVariables [*User Prompt*] [*Variables*]
Arguments:	
[*User Prompt*]	Optional, customized text to be displayed in the Enter SecureLogin Variables dialog box.
[*Variables*]	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
Description:	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. 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. 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. 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. After variables are stored for the user, the prompt is customized when the script is run. You can also customize the text in the prompt by using the SetPrompt command. See "SetPrompt" on page 88. **TIP:** Sometimes you will need to enter "dummy" variables in the script, depending on the placement of the DisplayVariables command. For the command to operate optimally, some $Variables must be listed after the command. The OnException EnterVariablesCancelled command can be used to prevent a user from cancelling the DisplayVariables prompt.
Syntax Examples:	DisplayVariables DisplayVariables "Enter your details" DisplayVariables "Enter a new password" $Password DisplayVariables "Enter your username and password" $Username $Password DisplayVariables "" $Username $Password

Example:
Windows script
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.
	`# Wrong Password Dialog Box`
	`Dialog`
	`  Title "Wrong Password"`
	`  Class #32770`
	`EndDialog`
	
	`DisplayVariables "Enter a new username and password" $Username $Password`
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

Divide

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	3.0
Type:	Variable manipulator
Usage:	Divide *Variable1 Variable2* [?Result]
Arguments:	
Variable1	The dividend. The first argument. The number that will be divided by the second argument. This argument will contain the result if the optional [?Result] argument is not passed in. If you use the *Variable1* argument without the [?Result] argument, *Variable1* must be a SecureLogin variable (either ?Variable1 or $Variable1). Otherwise, *Variable1* can be any numeric value.
Variable2	The divisor. The second argument. The number that the first argument is divided by. The *Variable2* argument can be a SecureLogin variable or a numeric value.
[?Result]	The quotient or result of the equation.
Description:	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.
Syntax Examples	Divide "1" "2" ?Result Divide ?LoginAttempts ?LoginFailures Divide ?LoginAttempts ?LoginFailures ?Result Divide ?LoginAttempts "3" Divide ?LoginAttempts "3" ?Result

Example:
Windows Script
The values of Control IDs 103 and 104 are read into variables. From there they are divided and typed into Control ID 1.
	`ReadText #103 ?Number1`
	`ReadText #104 ?Number2`
	`Divide ?Number1 ?Number2 ?Result`
	`Type ?Result #1`

DumpPage

Item	Description
Use with:	Advanced Web Script
SecureLogin Version:	3.5
Type:	Action

Item	Description
Usage:	DumpPage *Variable*
Arguments:	
Variable	The string variable to receive the page information.
Description:	Provides information about the current Web page. This information can be useful for debugging scripts for a Web page.
Example:	`DumpPage ?dump` `MessageBox ?dump`

DumpScript

Item	Description
Use with:	Advanced Web Script
SecureLogin Version:	3.5
Type:	Action
Usage:	DumpScript *Variable*
Arguments:	
Variable	The string variable to receive the script information.
Description:	Used for debugging a web page. Shows the structure of what SecureLogin has seen as tags.
Example:	`DumpScript ?dump` `MessageBox ?dump`

EndScript

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	All
Type:	Action
Usage:	EndScript
Arguments:	None
Description:	Immediately terminates execution of the script.

Item	Description

Example: Windows Script
The login box is detected. SecureLogin enters the username and password and clicks OK. If the "Incorrect Password" message is detected, SecureLogin tells the user that the password was incorrect and terminates the script.
	`Dialog`
	`  Title "Login Failure"`
	`  Ctrl #1`
	`EndDialog`
	
	`ReadText #65535 ?ErrorMsg`
	`If "Incorrect Password" -In ?ErrorMsg`
	`  MessageBox "You have entered an incorrect password"`
	`  EndScript`
	`EndIf`

Event

Item	Description
Use with:	Win32 application scripts
SecureLogin Version:	3.5
Type:	Dialog specifier
Usage:	Event *Event*
Arguments:	
Event	The application event to monitor. For a list of events that you can specify, see Appendix E, “Event Specifiers,” on page 133.
Description:	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. 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". You can apply the Event specifier only within a Dialog and EndDialog statement block. 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.
Syntax Examples:	Dialog Class "someclass" Event WM_ACTIVATE EndDialog Messagebox "Caught the WM_ACTIVATE message"

GetCheckBoxState

Item	Description
Use with:	Advanced Web Script
SecureLogin Version:	3.5
Type:	Action
Usage:	GetCheckBoxState *Item Number Variable*
Arguments:	
Item Number	The Windows control ID of the check box.
Variable	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.
Description:	Returns the current state of the specified check box.

Example:.	`GetCheckBoxState #25 ?state1`
	`GetCheckBoxState #26 ?state2`
	`Messagebox ?state1`
	`Messagebox ?state2`

GetCommandline

Item	Description
Use with:	Startup scripts, Windows
SecureLogin Version:	3.0.4
Type:	Action
Usage:	GetCommandline *Variable*
Arguments:	
Variable	Defines where the captured command line will be stored.
Description:	Captures the full command line of the program that is loaded and saves it to the specified variable. **TIP:** You can use GetCommandLine to detect and differentiate back-end systems or databases for use with multiple logins in the SAP application.

Item	Description

Example: Windows script 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.	`GetCommandline ?Text`
	`If ?Text Eq "C:\Winnt\Notepad.exe"`
	`  Killapp Notepad.exe`
	`EndIf`

GetSessionName

Item	Description
Use with:	Terminal Emulator
SecureLogin Version:	3.5
Type:	Action
Usage:	GetSessionName *?variable*
Arguments:	None
Description:	Finds the current HLLAPI session name that is being used to connect and returns it to the specified variable.
Example: Windows Script	GetSessionName ?Session_name

GetText

Item	Description
Use with:	Web
SecureLogin Version:	3.0
Type:	Action
Usage:	GetText *Variable*
Arguments:	
Variable	Defines where the captured text will be stored.

Item	Description
Description:	Gets all of the text from the screen and saves it to the specified variable. This command Is rarely used and is generally unnecessary. Use this command in a large Web script that might contain several If-Text statements. Under Netscape, each If-Text statement scans the screen to find the specified text. Each scan of the screen results in the screen flashing. However, 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.

Example:
Web Script
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.
	`GetText ?Text`
	`If "Login" -In ?Text`
	`  Type $Username`
	`  Type $Password Password`
	`EndIf`

GetURL

Item	Description
Use with:	Web
SecureLogin Version:	3.0
Type:	Action
Usage:	GetURL *Variable*
Arguments:	
Variable	Defines where the captured URL will be stored.
Description:	Captures the URL of the site that is loaded and saves it to the specified variable.

Example:
Web Script
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.
	`GetURL ?URL`
	`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`

GotoURL

Item	Description
Use with:	Web
SecureLogin Version:	2.5
Type:	Action
Usage:	GotoURL *URL* [*-frame*]
Arguments:	
URL	The URL that the browser will navigate to.
-frame	Opens the URL in the frame that started the script.
Description:	Makes the browser navigate to the specified *URL*. By default, the command opens the new Web page in the main window, rather than the frame that started the script. When you use the -frame option on a framed Web page, the URL redirect occurs only in the current frame rather than in the parent window.

Example:
Web Script
SecureLogin detects an incorrect password message, displays a message box informing the user, then browses to the Novell Web site.
	`If -text "Incorrect Password"`
	`  MessageBox "You have entered an incorrect password"`
	`  GotoURL "http://www.novell.com"`
	`EndIf`

If / Else / EndIf

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	All
Type:	Flow control

Usage 1:	If *Value1 Gt/Lt/Eq* V*alue2*
	#Do This
	Else
	#Do This
	Endif

Usage 2:	If -Text [-frame] *Text*
	#Do This
	Else
	#Do This
	EndIf

Item	Description

Usage 3:	If -Exists *Variable*
	  #Do This
	Else
	  #Do This
	EndIf

Usage 4:	If *Text_to_find* -In *Text_to_search_through*
	  #Do This
	Else
	  #Do This
	EndIf

Arguments:	
Gt/Lt/Eq/-In	Operators. They compare the value on the left side of the operator to the value on the right side of the operator. ◆ If *Value1* Eq *Value2* Assesses whether two values are equal. ◆ If *Value1* Gt *Value2* Assesses whether one value is greater than another value. ◆ If *Value1* Lt *Value2* Assesses whether one value is less than another value. ◆ If *Text_to_find* -In *Text_to_search_through* Searches for text from within specified text.
-Text	If -Text [frame] *Text*. A shortcut to evaluate text within windows. You can use the optional -frame switch within framed Web pages to restrict searching for the text in the current frame only.
-Exists	If -Exists *Variable*. Assesses whether the text specified in the variable is present.
Description:	Establishes a block to be executed if the *Operator* is found to be true. The Else command works inside an If block. This command is executed if the operator in the If block is false. The EndIf command terminates the If block.
Syntax Examples:	If ?Value1 Gt ?Value2 If -Text "Login” If -Exists $RunBefore If "Login" -In ?Text

Item	Description

Example 1: Web Script
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.
	`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`

Example 2: Windows Script
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.
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`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"`

Example 3: Web Script
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.
	`GetText ?WebText`
	`If "Login" -In ?WebText`
	`  Type $Username`
	`  Type $Password Password`
	`Else`
	`  EndScript`
	`EndIf`

Item	Description

Example 4: Startup Script 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.	If -Exists $LoadedBefore
	  EndScript
	Else
	  MessageBox -YesNo ?Result "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?"
	  Set $LoadedBefore "Yes"
	  If ?Result Eq "Yes"
	  GoToURL "http://www.company.com/SecureLoginDetails.htm"
	  EndIf
	EndIf

Include

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	3.0
Type:	Flow control
Usage:	Include *Platform-Name*
Arguments:	
Platform-Name	The name of the script to be included.
Description:	Allows commonly-used application script code to be shared by multiple applications. The script identified by *Platform-Name* is included at execution time into the calling application script. The application type selected for the script to be included should be compatible with the calling script's application type.

Example: Windows Script SecureLogin detects the login dialog, executes the notepad.exe script, then enters the user's credentials.	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`EndDialog`
	
	`Include Notepad.exe`
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

Increment / Decrement

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	All
Type:	Variable manipulator
Usage:	Increment *Variable* Decrement *Variable*
Arguments:	
Variable	The name of the variable to increase or decrease in value.
Description:	Counts 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. This instruction can be particularly useful in the following situations: ◆ When you configure an application whose login panel is similar to other windows within the application. ◆ To easily control the number of attempts a user can have to access an application.
Syntax Examples:	Increment ?RunCount Decrement ?RunCount

Example:
Windows Script
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.
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`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"`

KillApp

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	All
Type:	Action
Usage:	KillApp *Process-Name*
Arguments:	
Process-Name	The name of the process that will be terminated.
Description:	Terminates an application.

Example:
Windows Script
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.
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`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"`

Local

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	3.0
Type:	Variable manipulator
Usage:	Local *?Variable*

Item	Description
Arguments:	
?Variable	The runtime variable that will be declared as local.
Description:	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 ?Variable. 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. 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. 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.

Example:
Windows Script
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.
	`# Invalid Login Message`
	`Dialog`
	`  Title "Login Failure"`
	`  Class #32770`
	`EndDialog`
	
	`Local ?RunCount`
	`Increment ?RunCount`
	`If ?RunCount Gt "5"`
	`  MessageBox "Closing Application"`
	`  KillApp "PasswordText.exe"`
	`EndIf`
	`Type $Username`
	`Type $Password`

MessageBox

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

Item	Description
Arguments:	
[-YesNo]	Allows the user to select either Yes or No within the message box rather than being limited only to an OK button.
[-YesNoCancel]	Allows the user to select either Yes, No, or Cancel when a message box is presented.
?Variable	Required with the -YesNo or -YesNoCancel flag to store the result of the user action.
[-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 is not 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.
Data	The text to be displayed to the user.
Description:	Displays a dialog box that contains the text specified in the *Data* 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: `MessageBox "The User "$Username" has just been logged into the system″` You can set the -YesNo flag when calling a MessageBox. If the -YesNo flag is set, the MessageBox prompts the user with a box that has a Yes and a No button rather then an OK button. You can capture the result of the MessageBox immediately after the flag by using a runtime *?Variable*. The variable value is set to Yes, No, or Cancel.
Syntax Examples:	MessageBox "Script completed successfully” MessageBox -YesNo ?Result "Do you want to continue?” MessageBox -YesNoCancel ?Result -Background -DefaultNo "Do you want to continue?”

Item	Description

Example 1: Windows Script
SecureLogin detects the password dialog box, asks the user whether the user wants to change the password, and informs the user that the change was successful.
	`# Change Password Dialog Box`
	`Dialog`
	`  Title "Change Password"`
	`  Class #32770`
	`EndDialog`
	
	`MessageBox -YesNo ?Result "Your password has expired. Would you like to change it now?"`
	
	`If ?Result Eq "Yes"`
	`  Type $Username #1015`
	`  Type $Password #1004`
	`  ChangePassword $Password Random`
	`  Type $Password #1005`
	`  Type $Password #1006`
	`  Click #1`
	`  MessageBox "Password changed successfully"`
	`Else`
	`  Click #2`
	`  MessageBox "You elected not to change your password."`
	`EndIf`

Example 2: Terminal Launcher Script
Message boxes can be useful when troubleshooting scripts. The boxes can be displayed before each step in the script to enable the writer to see where the script fails to execute.
 	
NOTE: The WaitForText command cuts off the first character because it will find both Password and password and respond to all password entry points.
	`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"`

Multiply

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	3.0
Type:	Variable manipulator
Usage:	Multiply *Variable1 Variable2* [?Result]

Item	Description
Arguments:	
Variable1	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 *Variable1* argument without the [?Result] argument, *Variable1* must be a SecureLogin variable (either ?Variable1 or $Variable1). Otherwise, *Variable1* can be any numeric value.
Variable2	The multiplier. The second argument. The number that the first number will be multiplied by. *Variable2* can be a SecureLogin variable or a numeric value.
[?Result]	Optional. The product or result of the equation.
Description:	Multiplies one number by another. 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.
Syntax Examples:	Multiply "1" "2" ?Result Multiply ?LoginAttempts ?LoginFailures Multiply ?LoginAttempts ?LoginFailures ?Result Multiply ?LoginAttempts "3" Multiply ?LoginAttempts "3" ?Result

Example:
Windows Script
The values of control IDs 103 and 104 are read into variables. From there they are multiplied, then typed into control ID 1.
	`ReadText #103 ?Number1`
	`ReadText #104 ?Number2`
	`Multiply ?Number1 ?Number2 ?Result`
	`Type ?Result #1`

OnException/ClearException

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	3.0.4
Type:	Flow control
Usage:	OnException *Exception Name* Call *SubRoutine* ClearException *Exception Name*

Item	Description
Arguments:	
Exception Name	The name of the exception that you want to act on. Two exceptions are supported: ◆ ChangePasswordCancelled When a user clicks Cancel in the Change Password dialog box. ◆ EnterVariablesCancelled When a user clicks Cancel in the automatic variable prompt dialog box.
Subroutine	The name of the subroutine you want to run when the exception condition is found to be true.
Description:	Detects when certain conditions are met. Currently, this is when the Cancel button is clicked in either of two dialog boxes. When the condition is met, a subroutine is run. Use the ClearException command to reset the exceptions value.
Syntax Examples:	OnException ChangePasswordCancelled Call DisplayError ClearException ChangePasswordCancelled

Example 1:
Windows Script
The login has failed because the user has invalid credentials stored. Provide the user with an opportunity to verify the username and password. If the user clicks Cancel, the exception is executed. The user must then enter credentials.
	`# Login Failed Dialog Box`
	`Dialog`
	`  Title "Login Failed"`
	`  Class #32770`
	`EndDialog`
	
	`OnException EnterVariablesCancelled Call VariablesCancelled`
	`DisplayVariables "Verify your Username and Password and try again. Helpdesk x5555."`
	`ClearException EnterVariablesCancelled`
	
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`
	
	`Sub VariablesCancelled`
	`  OnException EnterVariablesCancelled Call VariablesCancelled`
	`  DisplayVariables "You cannot cancel this verification dialog box. Verify your username and password when prompted. Then click OK to retry logging in."`
	`  ClearException EnterVariablesCancelled`
	`EndSub`

Item	Description

Example 2: Windows Script
The user has been prompted to change the password. 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).
	`# Change Password Dialog Box`
	`Dialog`
	`  Title "Change Password"`
	`  Class #32770`
	`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
Use with:	Startup scripts, Windows
SecureLogin Version:	All
Type:	Dialog specifier
Usage:	Parent EndParent
Arguments:	None

Item	Description
Description:	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. 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. 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. 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. 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. 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. **TIP:** 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.

Example 1: Windows Script
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".
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class "Centura:Dialog"`
	`  Ctrl #4098`
	`  Ctrl #4100`
	`  Parent`
	`  Class "Centura:MDIFrame"`
	`  EndParent`
	`EndDialog`
	
	`Type $Username #4098`
	`Type $Password #4100`
	`Click #4101`

Example 2: Windows Script
The Parent command is used to click a button on the Login window's parent.
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`EndDialog`
	
	`Type $Username #1001`
	`Type $Password #1002`
	`Parent`
	`  Click #1`
	`EndParent`

PickListAdd

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	All
Type:	Action
Usage:	PickListAdd *Display-Text* [Return-Value]
Arguments:	
Display-Text	The text that will be displayed in the pick list for the specified option.
[Return-Value]	The value returned from the pick list. If you don't specify this parameter, the return will be the display text.
Description:	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. 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. Always use PickListAdd with the PickListDisplay command. Also, you typically use PickListAdd with the SetPlat command.

Example:
Windows Script
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.
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`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`

PickListDisplay

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	All
Type:	Action
Usage:	PickListDisplay *?Variable Display-Tex*t [NoEdit]

Item	Description
Arguments:	
?Variable	The output variable for the selected option.
*Display-Tex*t	The description text for the pick list box.
[NoEdit]	Prevents users from adding entries to the pick list.
Description:	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.
Syntax Examples:	PickListDisplay ?Choice PickListDisplay ?Choice "Select the account you want to use" PickListDisplay ?Choice "Select the account you want to use" NoEdit

Example:
Windows Script
The user has three accounts for this application and wants to be able to pick which one to use. The user picks an account and SecureLogin (using the SetPlat command) switches to that set of credentials.
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`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`

PositionCharacter

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

Item	Description
Description:	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.
Example: The password won't be valid unless the first, sixth and seventh characters are in uppercase.	`PositionCharacter Uppercase 1,6,7`

ReadText

Item	Description
Use with:	Startup scripts, Terminal Launcher, Windows
SecureLogin Version:	All
Type:	Action
Windows Usage:	ReadText #Ctrl-*ID ?Variable*
Terminal Launcher Usage:	ReadText *?Variable Character-Number Row-Number*
Arguments:	
#Ctrl-ID	The control ID number of the text to be read.
?Variable	The variable that will receive the text that is read.
Character-Number	The number of characters to be read.
Row Number	The horizontal position number of the first character to be read (for example, row).
Description:	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. In a Windows script, the ReadText command reads the text from any given *#Ctrl-ID* and sends it to the specified variable. For this command to function correctly, the *#Ctrl-ID* must be valid. In a Terminal Launcher script, the ReadText command reads a specified number of characters, starting at the *Row-Number*, and sends those characters to the specified *Variable*. The ReadText command won't work with Generic or Advanced Generic emulators. It only works with HLLAPI and some DDE emulators.
Syntax Examples:	ReadText #301 ?Text ReadText ?Text 4 6

Item	Description

Example 1: Windows Script
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.
	`# Login Failed Message`
	`Dialog`
	`  Title "Login Failed"`
	`  Class #32770`
	`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`

Example 2: Windows Script
Read the text from a Control ID and set the Database variable so the user isn't prompted to.
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`EndDialog`
	
	`ReadText #15 ?Database`
	
	`If -Exists $Database`
	`Else`
	`  Set $Database ?Database`
	`EndIf`
	
	`Type $Username #1001`
	`Type $Password #1002`
	`Type $Database #1003`
	`Click #1`

Example 3: Terminal Launcher Script Read a message in a Terminal Emulator and display it in a user -friendly format.	`ReadText ?Message 30 24 2` `MessageBox ?Message`

RegSplit

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	All
Type:	Action
Usage:	RegSplit *RegEx Input-Strin*g O*utput-String1 Output-String2*
Arguments:	
RegEx	The regular expression.
*Input-Strin*g	The string that will be split.
O*utput-String1*	The first subexpression.
Output-String2	The second subexpression.
Description:	Enables you to split a string by using a regular expression. *Output-String1* contains the first subexpression. *Output-String2* contains the second subexpression.

Example:
Windows Script
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 platform is set to that username, and SecureLogin enters the correct password
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`EndDialog`
	
	`ReadText #65535 ?Text`
	`RegSplit "Enter the password for (.*) account"`
	`?Text ?User`
	`SetPlat ?User`
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

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

Repeat / EndRepeat

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	All
Type:	Action

Item	Description
Usage:	Repeat [Loop#] EndRepeat
Arguments:	
[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.
Description:	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.
Syntax Examples:	Repeat Repeat 3

Item	Description

Example: Terminal Script The Repeat command watches the screen for messages and responds accordingly. The Break command jumps to the next repeat loop in the script.	`# Initial System Login`
	`WaitForText "ogin:"`
	`Type $Username`
	`Type @E`
	`WaitForText "assword:"`
	`Type $Password`
	`Type @E`
	`Delay 500`
	
	`# Repeat loop for error handling`
	`Repeat`
	
	`#Check to see if 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
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	All
Type:	Action
Usage:	RestrictVariable *Variable-Name Password-Policy*
Arguments:	
Variable-Name	The name of the variable to restrict.
Password-Policy	The name of the policy to enforce on the variable.
Description:	Monitors a *Variable* and enforces a specified *Password Policy* on the *Variable*. Any variable specified must match the policy or it won't be saved. 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.) 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. 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.

Item	Description

Example 1: Windows Script
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.
	`# Set the Password to use the Finance Password Policy`
	`RestrictVariable $Password FinancePwdPolicy`
	
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`EndDialog`
	
	`Type $Username #1001`
	`Type $Password #1002`
	
	`# Change Password Dialog Box`
	`Dialog`
	`  Title "Change Password"`
	`  Class #32770`
	`EndDialog`
	
	`Type $Username #1015`
	`Type $Password #1004`
	`ChangePassword $Password Random`
	`Type $Password #1005`
	`Type $Password #1006`
	`Click #1`

Example 2: Windows Script
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.
	`# Set the Password to use the Finance Password Policy`
	`RestrictVariable ?NewPwd FinancePwdPolicy`
	
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`EndDialog`
	
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`
	
	`# Change Password Dialog Box`
	`Dialog`
	`  Title "Change Password"`
	`  Class #32770`
	`EndDialog`
	
	`Type $Username #1015`
	`Type $Password #1004`
	`ChangePassword ?NewPwd Random`
	`Type ?NewPwd #1005`
	`Type ?NewPwd #1006`
	`Set $Password ?NewPwd`
	`Click #1`

Run

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows

Item	Description
SecureLogin Version:	All
Type:	Action
Usage:	Run *Command* [*Arg1* [*Arg2*] ...]
Arguments:	
Command	The full path of the command to be executed.
[*Arg1* [*Arg2]...]*	An optional list of arguments or switches for the command.
Description:	Launches the program specified in the *Command* with the specified optional [*Arg1* [*Arg2*] ...] arguments. The script doesn't wait for the launched program to complete.

Example:
Startup Script
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.
	`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`

SelectListBoxItem

Item	Description
Use with:	Advanced Web Scripts
SecureLogin Version:	3.5
Type:	Action
Usage:	SelectListBoxItem *Text of Item to set to* [*item Number*] [*-multiselect*]
Arguments:	
Text of Item to set to	The text item that you want SecureLogin to select in the list box.
[*Item Number*]	When multiple list boxes are found, this specifies which list box to address.
[*-multiselect*]	Used to select multiple list box entries by using a subsequent SelectListBoxItem command.
Description:	Selects entries from a list box.
Example:	`SelectListBoxItem "Remember Defects" #2 -multiselect` `SelectListBoxItem "Remember Enhancements" #2 -multiselect`

SendKey

Item	Description
Use with:	Terminal Launcher
SecureLogin Version:	All
Type:	Action
Usage:	SendKey *Text*
Arguments:	
Text	The text to be typed into the emulator screen.
Description:	Works only with Generic and Advanced Generic emulators. Use SendKey in the same manner as the Type command. Generally, the Type command is the preferred command to use. The Type command places the text into the Clipboard and then pastes it into the emulator screen. The SendKey command enters the text directly into the emulator screen. Variables don't work with the SendKey command. If you want to use variables, use the Type command. The Type command has many special functions, some of which can also be used with the SendKey command. For further details on these functions, see "Type" on page 96.

Example: Terminal Launcher Script
The SendKey command sends the username and password to the terminal emulator.
	`#Send Username`
	`SendKey mkurz`
	`SendKey "\N"`
	`#Send Password`
	`SendKey "Hu7%f"`
	`SendKey "\N"`

Set

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	All
Type:	Action
Usage:	Set *Variable Data*
Arguments:	
Variable	The variable that the data is being assigned to.
Data	The text or variable being read from and assigned to the variable.

Item	Description
Description:	Copies the value of *Data* into *Variable*. The *Data* can be any text or another variable. However, the *Variable* argument must be a ?Variable or $Variable.

Example 1: Windows Script
The script sets a ?RunCount variable to count the number of times the application is run.
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`EndDialog`
	
	`If ?RunCount Eq NOTSET`
	`  Set ?RunCount "1"`
	`Else`
	`  Increment ?RunCount`
	`EndIf`
	
	`Type $Username #10091`
	`Type $Password #1002`
	`Click #1`

Example 2: Windows Script
The script sets the ?NewPwd to the stored variable, $Password.
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`EndDialog`
	
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`
	
	`# Change Password Dialog Box`
	`Dialog`
	`  Title "Change Password"`
	`  Class #32770`
	`EndDialog`
	
	`Type $Username #1015`
	`Type $Password #1004`
	`ChangePassword ?NewPwd Random`
	`Type ?NewPwd #1005`
	`Type ?NewPwd #1006`
	`Set $Password ?NewPwd`
	`Click #1`

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

SetCheckBox

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

Example:	`Messagebox "Scroll down so you can see the 'Search Language' section and all the languages with the check boxes, then click OK on this messagebox."`
	`SetCheckBox #1 "checked"`
	`SetCheckBox #2 "checked"`
	`SetCheckBox #3 "checked"`
	`SetCheckBox #4 "checked"`
	`SetCheckBox #25 "checked"`
	`SetCheckBox #26 "checked"`
	`SetCheckBox #27 "checked"`
	
	`Messagebox "Did it select the first four languages and Norwegian, Polish and Portuguese Languages" -yesno ?advweb`
	
	`If ?advweb Eq yes`
	`  Set ?cmd37 "SetCheckBox command worked"`
	`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`

SetCursor

Item	Description
Use with:	Terminal Launcher (Only available in HLLAPI and some DDE emulators)
SecureLogin Version:	All
Type:	Action
Usage 1:	SetCursor *Screen-Position*
Usage 2:	SetCursor *X Coordinate* Y *Coordinate*
Arguments:	
Screen-Position	On the screen, the position that the cursor should be moved to.
X Coordinate	The horizontal coordinate. When *X-Coordinate* is specified, a row/column conversion is carried out before the cursor is set to the position.
Y Coordinate	The vertical coordinate. When *Y-Coordinate* is specified, a row/column conversion is carried out before the cursor is set to the position.
Description:	Sets the cursor to a specified *Screen Position* or *X Coordinate Y Coordinate*. 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.
Syntax Examples:	SetCursor 200 SetCursor 100 500

Example:
Terminal Launcher Script
The cursor is set to the correct position, then the credentials are entered.
	`SetCursor 200`
	`Type $Username`
	`Type @E`
	`Type $Password`
	`Type @E`

SetFocus

Item	Description
Use with:	Startup scripts, Windows
SecureLogin Version:	All
Type:	Action
Usage:	SetFocus *#Ctrl-ID*
Arguments:	
#Ctrl-ID	The ID number of the control that the keyboard focus will be directed to.

Item	Description
Description:	Gives the keyboard focus to a specified *#Ctrl-ID*. For the SetFocus command to function correctly, the *#Ctrl-ID* must be valid.

Example:
Windows Script
SecureLogin sets the focus to the username field (#1001), types the username, simulates the tab, types the password, then simulates Enter.
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`EndDialog`
	
	`SetFocus #1001`
	`Type $Username`
	`Type \T`
	`Type $Password`
	`Type \N`

SetPlat

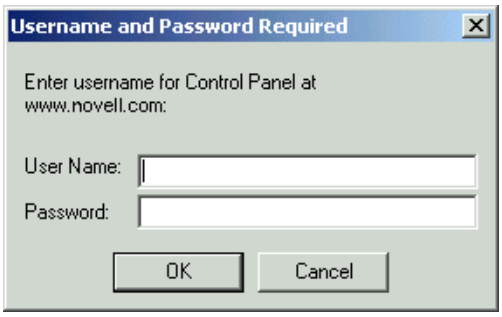
Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	All
Type:	Action
Usage 1:	SetPlat *Application-Name*
Usage 2:	SetPlat *RegEx Variable #Ctrl-ID*
Arguments:	
Application-Name	The application name that the variables are read from.
RegEx	A regular expression to be used as the application name.
Variable	Must be a ?Variable previously set (for example, using a pick list).
#Ctrl-ID	The control ID of the regular expression to be used.

Item	Description
Description:	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. 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. You can also use SetPlat to do the following: ◆ Tell 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. ◆ Tell 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. ◆ Create new applications, depending on what a user selected in a pick list. If the *Application-Name* doesn't exist, it will be created. SetPlat can also read from a *#Ctrl-ID* and support regular expressions.

Example 1: Windows Script
SecureLogin displays a pick list and sets a new platform so that multiple users can log in to the application. In this case, SetPlat creates a new platform called Default User, Global Administrator, or Regional Administrator. The respective $Username and Password are saved there.
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`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 (.*):"
```

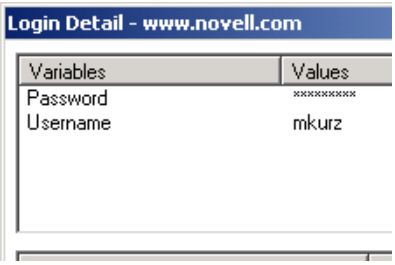
The script first reads the following line from dialog control ID 331:

```
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
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	All
Type:	Action
Usage:	SetPrompt *Prompt-Text*
Arguments:	
Prompt-Text	The customized text prompt to be displayed in the Enter SecureLogin Variables dialog box.
Description:	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.

Example 1: Windows Script
To replace the default text prompt in the Enter SecureLogin Variables dialog box, place the SetPrompt command at the bottom of the script
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`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"`

Item	Description

Example 2: Windows Script 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.	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`EndDialog`
	
	`SetPrompt "Enter Username==>"`
	`Type $Username #1001`
	
	`SetPrompt "Enter Password==>"`
	`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"`

Strcat

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	All
Type:	Action
Usage:	Strcat *Variable Input-String1 Input-String2*
Arguments:	
Variable	The variable that you want the result saved to.
Input-String1	The first data string or variable.
Input-String2	The second data string or variable.
Description:	Appends the second data string to the first data string. **Scenario:** You include the following line in a script: `StrCat ?Result "SecureRemote" "$Username"` When $Username is Tim, the ?Result variable contains the value "SecureRemote Tim."

Item	Description

Example: Windows Script 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.	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`EndDialog`
	
	`Readtext #1001 ?Username`
	`StrCat ?LoginID ?Username $Password`
	`Type ?LoginID #1002`
	`Click #1`

StrLength

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	3.0.4
Type:	Variable manipulator
Usage:	StrLength *Destination* [*Source]*
Arguments:	
Destination	The output variable. Also the input variable if no source is specified.
[*Source*]	The input variable. If you don't specify a source, SecureLogin reads the destination variable, makes the calculations, and writes over the variable.
Description:	Counts the number of characters in a variable and outputs that value to the *destination* variable. If only a destination variable is specified, the string is read from the destination, then the value is stored back in to it. If a source variable is specified, the string is read from the source, and the calculated value is stored in the destination variable. In this case, the source variable remains unchanged.

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

StrLower

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	3.0.4
Type:	Variable manipulator
Usage:	StrLower *Destination* [*Source*]
Arguments:	
Destination	The output variable. Also the input variable if no source is specified.
[*Source*]	The input variable. If not specified, SecureLogin reads the destination variable, makes the necessary changes, and writes over it.
Description:	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.

Example:
Windows Script
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.
	`#Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`EndDialog`
	
	`Readtext #1001 ?Username`
	`StrLower ?LowerCaseUsername ?Username`
	`Type ?LowerCaseUsername #1002`
	`Click #1`

StrUpper

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	3.0.4
Type:	Variable manipulator
Usage:	StrUpper *Destination* [*Source*]
Arguments:	
Destination	The output variable. Also the input variable if no source is specified.
[*Source*]	The input variable. If you don't specify a source, SecureLogin reads the destination variable, makes the necessary changes, and writes over the variable.

Item	Description
Description:	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.

Example:
Windows Script
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.
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`EndDialog`
	
	`Readtext #1001 ?Username`
	`StrLower ?UpperCaseUsername ?Username`
	`Type ?UpperCaseUsername #1002`
	`Click #1`

Sub / EndSub

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

Example:
Terminal Launcher Script
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 "Login"`
	`  Call Login`
	`EndIf`
	
	`If -Text "Wrong Password"`
	`  Call "WrongPassword"`
	`EndIf`
	
	`Sub Login`
	`  Type $Username`
	`  Type @E`
	`  Type $Password`
	`  Type @E`
	`EndSub`
	
	`Sub WrongPassword`
	`  DisplayVariables "Enter correct password" $Password`
	`  Call Login`
	`EndSub`

Submit

Item	Description
Use with:	Web
SecureLogin Version:	3.0
Type:	Action
Usage:	Submit
Arguments:	None

Description:	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.
	
	When used with Netscape, the Submit command is ignored.
	
	Submit occurs automatically with most scripts. For example, the following script for Hotmail* automatically does a Submit at the end:
	
	`Type $Username`
	`Type $Password Password`
	
	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:
	
	◆ Type \N
	◆ Type \T
	◆ Submit
	◆ Click
	
	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:
	
	`Type $Username #1001`
	
	`Submit`

Example:
Web Script
The script enters the username and password, then performs a manual Submit.
	`Type $Username #1`
	`Type $Password Password #2`
	`Submit`

Subtract

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	3.0
Type:	Variable manipulator
Usage:	Subtract *Start-Value Subtract-Value* [?Result]
Arguments:	
Start-Value	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 *Start-Value* argument without the [?Result] argument, *Start-Value* must be a SecureLogin variable (for example, ?*Start-Value* or $*Start-Value*). If the [?Result] argument is provided, *Start-Value* can be a SecureLogin variable or a numeric value.
Subtract-Value	The number that will be subtracted from the first argument. *Subtract-Value* 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).
Description:	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 Slina DLL) can be used to determine the number of days that have elapsed since the last password change. The Subtract command correctly subtracts when *Start-Value*, *Subtract-Value* and *Result* are between -2147483648 and +2147483647. (Doesn't work with fractions.)
Syntax Examples:	Subtract "1" "2" ?Result Subtract ?LoginAttempts ?LoginFailures Subtract ?LoginAttempts ?LoginFailures ?Result Subtract ?LoginAttempts "3" Subtract ?LoginAttempts "3" ?Result

Example:
Windows Script
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
Use with:	Advanced Web Script
SecureLogin Version:	3.5
Type:	Tag specifier
Usage:	Tag EndTag
Arguments:	None
Description:	Used to find HTML tags.

Example:
SecureLogin finds the form that has an attribute of "name" with a value of "login".
	`Tag "Form"`
	`  Attribute "Name" "Login"`
	`EndTag`

Title

Item	Description
Use with:	Startup scripts, Windows
SecureLogin Version:	All
Type:	Dialog specifier
Usage:	Title *Window-Title*
Arguments:	
Window-Title	The text to test against the window title.
Description:	Retrieves the title of the window and compares it against the string specified in the *Window-Title* argument. For this section of the script to run, the retrieved window title and the *Window-Title* argument must match exactly. 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. To uniquely identify a window, typically use the Title command with the Class or Ctrl command. You can use Window Finder to locate the window title. See “Finding Control IDs” on page 33.

Item	Description

Example: Windows Script
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.
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`EndDialog`
	
	`Type $Username #1001`
	`Type $Password #1002`
	`Click #1`

Type

Item	Description
Use with:	Startup scripts, Terminal Launcher, Web, Windows
SecureLogin Version:	All
Type:	Action
Terminal Launcher Usage:	Type [-Raw] *Text*
Windows Usage:	Type *Text* [*#Ctrl-ID*] Type [-Raw] *Text*
Web Usage:	Type *Text* [*#Field-ID*] Type *Text* [*Field-Name*]
Arguments:	
[-Raw]	By default, when typing into a Terminal Emulator or Windows application, SecureLogin verifies that the window exists before continuing. This verification process is disabled when the -Raw argument is provided, causing SecureLogin to type into whichever window has focus.
Text	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 "Windows-Specific Information" on page 97.
[*Field-ID*]	For Web scripts, this optional argument specifies the text field that the text will be typed in to. See."Web-Specific Information" on page 97.
[*Field-Name*]	For Web scripts, this optional argument specifies the name of the text field that the text will be typed in to. If you use this format, use it only for one Type command. All other Type commands in that application's script can't use a *#Ctrl-ID* argument. See "Web-Specific Information" on page 97.
[*key command*]	A keyboard function. See "Sending Keyboard Commands by Using Type" on page 98.

Item	Description
Description:	Used to enter data (for example, usernames and passwords) into applications. Some character sequences are reserved. These character sequences are used to type special characters like Tab and Enter. See Chapter 7, "Keystrokes and Functions," on page 113. If you can't determine control IDs, and the Type command is not working, you can use the SendKey command. This option is often useful in Terminal Launcher scripts where the emulator program is interpreting character keystrokes and interfering in the operation of a script. **Windows-Specific Information** In Windows, if the *#Ctrl-ID* 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 *Text* argument directly to the window and to the specific control that matches the *#Ctrl-ID* argument If the *Ctrl-ID* is not specified, SecureLogin sends keystrokes to whichever control has focus. In the Windows environment, the -Raw option is often useful when you work with Java applications and other applications where Window Finder is unable to determine control IDs for the text entry areas of an application. When you use the -Raw option, don't use the *#Ctrl-ID* argument. **Web-Specific Information** You can use either of two methods to specify which field receives *Text*. The first method uses absolute positioning through the *#Field-ID* argument. The *#Field-ID* is a number that refers to the location of the field within the HTML form. For example, #1 refers to the first text entry field in the Web form, and #2 refers to the second text entry field. The second method uses relative position via the *#Field-Name* argument. In this method, the SecureLogin agent first locates the text field within the HTML form that has a name that exactly matches the argument *Field-Name.* Then the *Text* is sent to that field. Other Type commands send their *Text* parameters to fields that are relative to this identified text field. For example, the Type command immediately preceding the Type command with the *Field-Name* argument is sent to the text field before the identified text field. See the example Web script.

Example 1: Windows Script
A typical use of the Type command in a Windows script.
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`EndDialog`
	
	`Type $Username #1001`
	`Type $Password #1002`
	`Type "DB2" #1003`
	`Click #1`

Item	Description

Example 2: Windows Script This typical example shows the use of the -Raw switch.
 	
NOTE: The -Raw switch is not actually required in this instance. It is only an example.
	`# Calculator Is Active`
	`Dialog`
	`  Title "Calculator"`
	`  Class #SciCalc`
	`EndDialog`
	
	`Type -Raw "15"`
	`Type -Raw "+"`
	`Type -Raw "20"`
	`Type -Raw "="`

Example 3: Web Script The SecureLogin agent automatically generated this script for the mail.yahoo.com site. This example shows the use of Password as the [*Field-Name*] argument.	In the following script, the SecureLogin agent first locates the text field called Password. The first Type command sends $Username to the field immediately before the Password field. The second Type command sends $Password to the text field located at the beginning.
	
	`Type $Username`
	`Type $Password Password`
	
	The same script could be rewritten using absolute placement. In this example script, the Submit command is also used to automatically submit the page.
	
	`Type $Username #1`
	`Type $Password #2`
	`Submit`

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 *key*.

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

Raw Key Commands

Commands	Description
Type |*xxx*	The format for sending a raw key command, where *xxx* represents the keyboard code. See Appendix D, "Keyboard Functions and Codes," on page 127.
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 114 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
Use with:	Startup scripts, Windows
SecureLogin Version:	All
Type:	Flow control
Usage:	WaitForFocus #*Ctrl-ID* [*Repeat-Loops*]

Item	Description
Arguments:	
#Ctrl-ID	The ID number of the control that will have the focus.
[*Repeat-Loops*]	The number of repeat loops that will run.
Description:	Suspends running the script until the *#Ctrl-ID* has received keyboard focus or until the *Repeat-Loops* argument expires. The *Repeat-Loops* argument is an optional value. It defines the number of loop cycles that will be run. The *Repeat-Loops* value defaults to 3000 loops if nothing is set. As soon as focus is received, the script continues. As the following line illustrates, you can set *Repeat-Loops* to never expire by setting the figure to a negative number: `WaitForFocus "#1065" "-1"` If the #Ctrl-ID is set to 0 (zero), it loops until the window defined in the Dialog / EndDialog statement is given keyboard focus. **NOTE:** Don't place WaitForFocus commands within Dialog / EndDialog statements.
Syntax Examples:	WaitForFocus #301 WaitForFocus #301 "2000" WaitForFocus #301 "0" WaitForFocus #301 "-1

Example:
Windows Script
After the Login dialog box has been detected, SecureLogin waits indefinitely for window #301 to get focus before entering the user credentials.
	`# Login Dialog Box`
	`Dialog`
	`  Title "Login"`
	`  Class #32770`
	`EndDialog`
	
	`WaitForFocus #301 "-1"`
	`Type $Username`
	`Type \T`
	`Type $Password`
	`Type \N`

WaitForText

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

Item	Description
Description:	Causes Terminal Launcher to wait for the specified *Text* 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. The *Text* can appear anywhere on the terminal screen. The *Text* is case-sensitive. If the *Text* is written in the wrong case, Terminal Launcher pauses and tries to find the correct *Text* in the correct case, pausing until the terminal screen times out. If WaitForText isn't working, try leaving the initial letter off the *Text*, 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". `WaitForText "ogin″` However, WaitForText "Login" will only work if "login" is presented on the screen as "Login". Also, some terminal emulators won't correctly match text that is hard against the left margin of the window. Again, if you encounter this situation, try to match text without the leading character.

Example: Terminal Launcher
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
	`WaitForText "ogin:"`
	`Type $Username`
	`Type @E`
	`WaitForText "assword:"`
	`Type $Password`
	`Type @E`

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.

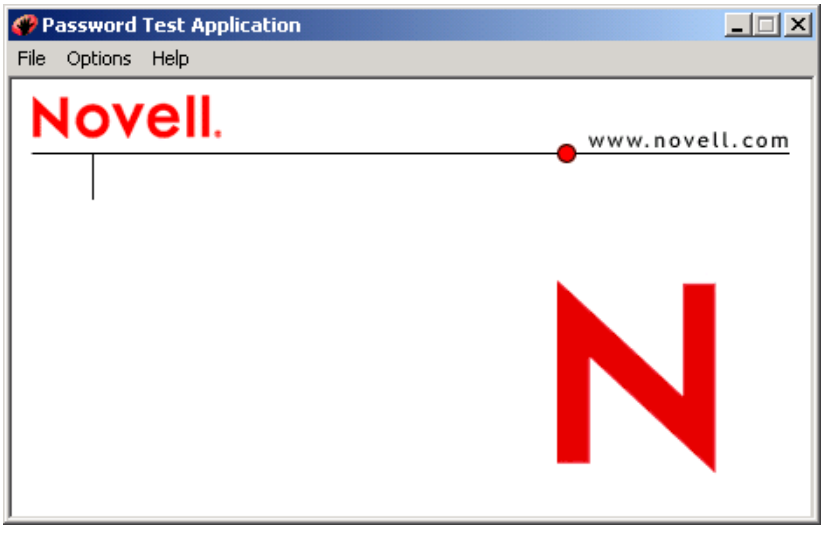
1. Run SecureLogin.

   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.

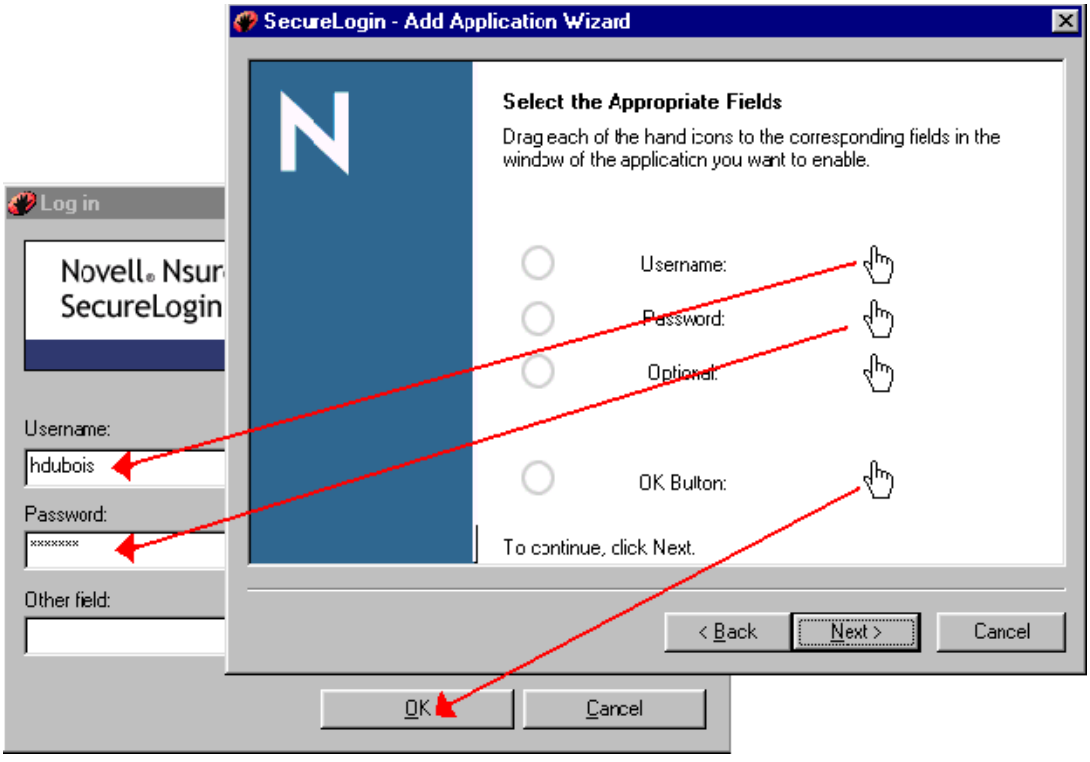
3d Click and drag the Add Application Wizard to one side of your screen, so that you can also view and work with the PasswordTest.exe dialog box.

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 "Adding a Windows Application" in the Nsure SecureLogin 3.51 Administration Guide.

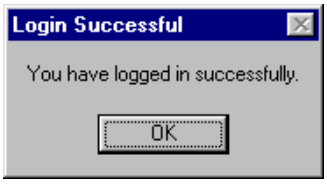
3g In the Add Application Wizard, click Next.

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 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.

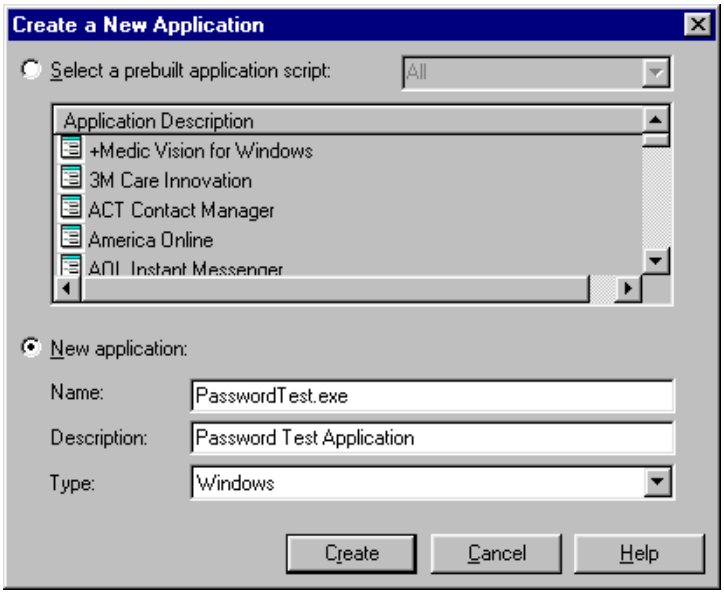
2 Create a new PasswordTest.exe application.

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.



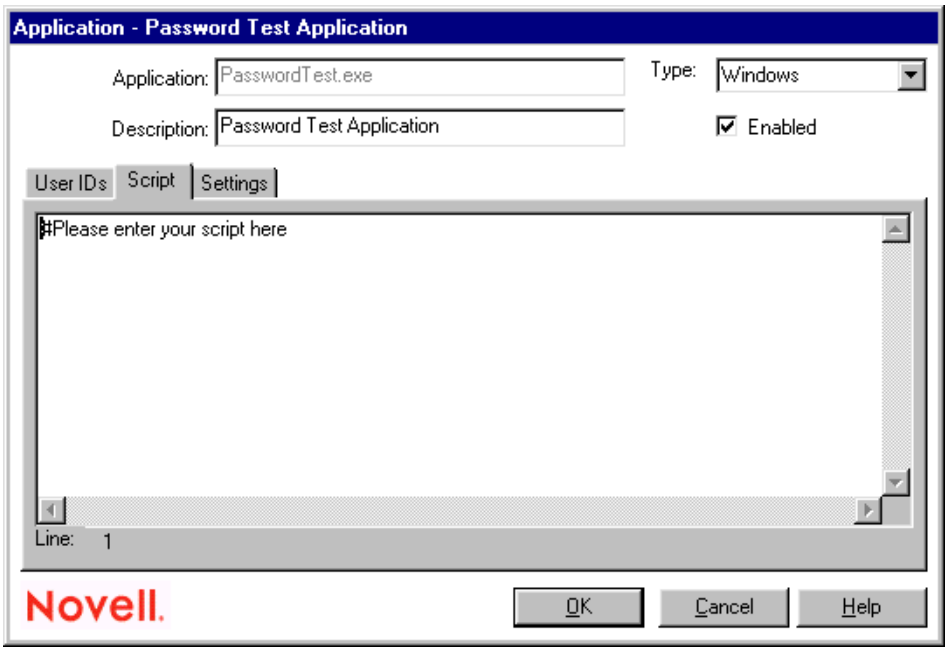
2d In the Name edit box, type PasswordTest.exe.

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 78.

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 49.

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 95.

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

For information on this command, see "Class" on page 44.

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 88.

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 96.

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 45.

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..

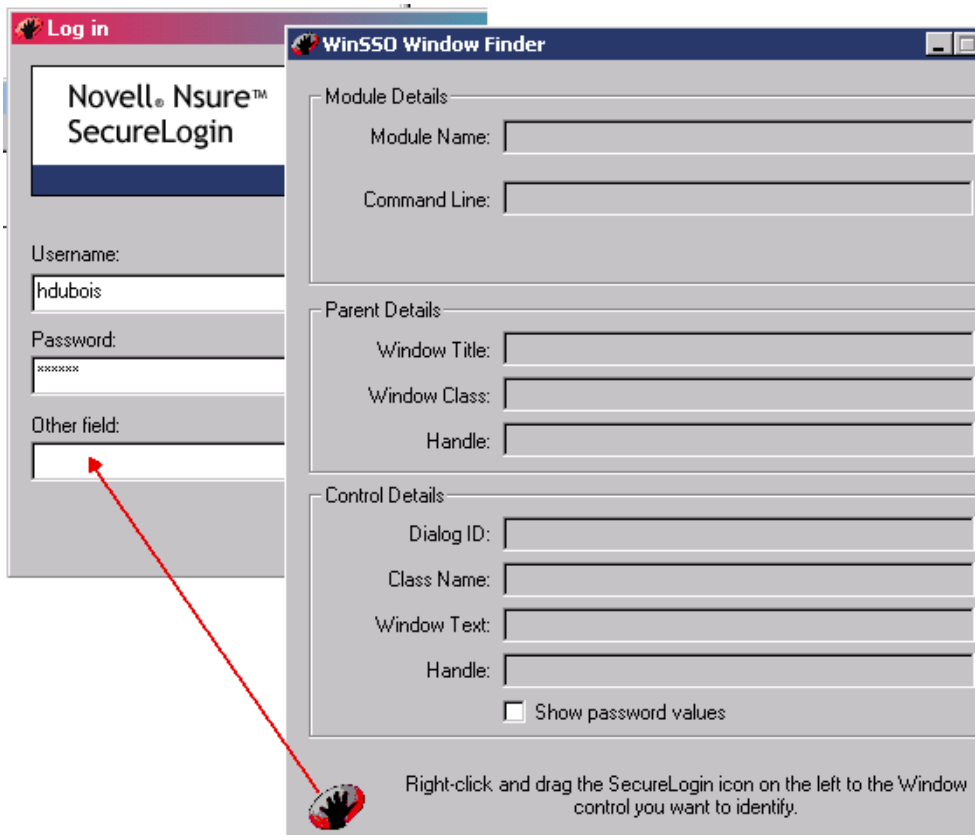
If the script contains an error, the Enter SecureLogin Variables dialog box doesn't appear. Instead, SecureLogin displays an error message, specifying the script line that contains the error. Return to the script and resolve the problem.

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

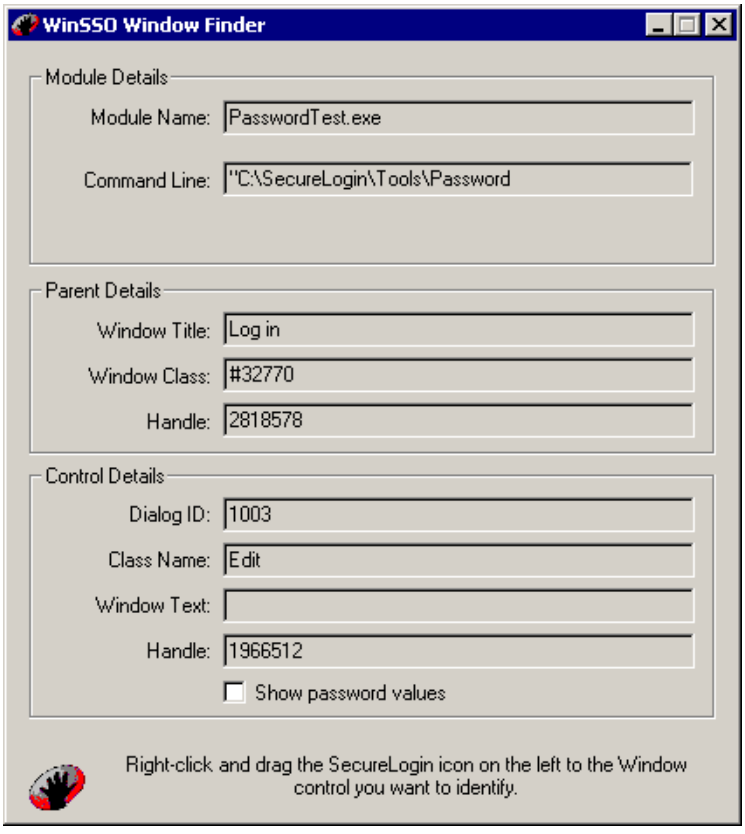
Experiment: Using Window Finder

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 64.

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 81.

ChangePassword: For information on this command, see "ChangePassword" on page 43.

3 Add a message block.

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

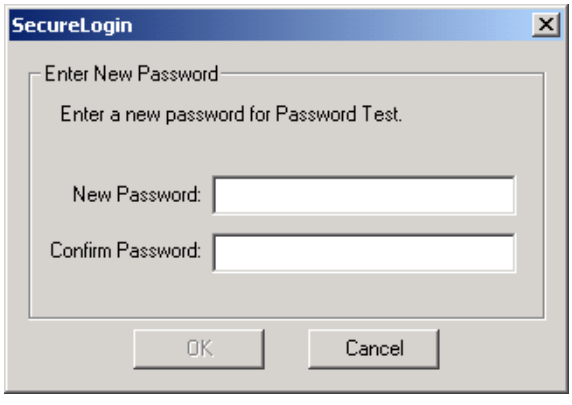
4 Add the remainder of the script.

```
# 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

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
Sent 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

The Type Command	Meaning	The Type Command	Meaning
@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

The Type Command	Meaning	The Type Command	Meaning
@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 iformation on the following;

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	$password password	Enters the value of the variable $password and displays asterisks because the first eight letters of the variable are `password`.
Type	$juanspassword password	Enters the value of the variable $juanspassword, but not as asterisks, unless the Web page masks the entry with asterisks.
Type	$password4juan password	Enters the value of the variable $password and displays asterisks because the first eight letters of the variable are `password`.

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 file 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.

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)

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
Click	SS, Web, Win	Action	All
ConvertTime	SS, TL, Web, Win	Variable manipulator	3.0.4
Ctrl	SS, Win	Dialog specifier	All
Delay	SS, TL, Web, Win	Action	All
Dialog/EndDialog	SS, Win	Dialog specifier	All
DisplayVariables	SS, TL, Web, Win	Action	All
Divide	SS, TL, Web, Win	Variable manipulator	3.0
DumpPage	Advanced Web Script	Action	3.5
DumpScript	Advanced Web Script	Action	3.5
EndScript	SS, TL, Web, Win	Action	All
GetCheckBoxState	Advanced Web Script	Action	3.5

Command	Use With	Type	SecureLogin Version
GetCommandLine	SS, Win	Action	3.0.4
GetSessionName	Terminal Emulator	Action	3.5
GetText	Web	Action	3.0
GetURL	Web	Action	3.0
GoToURL	Web	Action	2.5
If/Else/EndIF	SS, TL, Web, Win	Flow control	All
Increment/Decrement	SS, TL, Web, Win	Variable manipulator	All
KillApp	SS, TL, Web, Win	Action	All
Local	SS, TL, Web, Win	Variable manipulator	3.0
MessageBox	SS, TL, Web, Win	Action	All
Multiply	SS, TL, Web, Win	Variable manipulator	3.0
OnException	SS, TL, Web, Win	Flow control	3.0.4
Parent/EndParent	SS, Win	Dialog specifier	All
PickListAdd	SS, TL, Web, Win	Action	All
PickListDisplay	SS, TL, Web, Win	Action	All
ReadText	SS, TL, Win	Action	All
RegSplit	SS, TL, Web, Win	Action	All
Repeat/EndRepeat	SS, TL, Web, Win	Flow control	All
RestrictVariable	SS, TL, Web, Win	Action	All
Run	SS, TL, Web, Win	Action	All
SelectListBoxItem	Advanced Web Script	Action	3.5
SendKey	TL	Action	All
Set	SS, TL, Web, Win	Variable manipulator	All
SetCheckBox	Advanced Web Script	Action	3.5
SetCursor	TL	Action	All
SetFocus	SS, Win	Action	All
SetPlat	SS, TL, Web, Win	Action	All
SetPrompt	SS, TL, Web, Win	Action	All
StrCat	SS, TL, Web, Win	Variable manipulator	All
StrLength	SS, TL, Web, Win	Variable manipulator	3.0.4

Command	Use With	Type	SecureLogin Version
StrLower	SS, TL, Web, Win	Variable manipulator	3.0.4
StrUpper	SS, TL, Web, Win	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	SS, Win	Dialog specifier	All
Type	SS, TL, Web, Win	Action	All
WaitForFocus	SS, Win	Flow control	All
WaitForText	TL	Flow control	All

B FAQs on Scripting

This sectioni provides information on the following:

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® eDirectoryTM, 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 libsnmp.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\slsnmp.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.
- *Tex*t is the text to be displayed as the message at the host.

Example Script

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

Run C:\Progra~1\Novell\Secure~1\Slsnmp.exe SNMPCommunity1 192.168.156.23
"PSL - Incorrect password in finance system."
MessageBox "You have entered an incorrect password. The administrator has been
notified. 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	

Function	Decimal	Comment/Information
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	

Function	Decimal	Comment/Information
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	

Function	Decimal	Comment/Information
Left Shift	160	
Right Shift	161	
Left Ctrl	162	
Right Ctrl	162	
Left menu	164	
Right menu	**165**	
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
Volulme 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_CHANGECBCHAIN
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_MBUTTONDBLCLK
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_ICONERASEBKGND	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_LBUTTONDBLCLK	WM_NCLBUTTONDBLCLK
WM_ERASEBKGND	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 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 is 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 Configuration Guide for Terminal Emulation (http://www.novell.com/documentation/lg/securelogin351/index.html), 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 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 Configuration Guide for Terminal Emulation (http://www.novell.com/documentation/lg/securelogin351/index.html), 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 Configuration Guide for Terminal Emulation (http://www.novell.com/documentation/lg/securelogin351/index.html), 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.

G Documentation Updates

This section contains new or updated information on scripting and script commands. The information is new since Nsure™ SecureLogin 3.51.

This documentation is provided on the Web in two formats: HTML and PDF. The HTML and PDF documentation are both kept up-to-date with the documentation changes listed in this section. See Nsure SecureLogin 3.51 (http://www.novell.com/documentation/lg/securelogin351/index.html).

If you need to know whether a copy of the PDF documentation you are using is the most recent, check the date that the PDF file was published. The date is in the Legal Notices section, which immediately follows the title page.

New or updated documentation was published on the following dates:

January 7, 2004

Editing, formatting, and other minor changes were made in various places throughout the *Guide*. For example, the Type for the Tag/EndTag command was changed from Action to Tag Specifier.

April 20, 2004

Location	Change
"Internal Variables" on page 25	Corrected ?sysversion to ?sysversion(system)

April 28, 2004

Changed the -NoEdit switch to NoEdit (without a hyphen). This change occurred in the following commands:

- "PickListAdd" on page 71
- "PickListDisplay" on page 71
- "SetPlat" on page 85

May 13, 2004

Stated that SecureLogin works with whole number but not with fractions. This information applies to the following commands:

- “Add” on page 39
- “Divide” on page 52
- “Multiply” on page 66
- “Subtract” on page 94