# eTrust® Single Sign-On

## Tcl Scripting Reference Guide

r8.1

# CA Product References

This document references the following CA products:

- eTrust® Access Control (eTrust AC)
- eTrust® Directory
- Unicenter® Software Delivery
- eTrust® Audit

# Contact Technical Support

For online technical assistance and a complete list of locations, primary service hours, and telephone numbers, contact Technical Support at http://ca.com/support.

# Contents

# Chapter 1: Introduction to Scripts

This section contains the following topics:



## All About Scripts

eTrust® Single Sign-On (eTrust SSO) simplifies user login to password-protected applications. It also streamlines security administration and enhances overall system security.

eTrust SSO is transparent to the end user. To them, it appears that selecting an application from a menu or double-clicking an application icon opens the application directly. In fact, the user is invoking a program that:

- References a script (also referred to as a logon script)
- Carries out the preliminary checks and instructions given in the script
- Opens the application requested, provided the user is authorized

You must write a set of scripts, one for each application, before you can use eTrust SSO. The security or system administrator in charge of eTrust SSO is responsible for preparing the scripts. Generally, programmers write the scripts under the administrator's supervision.

Previous experience with the Tcl scripting language is not required, but the programmer should be familiar with the applications involved and their login processes.

You can also use the SSO Application Wizard to add your applications to SSO without the need to learn Tcl scripting for basic scripting tasks.

For more information on the Application Wizard, see the section "Using the SSO Application Wizard to Create SSO Scripts" in the *Implementation Guide*.

# About This Book

This book explains how to write scripts for eTrust SSO. It contains information for the following:

- Security and system administrators in charge of eTrust SSO systems
- Programmers who write eTrust SSO scripts
- Users who are allowed to write their own scripts

Script writers should be familiar with eTrust SSO and have a basic understanding of programming concepts.

# Conventions

eTrust SSO documentation assumes the following:

- Your mouse is configured for the right hand.
- Your keyboard keys are shown as they appear on the keyboard. For example, Enter means the key labeled Enter or Return on your keyboard.
- Sometimes, to make clear the difference between what you must copy from the instructions and what you must replace with your own data, bold and italic type are used.
  - Characters that are bold should be copied literally from the instructions. When used outside of syntax, bold is also used for emphasis. For example, you should **never** tape your password to the monitor.
  - Characters that are italic are variables for items that you must supply values, such as parameters. For example, when the manual says to type newusr *userName*, you could type newusr lebois for a new user named lebois, or newusr wald for a user named wald.
- A technique shown in boldface is not necessarily the only effective technique, but the book does not list all alternatives at every opportunity. For example, in describing a sequence of commands, the book may mention newusr wald and not mention that nu wald works equally well.
- Square brackets mean that an item is optional. For example:

  `./install_base` *`filename`* [*`options`*]

  means that you must type ./install_base, then a filename, then one or more options or no options.

- A pipe separates mutually exclusive items. Often the set of items is enclosed in braces ({}), which you are **not** intended to type when you type one of the items. For example:

  ```
  { username | groupname }
  ```

  means either a user name or a group name.

- Sometimes a command does not fit on a single line in your window, or on the page of this book. In both cases, the same technique is used: a backslash (\) at the end of a line of code indicates that the command continues on the following line. The use of backslashes in the book does not necessarily indicate that you need backslashes in the same places.

- Pathnames used in the examples in this book usually appear in POSIX format (drivename:/directory/... /filename) and the special formats required by Tcl.

  **Note:** POSIX style pathnames are standard in UNIX and are generally valid in the Windows operating system (except for DOS boxes). For an explanation, see Strings Containing Special Characters (see page 26).

# eTrust SSO Scripts

When application login is controlled by eTrust SSO, the end user does not open the application directly, even though it appears this way to the user.

The end user performs a standard application startup. In Windows 2000, XP, and 2003, the user selects an application from the eTrust SSO submenu of the Start menu (or using one of the operating system's other options for running a program).

To the user it appears that the application is invoked directly. What actually happens is that the SSO Client has been invoked.

The SSO Client carries out a number of steps, including checking user authorization. If access is allowed, the SSO Client invokes the application requested and performs the login process according to an application-specific script (also called a logon script).

eTrust SSO requires an application-specific script for each application under eTrust SSO control. The script is an ASCII file. The scripting language for eTrust SSO is a specially extended version of the Tcl language, and is described in greater detail later in this guide.

The primary task of scripts is to provide the instructions for logging a user into a specific application. In addition, a script may contain instructions for other tasks associated with the login process, such as changing a password.

Normally, scripts should be application-specific, not user-specific. To enable application-specific scripts to serve various users, eTrust SSO maintains separate login variables for different authorized users. The scripts refer to user login variables for individual login name and password and other data that may be necessary.

# How Scripts Are Activated

The method by which scripts are activated depends on the specific OS interface. For example, the following is the process for Windows 2000, XP, and 2003:

1. The user selects an application from the eTrust SSO submenu of the Start menu. The SSO Client allows the use of additional features of the Windows interface, including shortcut application icons on the desktop and application icons in an application window (a window displaying eTrust SSO start menu folder).
2. The SSO Client on the user's workstation sends a login request for the specific application to the SSO Server.
3. The SSO Server responds by sending back to the SSO Client the appropriate script from eTrust SSO script storage, and login variables from the database used by eTrust SSO.
4. After the appropriate script and login variables are received at the user's workstation, the SSO Client plugs the variables into the script and runs the script to log the user into the application.

# Script Storage

The eTrust SSO database contains a record for each application and the record includes the name of the application's script file. The scripts themselves are not part of the eTrust SSO database. By default scripts are stored on the SSO Server in the following location:

```
c:\Program Files\CA\eTrust SSO\Server\Scripts
```

We recommend that you store script files together in a directory on the SSO Server host. However, sometimes you might want to store scripts elsewhere. For example, if a script is being tested and is not ready for general use, you might want to store it on an end user's workstation.

While scripts are being written, they should be stored at the script writer's workstation, in the directory that contains the SSO Client files.

**More information:**

Beginning to Write Scripts (see page 81)

# Script Authoring

The security or system administrator in charge of eTrust SSO should prepare the scripts. In practice, programmers may write the scripts under the administrator's supervision.

A script must conform exactly to a specific application as it is implemented on the user's system. Therefore, the person writing eTrust SSO scripts should work together with an application administrator who has a detailed knowledge of the applications' login processes.

# Script Contents

The primary role of the script is to provide the instructions to automate the login process. Automating the login process usually involves the following steps:

- Starting the application or the application interface
- Waiting for the user ID prompt
- Responding correctly to the user ID prompt
- Waiting for the password prompt
- Responding correctly to the password prompt
- Completing the login process

The script may contain instructions for additional tasks related to sign-on, or even unrelated tasks that should take place at the time of sign-on. For example:

- A script could provide instructions for changing the user's password according to a variable sent from the SSO Server. This applies only to applications with password-based logins.
- If users logins are normally followed by a few standard initial steps within the application, the script can instruct the application to perform these steps automatically.

After all the instructions in the script have been executed, the application continues to run with no further input from eTrust SSO.

# Chapter 2: Tcl Language Basics

This section contains the following topics:



## Basic Tcl Language

This chapter explains basics of Tcl (Tool command language) that are relevant to eTrust SSO scripts, primarily:

- Commands and arguments
- Syntax
- Variables and variable substitution
- Strings
- Expressions
- Control structures
- Procedures

It is based on material from the publicly available slide sequence "An Introduction to Tcl Scripting," by the creator of Tcl, Dr. John Ousterhout. For Tcl news, documentation, and resources, see Tcl Developers Xchange at: http://www.tcl.tk.

# Introduction to Tcl

Tcl (rhymes with nickel) is a scripting language that gives you the use of variables, conditions, loops, procedures, and other common programming devices with a minimum of complexity. Tcl is an interpreted language. Its control structures are implemented as commands, not as special configurations of syntax.

Tcl is considered an easy language to extend. eTrust SSO uses a specially extended version of Tcl for many of its operations. Installing eTrust SSO also installs Tcl and the eTrust SSO extensions to Tcl.

## Commands and Arguments

A Tcl command consists of a command name plus any necessary arguments. The arguments appear after the command name, separated from the command name and from one another by one or more spaces or tabs.

For example, the set command assigns a value to a variable. Its syntax is:

```
set variable value
```

Therefore, to assign the value 5 to the variable a:

```
set a 5
```

A Tcl command generally returns a result string, which is known as a **return value**. The set command, for example, returns the value that it has assigned.

Following the execution of set a 5, a Tcl interpreter returns 5.

In many cases, especially with regard to eTrust SSO extensions, the return value may be unimportant or already known, as in the above example. When significant, it may be kept available, for example by setting a variable.

### Scripts

A Tcl script consists of a sequence of commands, separated from each other by new lines or by semicolons.

For example, the following script assigns values to two variables and then evaluates and returns their sum:

```
set a 5
set b 7
expr $a+$b
```

When properly delimited, a script can be an argument in a command.

## eTrust SSO Extensions to Tcl

An eTrust SSO extension is, from Tcl's point of view, nothing but another Tcl command.

All eTrust SSO extensions have the following syntax:

sso *extension-name extension-arguments*

For Tcl, the command name is sso, and the *extension name* and *extension arguments* that follow sso are the command arguments. If you keep this in mind, it can help solve problems of Tcl syntax.

The arguments for eTrust SSO extensions (that is, the arguments that follow the extension name) have the format:

*-keyName keyValue*

For example, you can write an argument that passes a time parameter as:

```
-time 20sec
```

**Note:** The format is mandatory for eTrust SSO extensions, but **not** for native Tcl commands. Arguments that begin with a "-" sign are options (switches), which are discussed in detail throughout this guide.

The examples that follow describe two eTrust SSO extensions to Tcl: sso msgbox and sso inputbox. A summary description of these extensions is presented here to show how they work.

## msgbox

The msgbox extension displays a Windows message box at the user's workstation. In its most basic form, without using any icon or button set options, the msgbox extension is written as follows:

```
sso msgbox -msg messageText
```

The msgbox extension has an -icon option that allows the script developer to display the box with one of three severity icons to indicate that the message is informative, a warning, or an error message. When this option is used, the extension syntax is:

```
sso msgbox -icon info|warn|error -msg messageText
```

The following command shows a message box with an error icon:

```
sso msgbox -icon error -msg "Fatal error!"
```

If the -icon option is not used, no icon is displayed. Other options set the text of the message box title and display a particular button set for the end user's response. These options are discussed later in this guide.

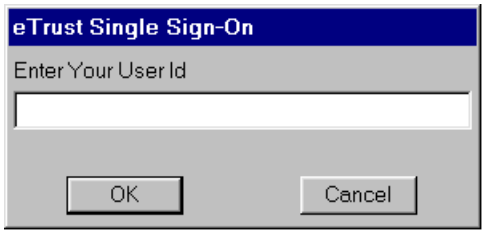
## inputbox

The inputbox extension displays a Windows dialog box at the user's workstation, and receives input from the user as a response. The extension has a mandatory -prompt *promptText* argument that sets a text string that is displayed in the dialog box above the input field. If no value is specified, then no text will be displayed.

Here is an example of the inputbox extension:

```
sso inputbox -prompt "Enter Your User ID"
```

This command displays the input box shown below and returns the user's response as the return value:




The extension also takes optional arguments for setting the dialog title and for limiting input to numerals. These options are discussed later in this guide.

# How Tcl Handles Arguments

Tcl uses **quoting by default**; that is to say, arguments are treated as strings except in special cases. In this respect, Tcl differs from such languages as C:

- In C, an expression is usually evaluated when a program runs. If you write x = 4; y = x + 10, then y is assigned the value 14.
- In Tcl, if you write set x 4; set y x+10, y receives the string value x+10 because Tcl was not explicitly instructed to evaluate x+10.

However, Tcl can handle arguments in other ways, in addition to handling them as literal strings. For example:

- The Tcl expr command treats a string as an expression and returns the expression's value. For example, expr 24/3.2 returns the value 7.5.
- When you enclose a string argument within square brackets, Tcl evaluates it as a command (that is, Tcl executes it and returns the result). For example, the set command can be used to hold the return value of a command for later use:

  ```
  set username [sso inputbox \
                    -prompt "Please enter your name"]
  ```

  In this example, the variable is set to the text string entered in the input box. This feature is called *command substitution* and is detailed in a later section in this chapter.

Some Tcl commands can receive optional arguments, which are known as **switches**. These arguments begin with a hyphen. For example:

```
sso msgbox -icon error -msg "Fatal error!"
```

# Variables and Variable Substitution

In Tcl, all variables are string variables. You declare a variable by using it. The name of a variable can comprise any combination of letters, digits, and underscores, but not spaces or special characters (non-alphanumeric characters). To refer to the value of a variable (rather than to the variable itself), use a $ sign as a prefix to the variable name.

The following table shows some examples of how variables are evaluated.

| Commands | Results | Notes |
|---|---|---|
| set b 66 | 66 | The variable b receives the value 66. |
| set a b | b | The variable a receives the value b (since b is not evaluated in this command). |

| Commands | Results | Notes |
|---|---|---|
| set a $b | 66 | The variable a receives the value that b has already received. Compare this example with the previous one. |
| set a $b+$b+$b | 66+66+66 | The variable a receives a string value in which the value of b appears three times. The string itself is not evaluated. |
| set a $b.3 | 66.3 | The variable a receives a string value that starts with the value of b. |
| set a $b4 | cannot read "b4": no such variable | The command is not executed because although b4 is a valid variable name, no variable with that name has been declared. In the expressions $b+$b+$b and $b.3, valid names of variables were delimited by non-alphanumeric characters that could not be part of a variable's name. |

**Note:** The first argument after the set command must be a valid variable name and not a variable value. If you prefix a $ sign to the first argument in the set command, making the first argument a variable value, the script will not work properly and the variable will not be assigned the value in the second argument. Furthermore, in some cases, you may not get an error message on this condition.

# Command Substitution

To use the result of a command as an argument of another command, put the command whose result is to be an argument in the appropriate position and put brackets around it. This is called **command substitution**. The string between the brackets is evaluated as a command and the command's result replaces the bracketed string. For example, the command:

```
set name [sso inputbox -prompt "What is your name?"]
```

executes the SSO inputbox command by displaying the message "What is your name?" and an input field. Next, the variable yourname receives the user's response.

The following table includes some examples of command substitution:

| Commands | Results | Notes |
|---|---|---|
| set b 8 | 8 | The variable b receives the value 8. |

| Commands | Results | Notes |
|---|---|---|
| expr $b+5 | 13 | The expression $b+5, which is 8+5, is evaluated. |
| set a [expr $b+2] | 10 | The variable b is evaluated, and then the expression $b+2, which is 8+2, is evaluated. The variable a, receives the value of the expression. |
| set a "b-3 is [expr $b-3]" | b-3 is 5 | The variable b is evaluated; then the expression $b-3, which is 8-3, is evaluated; and then the result is inserted into the string for a. The double quotes are needed because the string contains spaces. The variable a, receives the value of the expression. |
| set a [expr $b-3].3 | 5.3 | The variable a receives a one-word string value that starts with the result of [expr $b-3]. |

You can put brackets around a script of any length (not necessarily a single command), and the result of that script's last command is used as if the brackets contained a single command.

# How to Avoid Variable and Command Substitution

To indicate a string containing a $ sign or square brackets is to be treated as is, without any variable or command substitution, surround the string with braces. For example:

```
sso msgbox \
     -msg {Example of Tcl command: set a $b.[expr 1+1]}
```

This command displays a window with the message:

```
Example of Tcl command: set a $b.[expr 1+1]
```

as no variable or command substitution occured.

You must **always** leave a space in front of a left brace ( { ) that does not open a line and after a right brace ( } ) that does not end a line.

This example is correct because it has a space between the two braces on the first line:

```
if { $save_ssoerr == 100 } {
     sso msgbox -msg "Fatal Error!"
     exit
     }
```

This example is **not** correct because it does not have a space between the two braces on the first line. In this example, the script is aborted:

```
if { $save_ssoerr == 100 }{
     sso msgbox -msg "Fatal Error!"
     exit
     }
```

# Strings Containing Special Characters

Special (non-alphanumeric) characters can be handled as if, like alphanumeric characters, they were literals. The following techniques are used:

- To insert a comment, use the # character at the start of a line or after a semicolon; all the following characters up to the next new line or semicolon are treated as a comment. For example:

  ```
  # display a message box with "Program Running"
  sso msgbox -msg "Program Running"
  ```

- When you put double quotes (") around a string, spaces and semicolons in the string are treated as literals. For example:

  ```
  set x 5; set y 7
  sso msgbox -msg "x is $x; y is $y"
  ```

  displays a message box showing x is 5; y is 7, a string that includes spaces and a semicolon. Without double quotes as delimiters, the words between them are understood not as forming a single string, but rather as several arguments with the semicolon regarded as a command terminator.

  Double quotes do not prevent variable or command substitution. In the previous example, $x and $y are replaced by their values.

- To include a $ sign or square brackets in a string as is, without variable or command substitution, put braces around the string. However, braces following the expr command do not prevent the enclosed string from being evaluated. Braces can also be used, like double quotes, to delimit a string containing spaces or semicolons.

  For example:

  ```
  sso msgbox -msg {[expr $b*$c]}
  ```

  displays the message [expr $b*$c] and neither the variables nor the expression is evaluated.

  If a pair of braces is open, a new line that follows the opening brace does *not* end a command. For example:

  ```
  sso msgbox -msg {
  Welcome to SSO}
  ```

  is interpreted as a single command.

- Backslashes are escape characters. When a backslash is placed in front of an individual special character, the special character is treated like a regular alphanumeric character (\$ is treated like the literal $). For example:

  ```
  sso msgbox -msg string\ with\ \$,\ \\\ and\ space
  ```

  displays a message that includes a $ sign, a backslash, and spaces: string with $, \ and space. Without backslashes in front of blank spaces and special characters, the spaces and special characters would not be considered as elements of a single string.

  A backslash also serves as a continuation mark. When a backslash (\) is used at the end of a line of code it indicates that the command continues on the following line. No spaces are allowed after a backslash if it is intended as a continuation mark. This allows you to use a new line without ending a command. For example:

  ```
  sso msgbox -msg \
  "Welcome to SSO"
  ```

  These two lines are understood as a single command.

  **Note:** Because Tcl uses the backslash as an escape character, it cannot handle pathnames written in DOS style (that is, with single backslash separators). However, pathnames with single backlash separators within braces are valid, as are pathnames with double backslash separators:

  "C:\Notes\notes.exe" is an invalid pathname in Tcl.
  {C:\Notes\notes.exe} is valid.
  "C:\\Notes\\notes.exe" is valid.

When you use quotation marks, backslashes, and braces, to combine words and symbols into a single string, the string is handled as a single value. For example:

```
set a "two \
words"
sso msgbox -msg $a
```

displays two words in a message box.

# Operators and Their Precedence

Tcl uses the operators listed in the following table, which are similar to those in C. The numbers indicate precedence, with 1 indicating the highest precedence.

| Precedence | Operator | Description |
|---|---|---|
| 1 | - | Negation, for example -x is x times -1 |
| 1 | ! | Not, for example if x is 0 then !x is 1 |
| 1 | ~ | Bitwise complement |
| 2 | * | Multiplication |
| 2 | / | Division |
| 2 | % | Remainder or modulo, for example 17%5 is 2 |
| 3 | + | Addition |
| 3 | - | Subtraction |
| 4 | << | Bitwise left shift |
| 4 | >> | Bitwise right shift |
| 5 | < | Less than, for use in Boolean expressions |
| 5 | > | Greater than, for use in Boolean expressions |
| 5 | <= | Less than or equal to, for use in Boolean expressions |
| 5 | >= | Greater than or equal to, for use in Boolean expressions |
| 6 | == | Equals, for use in Boolean expressions |
| 6 | != | Does not equal, for use in Boolean expressions |
| 7 | & | Bitwise AND |
| 8 | ^ | Bitwise exclusive OR |
| 9 | \| | Bitwise inclusive OR |

| Precedence | Operator | Description |
|---|---|---|
| 10 | && | Logical AND |
| 11 | \|\| | Logical OR |
| 12 | ? and : | If nonzero then and else; for example:<br>a?b:c<br>means if a is nonzero then b, otherwise c. |

# Expressions

Tcl expressions can serve as arguments for the expr command and for other commands. They resemble expressions that C uses, particularly those for C integers and double (float) values. However, Tcl includes extra support for string operations.

As previously stated, command substitution and variable substitution can occur within expressions.

The following commands include some examples of expressions:

| Commands | Results | Notes |
|---|---|---|
| set b 5 | 5 | The variable b receives the value 5. |
| expr $b*4 – 3 | 17 | The expression $b*4, which is 5*4, is evaluated, then 3 is subtracted from that result. |
| expr $b <= 2 | 0 | First $b is evaluated and then the Boolean expression, which is 5<=2, is evaluated. Since the Boolean expression is false, the result returned is 0. |
| set a Bill<br>expr {$a < "Anne"} | Bill<br>0 | You can perform relational operations on text strings, according to their ASCII values. Note that you must use curly braces in the second expression so that $a < "Anne" is understood to be a single string. |
| expr 6 * cos(2*$b)<br>expr {$b * [fac 4]} | -5.0344<br>3<br>120 | You can use various other common mathematical functions. |

Use parentheses for clarity. For example, the second example should be written as follows:

```
expr ($b*4) – 3
```

# Conditional Control Structures

Control structures in Tcl are implemented as commands, not as special configurations of syntax. Tcl control structures may look like C, but syntactically they are Tcl commands taking Tcl scripts as arguments.

Tcl uses the if command and the switch command to control conditional branching. These commands are described in the following pages.

## if [elseif, else]

**Syntax**

The basic if command takes a Boolean expression and a script (one or more commands) as arguments. If the condition is true (if the Boolean expression is non-zero), the script is executed.

```
if BooleanExpression script
```

The if command can be followed with one or more elseif clauses. Each elseif clause consists of the word elseif, a Boolean expression, and a script. If the original condition is not true, then the script following the first true elseif clause is executed.

```
if BooleanExpression script\
     elseif BooleanExpression script \
     elseif BooleanExpression script ...
```

You can use an else clause to close an if command. The *else* clause consists of the word else and a script. If neither the if condition nor any of the *elseif* clauses (if present) is true, then the else script is executed.

```
if BooleanExpression script\
     elseif BooleanExpression script \
     elseif BooleanExpression script \
     else script
```

The if command, regardless of the number of clauses it contains, remains a single command and therefore cannot contain any command terminators (semicolons or new lines). In order to type an if command on several lines (for the sake of readability), preface each new line with a backslash or enclose the new line in braces.

It is good practice to use braces around the expressions and scripts. However, **do not** begin a line with a left brace, because this indicates the start of a new command. When a string appears as an if argument, it must be placed in quotation marks.

### Example

This example shows a script using if, elseif, and else to display different messages according to the value of an error variable (_SSOERR) which is expected to be between 0 and 100:

```
set save_ssoerr $_SSOERR
if { $save_ssoerr == 100 } {
     sso msgbox -msg "Fatal Error!"
     exit
} elseif { $save_ssoerr > 0 && $save_ssoerr < 100 } {
     sso msgbox -msg "Problem encountered"
     exit
} elseif { $save_ssoerr == 0 } {
     sso msgbox -msg "Successful completion"
} else {
     sso msgbox -msg "Unexpected SSOERR value:$_SSOERR"
     exit
}
```

The style of this example makes use of the manner in which the Tcl interpreter parses commands. As previously noted, new lines are command terminators, but not when they are in a group delimited by braces. Placing an opening curly brace at the end of a line takes advantage of this property to extend the if command over multiple lines.

## switch

### Syntax

The switch command, like the case command in some languages, is a compact way of expressing an if that has many elseif clauses. The switch command executes one of a number of scripts (consisting of one or more commands) according to the value of a variable.

The command's first argument is a variable value to be matched. Following this argument is any number of compound arguments. Each compound argument consists of a value or pattern plus a script.

When the script is run, the value or pattern of each compound argument is compared with the value of the first argument of the switch command. When a match is found, the script of that compound argument is executed and the script's return value is then returned by the switch command.

A default compound argument can be placed after all the other compound arguments. This consists of the value default and a script. If no match is found with any of the previous arguments, then the default script is executed.

Scripts are easier to understand if you enclose individual compound arguments and the whole set of compound arguments in braces.

**Example**

This example also evaluates error values, but uses switch instead of if:

```
set save_ssoerr $_SSOERR
switch $save_ssoerr {
     100 {
       sso msgbox -msg "Fatal error!"
       exit
       }
     80 {
       sso msgbox -msg "Invalid parameter encountered"
       exit
       }
     20 {
       sso msgbox -msg "Missing argument"
       exit
       }
     0 {
       sso msgbox -msg "Successful completion"
       }
   default {
      sso msgbox -msg "Unexpected SSOERR value: $_SSOERR"
      exit
      }
}
```

**Note:** The switch command uses glob-style matching as default but can accept -exact or -glob itself as an optional argument.

**More information:**

Specifying Windows (see page 96)

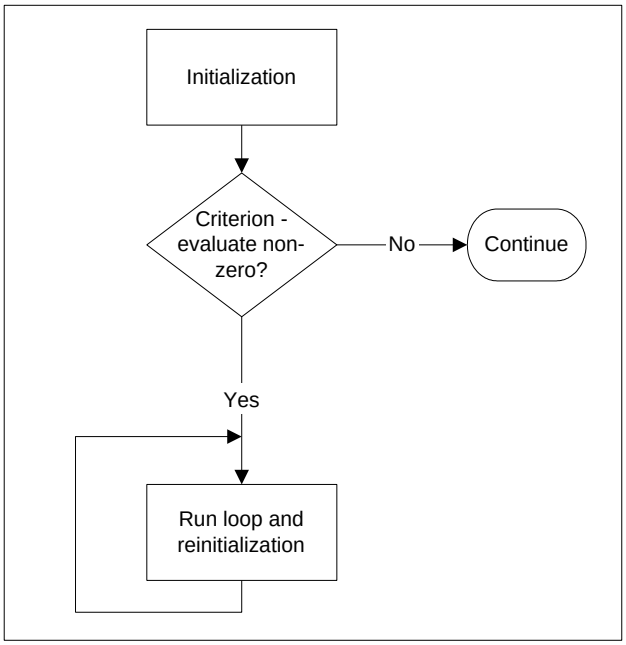
# Loop Control Structures

Like conditions, loops in Tcl are implemented as commands, not as special configurations of syntax. Like many other languages, Tcl initiates and controls looping with for, while, and incr (increment) commands.

## for

### Syntax

The for command initiates zero or more iterations of a loop. It takes four arguments: an initialization command or script, a looping criterion, a reinitialization script to be run at the end of each iteration, and a command/script that runs each time the loop iterates.



### Example

The following example displays a message box with the text "testing...1...2...3...":

```
set txt "testing..."
for {set i 1} {$i <= 3} {incr i 1} {
     append txt " $i..."
}
sso msgbox -msg "$txt"
```

where:

**{set i 1}**

Initializes the loop counter.

**{$i <=3}**

Is the looping criterion.

**{incr i 1}**

Increments the loop counter.

**append**

Is a Tcl command that takes a variable name as its first argument and concatenates its remaining arguments onto the current value of the named variable.

## while

### Syntax

The while command repeats a specified script as long as a specified expression remains non-zero. As arguments, it takes the expression and the script. The command checks the control expression before each execution of the script.

### Example

The following example displays an input box continuously until the user enters a non-empty value in the input field:

```
set user_name ""
while { $user_name == "" } {
    set user_name [sso inputbox -prompt "What is your name?"]
 }
sso msgbox -msg "Hello, $user_name"
```

## incr

**Syntax**

The incr command is used with for and while to update loop counters. It takes two arguments: the first is a variable; the second is an increment value. The increment can be negative. If you omit the second argument, a positive value of 1 is used to increment the variable.

**Examples**

To increment the variable *counter* by 1:

```
incr counter
```

To decrement the variable *remainder* by 2:

```
incr remainder -2
```

# Procedures

Procedures in Tcl are defined by using the proc command. Once defined, a procedure is invoked by name exactly like any built-in Tcl command.

## proc

**Syntax**

The proc command defines a procedure. It takes three arguments:

- The procedure's name
- A list of the procedure's arguments, or an empty string ("") if the procedure takes no arguments
- A script for the operations the procedure carries out.

For example:

```
proc display_error { {errmsg "Error occurred!"} } {
sso msgbox -icon error -msg "$errmsg"
}
```

creates a procedure named *display_error* that receives a variable named errmsg as an argument and displays it in a window. An example of a command that invokes the new procedure:

```
display_error "Invalid characters were entered"
```

Procedure arguments can be inserted whenever a procedure is run in a script. In addition, you can give procedures default arguments when the procedure is defined, and these same default arguments are used whenever the script calls the procedure.

A procedure can have any number of arguments. Following is an example of a procedure that has three arguments:

```
proc display_sum {a b c} {
    set sum [expr $a+$b+$c]
    sso msgbox -msg "$a+$b+$c=$sum"
}
```

A procedure may have no arguments at all:

```
proc end_script {} {
   sso msgbox -msg "Thank you for using eTrust SSO"
   exit
}
```

## Default Argument Value

When you specify an argument to the proc command, you can also specify a default value by including the default value immediately after the argument variable. The default value will be assigned if no value is specified for the argument when the procedure is invoked.

For example:

```
proc display_error { {errmsg "Error occurred!"} }
sso msgbox -icon error -msg "$errmsg"
}
```

If the procedure display_error is invoked without arguments:

```
display_error
```

then the message “Error occurred!” is displayed. If an argument is specified:

```
display_error "Invalid characters were entered"
```

then the argument declared is displayed as the error message; in this case “Invalid characters were entered”.

If a procedure has more than one argument and you want to specify default values for some or all of the arguments, enclose each argument-value pair in braces. Always list any arguments that have defaults after any arguments that do not.

When you call a procedure, you have to declare the values of all the arguments without defaults in the order in which they appear in the proc command. For example:

```
proc display_error {errmsg {severity error} } {
sso msgbox -icon $severity -msg "$errmsg"
}
display_error stop
```

In the previous example, the first argument (errmsg) has no default value and the second argument (severity) has a default value.

**Note:** It pays to carefully examine the use of multiple defaults in a procedure.

A Tcl interpreter activates defaults according to the position of their arguments as given in the procedure definition. Therefore, when an argument is defaulted, there is no way to give a value to any of the arguments that follow it in the argument list.

**More information:**

Procedures with an Indeterminate Number of Arguments (see page 267)

## return

To return a value from a procedure, use the *return* command. Note that, as in other languages, the return command terminates the procedure execution and returns control to the procedure caller. For example:

```
proc sub1 {x} {
    set x_minus_1 [expr $x-1]
    return $x_minus_1
}
```

You can invoke the above procedure like this:

```
set msg_len 80
set msg_len [sub1 $msg_len]
```

## global

In general, a procedure recognizes no variables from outside itself. Arguments are one exception. Some other exceptions can involve the use of the global command and the uplevel command.

The arguments of the global command are the names of variables from outside the procedure. For example:

```
global _LOGINNAME _PASSWORD
```

Once you have used the global command, those variables are understood to be the variables declared in the script outside the procedure.

Other than arguments, all the variables of a procedure are valueless each time the procedure starts executing. If you need to retain a value between executions of the procedure (the way a static variable retains its value in C), you can use a uniquely named global variable and use the global command early in the procedure.

**More information:**

Advanced Tcl commands (see page 264)

# String Manipulations

Tcl provides several commands that manipulate, examine, and modify strings. This section describes the string command.

## string

**Syntax**

The string command is a family of operations, all beginning with the word string followed by a command modifier. Several of them refer to the position-numbers of characters in a string, with the first position in the string being number 0.

| Command Modifier | Description |
|---|---|
| compare | string compare *string1 string2*<br>Returns:<br>-1 if string1 < string2<br>0 if string1 == string2<br>1 if string1 > string2 |
| match | string match *glob-pattern string2*<br>Returns:<br>1 if there is a glob-style match between *glob-pattern* and *string2*.<br>0 if there is not. |

| Command Modifier | Description |
|---|---|
| first | string first *string1 string2*<br>Returns the position-number where the first occurrence of *string1* starts in *string2*.<br>If string1 does not appear in *string2*, then string first returns -1. |
| last | string last *string1 string2*<br>Returns the position-number where the last occurrence of *string1* starts in *string2*.<br>If string1 does not appear in *string2*, then string last returns -1. |
| index | string index *string1 position*<br>Returns the character from *string1* that corresponds to the specified *position* number. |
| range | string range string1 position1 position2<br>Returns the characters from *string1* that start at position number *position1* and end at position number *position2*. To return characters up to the end of the string, specify **end** for *position2*. |
| tolower | string tolower *string1*<br>Returns *string1* converted to all-lowercase characters. |
| toupper | string toupper *string1*<br>Returns *string1* converted to all-uppercase characters. |
| trimleft | string trimleft *string1 [string2]*<br>Returns:<br>*string1* stripped of specified leading characters given by *string2*.<br>If you do not use *string2*, string trimleft strips away leading spaces. |
| *trimright* | string trimright *string1 [string2]*<br>Returns:<br>string1 stripped of specified trailing characters given by *string2*.<br>If you do not use string2, string trimright strips away trailing spaces. |

# Tcl Utility Commands

You can use Tcl's cd and pwd commands to control DOS and Windows directories.

## cd

The cd command changes the client's working directory. For example:

```
cd "c:\\my_dir"
```

changes the working directory to my_dir. This command can be used to find the right .dll file for a Windows program such as a 3270 emulator.

## pwd

The pwd command returns the working directory. For example:

```
set cur_dir [pwd]
 sso msgbox -msg "$cur_dir"
```

displays a message box with the current directory name.

**Note:** The SSO extension selecttree allows you to select directories and files in Windows 2000, XP, and 2003 using Explorer.

**More information:**

Windows Extensions (see page 51)

# Tcl Syntax Summary

These rules and examples review the main concepts of Tcl syntax:

- A command is a sequence of words, with words separated by spaces or tabs. It begins with a command word and may be followed by one or more arguments.
- A script is a sequence of commands, with new lines or semicolons separating the commands.
- $*var1* represents the value of the variable named *var1*.
- [*command1*] represents the result of the command *command1*.

- *command1 a b* means *command1* receives two arguments: *a* and *b*.
- *command1* "*a b*" means *command1* takes one single argument: *a b*.
- set a 1
  set b 2
  command1 "$a $b"

  means *command1* takes a single argument*:* 1 2.
- "*string1*" means string1 regardless of blanks, tabs, and semicolons.

  {*string1*} means *string1* literally, regardless of all special characters.
- \ means that the character following the backslash is to be taken literally, even if it is a special character.
- # means that the following characters up to the next new line or semicolon are a comment

# Chapter 3: Learning About Extensions

This section contains the following topics:



## The eTrust SSO Extensions

eTrust SSO extensions are classified according to the type of operations they perform and the elements they manipulate. They all have similar formats and use arguments in the same way.

**Important!** *In some of the examples in this book, pathnames are split over two lines to fit the page. Pathnames in actual scripts should be written on one line and not divided.*

### Format and Arguments

All the SSO extension commands begin with the word sso, and have the following general syntax:

sso *extension-name* [*extension-arguments*]

SSO extension arguments are expressed in the -key *keyValue* format (except for a few exceptions that are documented). These arguments, which can be either mandatory or optional, can appear in any order, as long as the key value follows after the key and before the key of the next argument.

The argument key is written as a single word preceded by a hyphen. The key value is a Tcl string. If necessary, the key value itself can begin with a hyphen (although this should be avoided for the sake of good form and readability). Following are examples of valid arguments:

```
-sync y
-offsx 30
-msg "Fatal error!"
-password $_PASSWORD
-labelglob "*Notes*"
-title "Untitled"
-titleglob "Untitled*"
-titlexact "Untitled - Notepad"
-class "Notepad"
```

You can use the -target key to specify two criteria that must be used together when you search for a target window. You can use both "-title|-titleglob|-titlexact" and "-class" to search for a single window. The target window must then meet both criteria. For example:

```
-target {-title windowtitle -class windowclass}
```

## Completion Code

Most SSO extensions return a completion code to signify the status of the action. A completion code value of zero indicates that the extension performed its intended task. Other code values, which are positive values, indicate failure and identify problems.

The completion code is returned in the variable _SSOERR. Normally, the return of a non-zero completion code stops the script execution, unless the _ERRORMODE variable is set to resume or msg mode.

In *resume* mode, when the SSO extension fails, eTrust SSO sets _SSOERR to a non-zero value and resumes execution of the script. In *msg* mode, eTrust SSO displays an error message box, sets the _SSOERR variable to a non-zero value, and resumes execution of the script.

**Note:** The _SSOERR variable is set by all eTrust SSO extensions except the msgbox and sleep extensions.

**More information:**

Error Handling (see page 84)
Completion Code Table (see page 223)

## Return Value

Most SSO extensions have a return value, which is the result of the extension's execution. For example, the return value might be the title of a window or the ID of a button selected by the user. However, not all return values are needed for practical scripting.

The return value is different than the completion code. The return value returns the result of the Tcl command and is produced by most Tcl commands, whether basic Tcl or SSO extensions. The completion code, which is unique to SSO extensions, tells you whether the SSO extension succeeded or failed in performing its task and is returned within the _SSOERR variable.

The following example demonstrates this difference:

```
set _ERRORMODE resume
     set userid [sso inputbox -prompt "Enter your User ID"]

if { $_SSOERR != 0 }{
   sso msgbox -msg "The inputbox extension failed with \
     a completion code of $_SSOERR"
exit
} else {
   sso msgbox -msg "The inputbox extension succeeded. \
     Its return value was $userid"
}
```

Not all values returned by Tcl commands are relevant to SSO scripts. When the return value of an extension is not specified in the extension's description, you should not refer to it.

## Extension Types

There are six categories of SSO extensions:

**General extensions**

Used to perform general operations, most of which are environment-independent.

**Windows extensions**

Used to access and manipulate visual elements in the Microsoft Windows environment.

**Network extensions**

Used to issue commands to the network service providers, including Windows NT and Novell NetWare servers.

**html_ (browser) extensions**

Used to provide support for Microsoft's Internet Explorer 4.*x*. Because Internet Explorer *4.x* uses elements that are not Windows-standard, scripts for logging into applications via this browser must use html_ extensions.

**Login extensions**

Used to get login services from the SSO Server.

**hllapi_ extensions**

Used to provide specialized support for HLLAPI emulations for mainframes and AS/400s in situations where general and Windows extensions do not provide all the functions SSO needs to automate login.

**More information:**

eTrust SSO Extensions for HLLAPI (see page 235)

# General Extensions

There are two types of general extensions: interactive extensions and control extensions.

## Interactive Extensions

Through the interactive extensions, eTrust SSO enables a script to communicate with the end user. The communication is established by displaying messages, asking questions, requesting input, and accepting user responses.

By using interactive extensions, the script writer can specify message boxes and dialog boxes that are displayed on the user's workstation. These extensions control the following elements of message and dialog boxes:

- *Title* (caption of a message or dialog box)
- Text message
- *Text box* (input field for the user to enter text in; also referred to as an edit box)
- *Labels* (captions for text boxes)
- *Command buttons* (for user response)

## msgbox

The *msgbox* extension is used to display a message box (a window containing a message) to pass an instruction or notice to the user, or to ask the user a question that can be answered with a standard answer by means of a command button. Available sets of command buttons are OK, Yes-No-Cancel, and Abort-Retry-Ignore.

The msgbox arguments include:

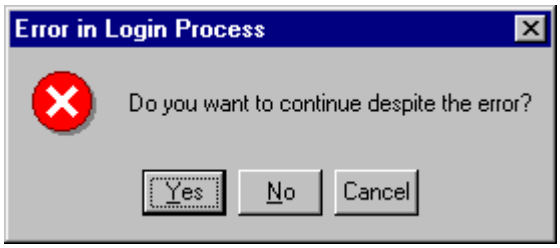
- The message box title
- The message box text
- Message severity icon (informational, warning, error, or no specification)
- Command button sets at the bottom of the message box

  You can choose between several sets of buttons; the default set is an OK button. You can examine the return value of msgbox to determine which button was selected (clicked) by the user.

Here is an example of a msgbox command with its arguments:

```
set user_response \
   [sso msgbox -icon error  -buttons yesnocancel \
    -title "Error in Login Process" \
    -msg "Do you want to continue despite the error?"]
```

It produces the following message box:



A script usually follows a msgbox command like this one with a Tcl branching command that uses the user's response. For example:

```
switch $user_response {
    yes {
       # continue execution
       }
    no {
       # stop execution
       exit
       }
    cancel {
      # repeat the last operation
       ...
       }
    default {
       sso msgbox -msg "Unexpected error"
       exit
       }
}
```

## askyesno

The *askyesno* extension is a simplified variant of the msgbox extension. It provides only one set of buttons (Yes-No) and does not support the severity icon option argument.

Here is an example of the askyesno extension and an associated Tcl branching command:

```
set selection [sso askyesno -prompt "Change password?"]
 switch $selection {
    yes {
      # change password
      ...
      }
    no {
      # do not change password
      ...
      }
    default {
      sso msgbox -msg "Unexpected error"
      exit
      }
}
```

### statusbox

The s*tatusbox* extension is another variant of the msgbox extension. It displays a box with text. Unlike the msgbox extension, which suspends script execution until the user presses the OK button, the script continues to execute while the statusbox extension displays a message.

### inputbox

The *inputbox* extension is used to get the user to input a textual or numerical response in a text box. The extension returns the user's response in a return value. You can put a default answer in the input field. It is also possible to specify that the user type in only numeric text.

Here is a simple example of an inputbox extension that shows a dialog with a text box and a text box label and assigns the user's response to the variable answer:

```
set answer [sso inputbox -prompt "Please enter your name"]
```

### pwdbox

The *pwdbox* extension is similar to the inputbox extension. The main difference between them is that when the user types data in the input field of the pwdbox extension, the written text is not displayed as it is typed in, but asterisks appear instead of the typed characters. This allows the user to enter a password without having the password appear on the screen.

You should use the -retype y|n option to duplicate the input field so that the user has to verify the input, especially when you want the user to type in a new password and you want to prevent a typing error.

Here is an example of a pwdbox command and the dialog it displays:

```
sso pwdbox -retype y
```


## Control Extensions

These extensions invoke programs and suspend script operations.

## run

The *run* extension invokes a program. The run extension with the *-getpid y* option makes the program the *current program* by assigning an identifier for the program to the variable _PID. Other extensions often need to identify the current program in order to carry out their operations.

If you use the *-mainwin y* option with the "sso run" extension, the extension searches for the main window opened by the program being run. If found, the window becomes "current" to start receiving user input. The extension variables such as _WINDOW, _WIN_TITLE, _WIN_INFO are updated to match this new "current" window.

Here is a simple example of the run extension that invokes telnet:

```
sso run -path "telnet"
```

The example assumes that the telnet program is in the search order for programs (PATH). However, if this were not so, you could specify the full path for the program. For example:

```
sso run -path "C:\\winnt\\system32\\telnet.exe"
```

**Note:** Tcl does not accept pathnames with single backslash separators. eTrust SSO scripts running under Windows generally accept POSIX style pathname (forward slash separators) except in DOS boxes. Use the following format in most cases:

```
"drivename:/directory/directory/... /filename"
```

The *-args* option specifies arguments for the executed program and the *-dir* option specifies the working directory for program execution.

For example, you could start telnet and also give the program initial arguments:

```
sso run -dir "C:/unix/work" -args "unix1" -path "telnet"
```

The *-oneinstance y* switch is used to indicate that a new instance of the program should not be started up if an instance is already running in the system.

For example, to run telnet only if no telnet session is already running, use:

```
sso run -oneinstance y -path "telnet"
```

If we want to open a session of Notepad, and want to avoid confusion with other existing Notepad sessions, we can use the *-mainwin y* option.

For example, instead of using three commands to launch a program, then find the window (this might fail if more than one window is found), and simulate a user typing action:

```
sso run -path Notepad
sso window -title "Untitled - Notepad"
sso type -text "sample text type into the Notepad window"
```

You can do it in a simpler and more robust way, such as launch program with -mainwin y and simulate a user typing action:

```
sso run -path Notepad -mainwin y
sso type -text "sample text to type into the Notepad window"
```

### sleep

The *sleep* extension is used to suspend script execution for a specified amount of time. The sleep extension is useful when you want execution of the script to wait for some external event to happen, but you cannot, or do not want to, use the SSO timeout mechanism. You can specify the amount of time you want to sleep in seconds (default) or in milliseconds.

All the examples below suspend script execution for 5 seconds:

```
sso sleep -time 5
sso sleep -time 5s
sso sleep -time 5000ms
sso sleep -time 5000msec
```

## Windows Extensions

The eTrust SSO Windows extensions enable the script to control the visual elements of the Microsoft Windows environment. Among their various functions, they allow eTrust SSO to select a specific window, to type text, to push buttons, and to obtain the contents of designated fields. There are three groups of Windows extensions: those that operate at the level of windows, those that operate on fields, and those that operate at the level of text.

## Window-Level Extensions

For purposes of eTrust SSO scripting, it is important to differentiate between two classes of windows on the desktop: top-level windows and subwindows.

Top-level windows include:

- Windows that contain icons. These windows are referred to in Microsoft documentation as "open windows."
- Application windows, which may themselves contain other windows.
- Message boxes, dialog boxes, and property sheets.

Subwindows are generally windows created by an application and are referred to by Microsoft as "document windows." Some of the Windows extensions operate only on top-level windows and some only on subwindows.

### Specifying Windows

Some Windows extensions need you to specify the window that the extension will work on. eTrust SSO searches for this window from among all the top-level windows (as opposed to the subwindows) that currently exist. If the specified search criteria are matched by more than one window, eTrust SSO chooses the window that was created by the *current program* (the program that was executed by the last run extension), if such a program is active. If eTrust SSO fails to find the specified window, or if it fails to choose a window among all the matching ones, eTrust SSO returns an error value.

You can specify the search criteria for windows in several ways:

- The simplest way is to specify all or part of the text that appears as the title (the caption of the title bar) of the required window. For example, to search for the following window:

Use one of the following commands:

```
sso window -title "Control Panel"
sso window -title "Control"
```

However, even if your caption specification is exact, ambiguity problems can still occur. In the example, you received the Control Panel window. However, if there had been another open window with the title "Control Panel Help," you might have received that window. Using the -titlexact switch prevents ambiguity by indicating that the specified caption is the exact title of a window, rather than only the first part of the window's title. This is advisable when there is a possibility that the title you are searching for is also a partial title of another window.

Using the above example, type the following to avoid any ambiguity:

```
sso window -titlexact "Control Panel"
```

- You can also make the search more specific by including more than one criteria. To do this you should use the -target option. You can combine any of "-title|-titleglob|-titlexact" with "-class". The following example will only find a Notepad document that is called "Word". It will not find any other window called "Word".

  ```
  sso window -target {-class "Notepad" -title "Word.txt - Notepad"}
  ```

  If any criteria within the -target curly brackets are not met, the error message "window not found" occurs.

  When the "-title|-titleglob|-titlexact" and "-class" are used outside the "-target" option, each occurrence of them is treated as a potential search target and search will be conducted in the order they are specified in the command. The first window found will be returned.

  Within the "-target" option, you can use both "-title|-titleglob|-titlexact" and "-class" to search for a single window. The target window then has to meet both options. When the same option is doubled, the last one will be used.

- A more inclusive way of specifying the window that you want is to provide a pattern of the title text to be found. This pattern is composed in a standard format, called *globbing,* and is indicated by the *-titleglob* option. This is useful when you do not know exactly how the window title will appear, as happens when the title is a dynamic string of text that is built of variable data.

  **Note:** What is written in the title may be different if the application is started in different ways.

  The following are a few globbing examples, which match the Control Panel window:

  ```
  sso window -titleglob "Cont*"
  sso window -titleglob "Cont??? Panel"
  sso window -titleglob "Cont??? P*"
  sso window -titleglob {[ABCDEFGH]ontrol Panel}
  sso window -titleglob {[A-H]ontrol Panel}
  ```

- Another way of specifying the window you want is to provide the class of the window along with the window's title (the title can be plain text or a pattern). The window class can be either a Windows pre-defined class, such as DialogBox, or an application-specific class. This notation is useful when there are two or more windows with identical captions.

  If you do not know the class of the specific window, you can determine it by using an appropriate utility, such as Spy (which is bundled with Microsoft Visual C++). Here is an example using a window class criterion:

  ```
  sso window -class CabinetWClass
  ```

## The Target Window

Many extensions do not specify the window in which they function, but rather operate within the *target window,* which has already been designated by the action of a previous extension.

Field-level extensions (see Field-Level Extensions later in this chapter) work in a *target window.* For example, sso click -label OK will look for the OK command button inside the target window, and then push it.

The target window may be one of the following:

**The current window**

This is the window that was found by the most recent window or subwindow extension. The variable _WINDOW holds its window handle.

**The active window**

This is the window that is presently on the screen and operating. A window can be activated by a window extension. An example of an active window that is not a current window is a login dialog box displayed by a program.

Which of the two above windows will be the target window depends on the setting of the _TARGET_WIN variable, which can be current or active.

When a script begins executing, the target window is the active window (the default value of the _TARGET_WIN variable is active). In many cases, you do not need to change the target window. There are cases, however, when you want the script to control the choice of the target window. If there is a possibility that an unexpected window could open and become the active window, then you should set _TARGET_WIN to current and use the window extension to set the current window. For example, to make sure that the target window will be the Control Panel window:

```
set _TARGET_WIN current
sso window -title "Control Panel"
...
```

## window

The *window* extension can be used for several different functions:

- To set the current window, as explained above. For example:

  ```
  set _TARGET_WIN current
  sso window -title "Network Neighborhood"
  ```

  Makes "Network Neighborhood" the current window.

- To focus on a window (to make it the active window, as explained above). This is a default action of window. For example, to make the My Computer window the active window:

  ```
  sso window -title "My Computer"
  ```

  **Note:** The window extension makes the designated window both the active window and the current window. To run window without making the designated window the active window use the -waitfocus 0 option. For example:

  ```
  sso window -waitfocus 0 -title "My Computer"
  ```

  makes the "My Computer" window the current window without making it the active window.

- To suspend script execution either until a specific window appears or until one of several specified windows appears. The return value will indicate which specified window actually appeared. For example:

  ```
  set appeared_window [sso window \
      -title "Network Neighborhood" \
      -title "My Computer"]
  ```

  If none of the specified windows appears before the timeout value is reached, an error value is returned.

- To set a window's size, using the -size min, max, open or same options. For example:

  ```
  sso window -size min -title "Netscape Mail"
  ```

## subwindow

The *subwindow* extension is similar to the window extension, but it searches for a specified document window among the document windows of the target window, rather than for a top-level window. The main function of the subwindow extension is to manipulate a special kind of subwindow, created by Multiple Document Interface (MDI) applications such as Microsoft Word, Microsoft Visual C++, and some 3270 emulations. These special subwindows look very similar to top-level windows, but cannot be handled by the window extension.

You can use the subwindow extension to perform several functions:

- To set the current window, as explained above. For example:

```
set _TARGET_WIN current
sso window -title "Microsoft Word"
sso subwindow -title "Document1"
```

- To wait for a subwindow to appear, or wait for one of several subwindows to appear. The return value indicates which specified subwindow actually appeared. For example:

```
sso window -title "Microsoft Word"
set subwindow [sso subwindow \
    -title "Document1" \
    -title "Document2"]
```

  While SSO is waiting for one of the specified subwindows to appear, script execution is suspended. If none of the specified subwindows appears before the timeout value is reached, an error value is returned.

- By default, subwindow makes a document window the active window. If you don't want the specified window to be the active window, use the -waitfocus 0 option. For example, to make the Doc 2 document window the current window, but not the active window:

```
sso subwindow -waitfocus 0 -title "Doc 2"
```

- To set a document window's size, use -min, -max, -same and -open switches. For example:

```
sso subwindow -size min -title "Document1"
```

The standard window-specification format is used to specify the document window or windows on which the subwindow extension will work.

## wintitle

The *wintitle* extension returns the title (the title bar caption) of the current window or of the active window. You can use wintitle in the following two ways.

Use the -window active|current option to get the title of the current or the active window. For example:

```
sso window -title "Microsoft Word"
set caption [sso wintitle -window current]
sso msgbox -msg $caption
```

If you do not specify a switch, you will get the title of the target window, which can be either the current window or the active window, depending on the setting of the _TARGET_WIN variable. The following example is equivalent to the previous one:

```
set _TARGET_WIN current
sso window -titleglob "Microsoft W*"
set caption [sso wintitle]
 sso msgbox -msg $caption
```

## getmsgtext

Use the *getmsgtext* extension to get the message text that appears in a message box. eTrust SSO searches for the specified message box, and retrieves the text from the one it finds. Use the standard window-specification format, to specify the message box on which the getmsgtext extension works.

For the following window:


```
set text [sso getmsgtext -title "Warning!"]
```

sets the text variable to the text string Your password is about to expire.

### screensize

The *screensize* extension retrieves the dimensions of the display area of the local workstation and stores them in height and width variables. The script can use these variables to locate the cursor at the proper insertion point for text entry. For example, the following commands retrieve the screen size and display it in a message box:

```
sso screensize
sso msgbox -msg "Height is $_SCREEN_HEIGHT and width is $_SCREEN_WIDTH"
```

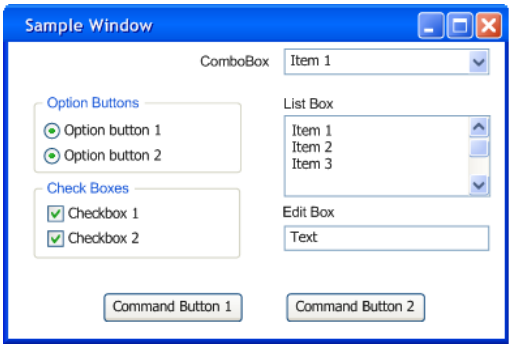
The *shutdown* extension will either shutdown or restart the computer. The Force option will automatically terminate all applications currently running.

## Field-Level Extensions

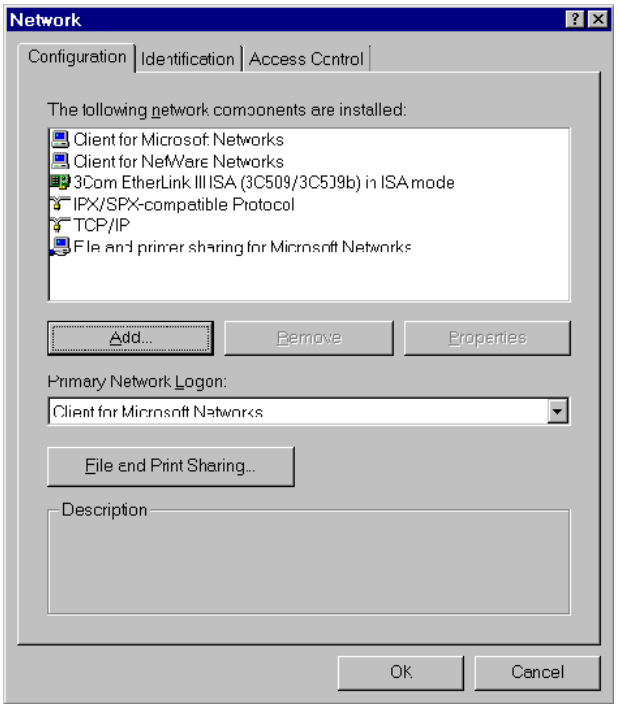
eTrust SSO defines as a field any element of the following types within a Windows window, message box, dialog, or property sheet. These elements usually belong to a set of types, defined in Microsoft Windows documentation. The element types that eTrust SSO relates to as fields include:

- Edit box (text box)
- Combo box
- List box
- Check box
- Command button
- Option button (including radio button)
- Tab (also called tabbed page in Microsoft documentation)
- Tree (used in, for example, Windows Explorer and NT Registry)
- Menu

Following is a sample window with examples of text boxes, check boxes, combo boxes, list boxes, and command buttons:

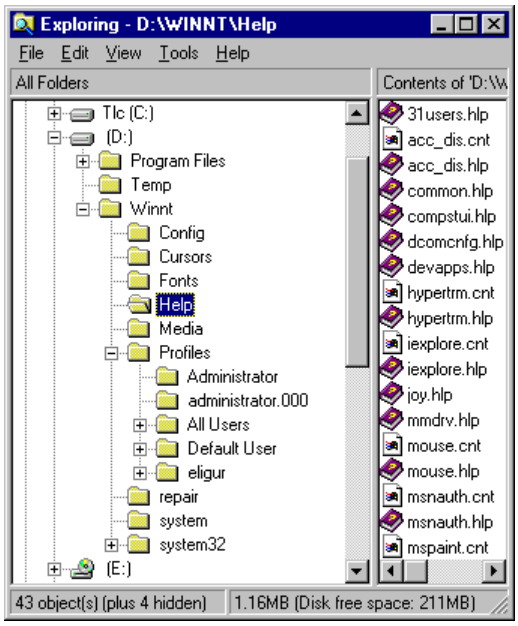


The following window shows tabs, a list box, a combo box, and command buttons. The Configuration tab is selected and that the command buttons Remove and Properties are unavailable (dimmed out).




SSO field-related extensions allow the script to manipulate all these elements, performing operations like clicking check boxes, pushing command buttons, and setting the contents of a text box.

The following window from Windows Explorer shows a menu and a tree:




## Specifying Fields

All field-related extensions require that you specify the target field that they should work on. For example:

```
sso click -label OK
```

The argument OK designates the target field, which is searched for within the target window.

You can specify the target field either by its label (caption), by its class, or by its position and class.

## Specifying Fields by Label

Usually you specify the target field by its label (caption). For some extensions, (such as check, and click) the label is the text value of the target field itself.

For some field-level extensions (such as setfield, getfield, and selectitem), the label you specify is the static text (either prompt or field identifier) that is near the target field. You can specify the location of the target field relative to the label by using the option -pos right|left|top|below|self.

For example, if the target window is the Network window, then "Windows Logon" can be selected using the command:

```
sso selectitem -control combo -pos below \
-label "Primary Network Logon:" -item "Windows Logon"
```

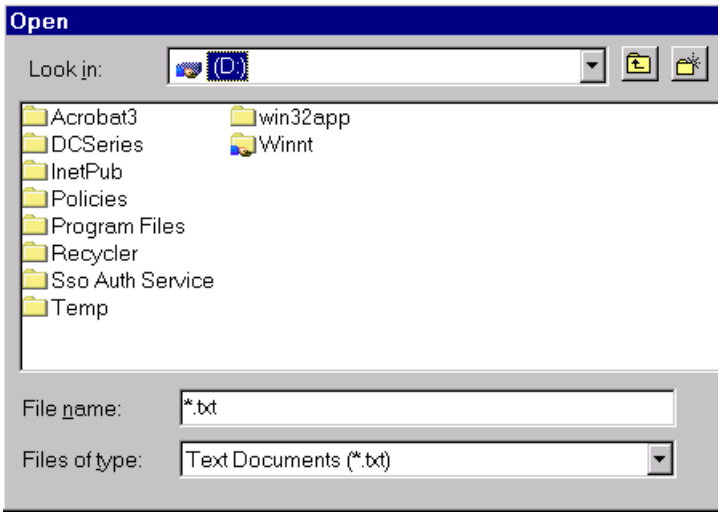
Different extensions have different positional defaults. For example, the setfield extension's positional default is -right and the selectitem extension's default is -below. Therefore, the command below is equivalent to the previous example:

```
sso selectitem -control combo \
     -label "Primary Network Logon:" \
     -item "Windows Logon"
```

The command searches for the target field within a geometric range that is determined by the position of the label. For example, if the switch or default indicates that the target field is to the right of the label, then the search range will be on the same horizontal line where the caption was found, with a certain degree of freedom.

You can specify the label of the target field (that is, the search criterion for the target field) in several ways, similar to the ways you specify a window's title.

- The simplest way to specify a target field is to provide all or part of the label of the desired field. For example, if you want to get the contents of the field captioned "File name:" in the following window:



Here are examples of valid ways to do it:

```
sso getfield -label "File name:"
 sso getfield -label "File"
```

- The more exact your label specification is, the fewer ambiguity problems you will have. You can indicate that the specified label is the exact label (rather than only its first part) by using the -labelexact option. This is necessary if there is a possibility that the label you are searching for is also the first part of another field label in the target window. If, as another example, you want the "File name:" field, but there is another field with the caption "File name: choose from the list", the -labelexact option would be used as follows:

  ```
  sso getfield -labelexact "File name:"
  ```

- Another way to specify the target field's label is to provide a pattern of the label text to be found. This pattern is composed in a standard format, called *globbing,* and is indicated by a -glob switch. This is useful when you do not know exactly how the caption will appear, as when the title is a dynamic string of text that is built of variable data. The following are a few globbing examples, which all match the File Name field:

  ```
  sso getfield -labelglob "Fi*"
  sso getfield -labelglob "Fi?? name:"
  sso getfield -labelglob "Fi?? n*"
  sso getfield -labelglob {[ABCDEFGH]ile name:}
  sso getfield -labelglob {[A-H]ile name:}
  sso getfield -labelglob {[^I-Z]ile name:}
  sso getfield -labelglob {Fil[a-z] name:}
  sso getfield -labelglob {Fil[a-z]*}
  ```

**More information:**

Specifying Fields (see page 100)

## Specifying Fields by Class

The syntax for specifying fields by class is:

```
-class className
```

For example, to use the getfield extension to get the contents of an Edit box, use:

```
sso getfield -class Edit -pos self
```

Specifying fields by class is effective only when there is only one field of that class in the target window.

### Specifying Fields by Position and Class

Specifying a field by its label is more convenient and less error prone than specifying a field by position alone. However, in some situations you are forced to specify by position. This can be the case with applications that were developed with Visual Basic. The label (field caption) generated by Visual Basic is not a separate static field, so you cannot use a label identifier.

The syntax for specifying the position of the target field is

```
-ord number -class className
```

where -ord number is the ordinal number of the field among all the fields of the same class within the target window and -class *classname* is the type of the field (actually the class of the element that forms the field).

The search for fields of a specified class by position is carried out row by row from the top of the target window to the bottom, and from left to right within each row. SSO selects the nth field of the specified class that it finds. The index of the first field is 1.

The following command retrieves the contents of the first text box in a window:

```
sso getfield -ord 1 -class Edit -pos self
```

The following command selects Tile (background position) in the second listbox in the Windows "Display Properties" properties sheet:

```
sso selectitem -ord 2 -class listbox -item "Tiles" -pos self
```

The following command pushes the first command button:

```
sso click -ord 1 -class Button
```

If there are invisible fields in the window, eTrust SSO will treat those invisible fields just like visible fields (that is, include them in a count, access them, and manipulate them).

If you do not know the class of a specific field or if you suspect there are invisible fields in the window, you can use an appropriate utility, such as Spy, to identify all the elements present. (Spy comes bundled with Microsoft Visual C++).

### check

Use the *check* extension to check or to clear a check box. Consider the following check box, which is unchecked:

The following command puts a check in the check box and turns on the option:

```
sso check -label "Reconnect at Logon"
```

For a check box that *is* checked, as follows:


The following command clears the check box and turns off the option:

```
sso check -label "Reconnect at Logon"
```

## click

Use the *click* extension to simulate one or more mouse clicks at a specified field or location. You can specify the exact position where the mouse pointer should be located before the click is simulated. This position, which is specified in pixels, is a relative position, within a specific field or within the whole target window.

The following command locates the mouse pointer at coordinates 10,20 (10 on the x axis, 20 on the y axis), relative to the top-left corner of the target window:

```
sso click -offsx 10 -offsy 20
```

The following command locates the mouse pointer relative to the top-left corner of the "OK" button:

```
sso click -offsx 10 -offsy 20 -label OK
```

If you specify a null argument for -offsx or -offsy by using double quotation marks (""), the mouse pointer is located exactly in the middle of the appropriate axis. The following command locates the mouse pointer exactly in the middle of the target field:

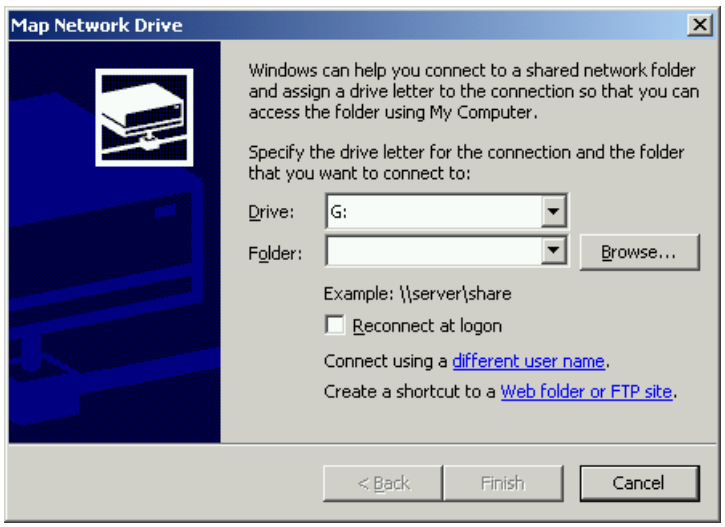
```
sso click -offsx "" -offsy ""
```

The *-numclicks* number option controls the number of clicks the extension performs. To simulate a double click, use -numclicks 2.

A click extension can be used in place of a check extension.

## push

Use the push extension to "push" (click) a command button. For example, if you want to close this window:



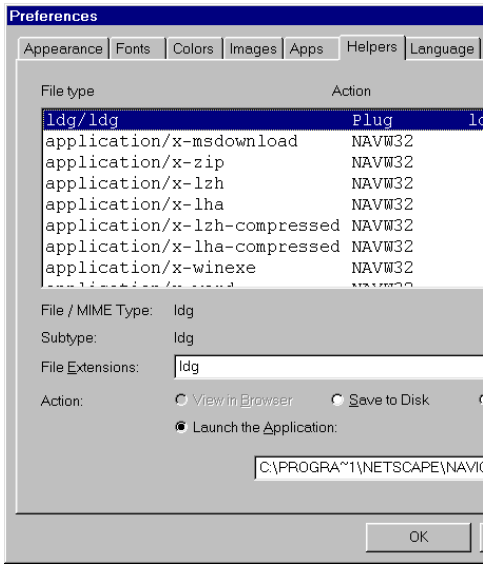

Enter:

```
sso push -label "Cancel"
```

## selectitem

Use the *selectitem* extension to select an item from a list box or from a combo box (editable drop-down list box).

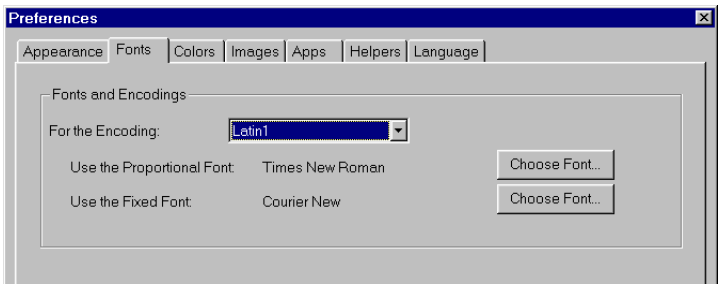
Here is an example of a list box named Preferences in Netscape Navigator:



To select the ldg/ldg item from the File Type list box, enter:

```
sso selectitem -label "File Type" -item "ldg/ldg"
```

This is an example of a combo box:




To select the Latin1 item from the "For the Encoding" combo box, enter:

```
sso selectitem -control combo -pos right \
       -label "For the Encoding:" -item "Latin1"
```

By default, eTrust SSO simulates one mouse click when it selects an item. You can make it simulate a double-click, or no click at all, using -numclicks 2 or -numclicks 0 switches.

## selecttab

Use the *selecttab* extension to select a tab (tabbed page) from a property sheet. For example, in Preferences in the previous illustration, you select the Language tab by:

```
sso selecttab -tab "Language"
```

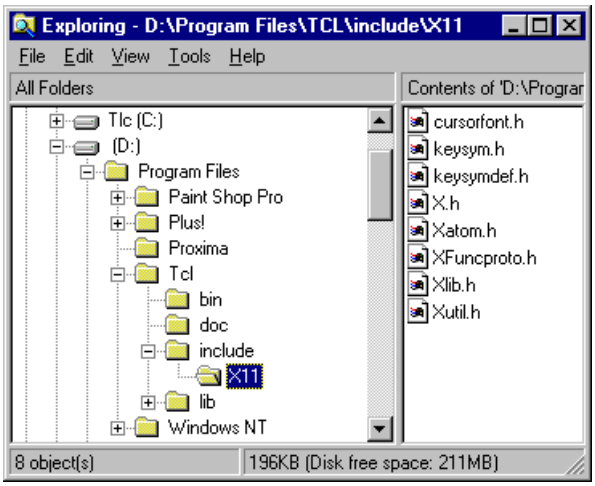
## selecttree

Use the *selecttree* extension to select an object in a tree (for example, in Windows Explorer). The extension's -action option allows you to toggle a selection on or off, to expand a selection (that is, to show the nested folders within the selected folder), or to collapse a selection (that is, to hide the display of nested folders within a selection).

For example, the command:

```
sso selectree -action expand \
  -path "Desktop/MyComputer/D:/Program Files/Tcl/include"
```

produces the following result:




**Note:** The full path must be used as an argument. Path elements must be separated by a slash (/), not by a backslash (\).

## menu

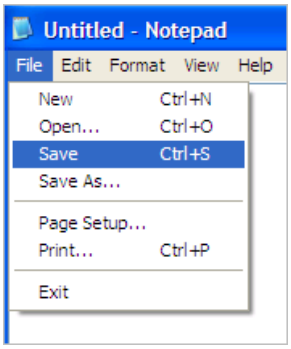
Use the menu extension to select a menu item from the menu bar. The menu functionality also works with AA (Active Accessibility) floating menu bars.

| Setting | Result |
|---|---|
| -menutype is set to 0 | The SSO interpreter will try with GetMenu first, if that fails, it will try again using AA |
| -menutype is set to 1 | The SSO interpreter will use GetMenu which only works for non-floating menus. |
| -menutype is set to 2 | The SSO interpreter will use Active Accessibility APIs which only works floating menus that support AA. |

**Example 1**

To select Save from the File menu in Notepad (which uses a non-floating menu), enter:

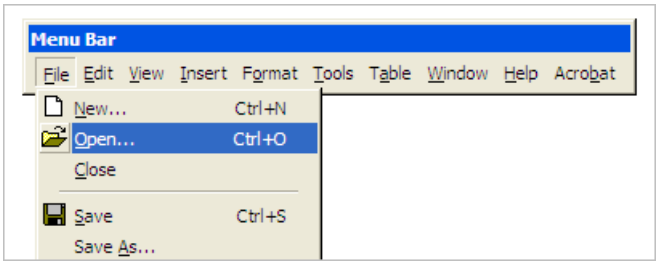
```
sso menu -menutype 1 -item "File/Save"
```




**Example 2**

To select Open from the File menu in MS Word 2000 and above (which uses AA-supported floating menus), enter:

```
sso menu -menutype 2 -item "File/Open"
```




**Note:** If you are not sure whether the application you are referring to uses floating or non-floating menus you should use the -menutype 0 option.

For floating menus that do not support AA, such as the recycle bin application, the only way to select a menu item is by sending appropriate keystrokes to the target window. For example, in the Recycle Bin window on the View menu, select List and run these commands:

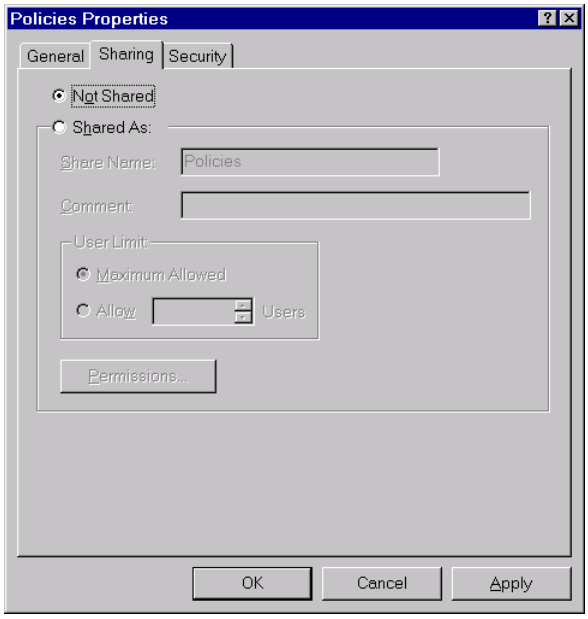
```
sso window -title Recycle
sso type -text "%V{DOWN}{DOWN}{DOWN}{DOWN}{DOWN}{DOWN}{ENTER}"
```

| Command | Result |
|---|---|
| %V | Presses Alt-V which selects the View menu. |

| Command | Result |
|---|---|
| {DOWN}[x6] | Presses the Down key six times to go to the List menu item.<br>Please use the menu's hot-key whenever you can.<br>The number of times you need to list the down command is dependant on the operating system. For example you should use five "Downs" for Windows 2000 recycle bin application, and six "Downs" for Windows XP recycle bin application. |
| {ENTER} | Activates the List menu item |

## getbtnstate

Use the getbtnstate extension to check the selection status of an option button. Consider the following property sheet:



If the following commands are run:

```
set not_shared [sso getbtnstate -label "Not Shared"]
set shared_as [sso getbtnstate -label "Shared As"]
set max_allowed [sso getbtnstate -label "Maximum Allowed"]
```

then the not_shared variable is set to 1, the shared_as variable is 0, and the max_allowed variable is empty string.

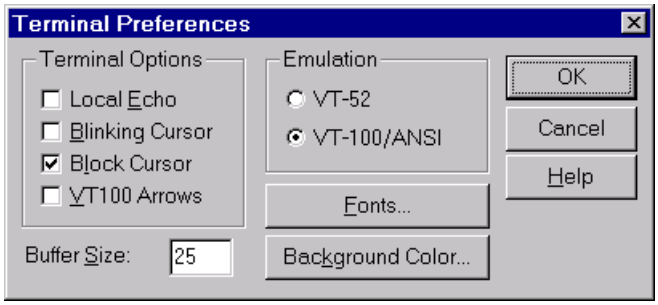
The getbtnstate extension should be used when you want an option button to be in a specific state, but you are not sure of its current state. If you click the button, you might switch it to the unwanted state. In this case, you can check the current state of the button, and act accordingly.

For example:

```
set not_shared [sso getbtnstate -label "Not Shared"]
if { $not_shared == 0 } {
   sso click -label "Not Shared"
}
```

### setfield

Use the *setfield* extension to set the content of a text field. For example, if the following dialog is the active window:




You can set the buffer size to 30 by using the following command:

```
sso setfield -label "Buffer Size" -value 30
```

### getfield

The *getfield* extension retrieves the contents of a field, generally the contents of a text field. For example, if the window shown above is the target window, you can use the following command to retrieve the buffer size and assign it to the buff_size variable:

```
set buff_size [sso getfield -label "Buffer Size"]
```

You can get the same result by specifying the target by class:

```
set button_caption [sso getfield -class Edit -pos self]
```

The extension can also get the contents of other fields, such as button.

## Text-Level Extensions

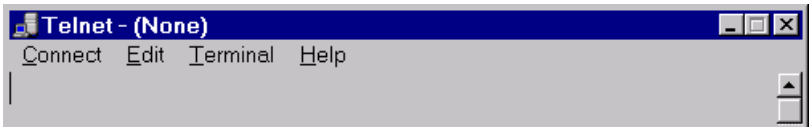
### waittext

The *waittext* extension looks for text, rather than a field. The extension monitors the application that owns the target window (called *the target application* for the purposes of eTrust SSO), and waits for a text string that the application writes to the target window.

Usually, the waittext extension is used with text-based applications, like telnet, 3270 emulations, and 5250 emulations. (It also works with DOS applications in a DOS box, but not when the application is full screen.) A typical SSO script to log into a telnet session looks like this:

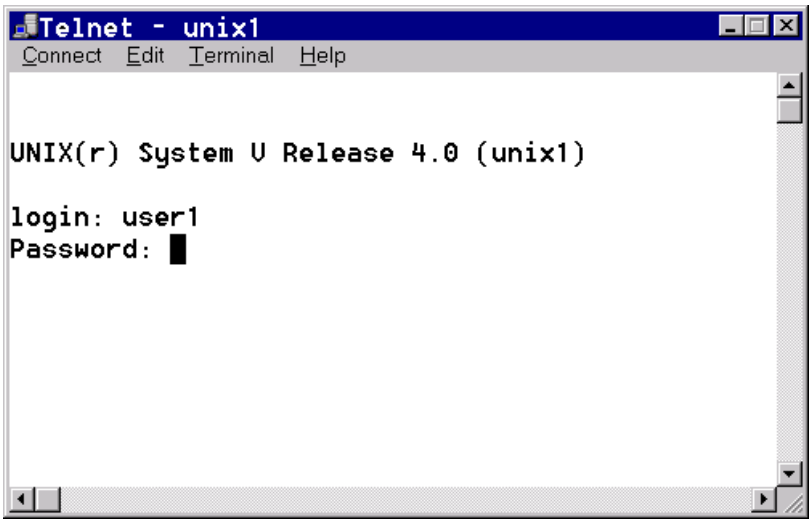
```
sso run -args "unix1" -path "telnet"
sso window -title "Telnet"
sso waittext -text "login:"
 sso type -text "$_LOGINNAME{enter}"
sso waittext -text "Password:"
 sso type -text "$_PASSWORD{enter}"
```

In this example, immediately after the run extension runs, a telnet window similar to this appears:




Then the waittext extension runs and script execution waits until the login: text appears, or until the timeout value is reached.

If there are no problems, then at the end of the script, just before the telnet session is actually opened, the telnet window looks like this:




The waittext extension can accept arguments for more than one text string. The wait period will end when any of the text strings appear, or when the timeout period ends. The return value will indicate which of the specified text strings actually appeared.

For example:

```
set appeared_text [sso waittext \
                   -text "text1" -text "text2"]
```

While SSO is waiting for one of the specified texts to appear, the script execution is suspended. If none of the specified texts has appeared when the timeout value is reached, an error condition is returned.

## type

You can input everything that a user can enter from the keyboard using the type extension in a script. The type extension simulates a sequence of keyboard keystrokes, either alphanumeric characters or special characters. For example, if you want to type the value "abc" in a text field, and then hit the Enter key, you type:

```
sso type -text "abc{enter}"
```

Special character representations, like {enter} for the Enter special character, are called special character mnemonics.

To simulate a keystroke of an alternate value of a specific key, you can use special notations: "+" for the Shift key, "^" for the Ctrl key, and "%" for the Alt key. For example, to simulate the keystroke sequence Ctrl+S (which is usually a shortcut for the "save" command), you type:

```
sso type -text "^s"
```

You can simulate repeating alphanumeric or special keystrokes a given number of times. For example, to simulate the keystroke sequence "Tab Tab Tab 10000" type:

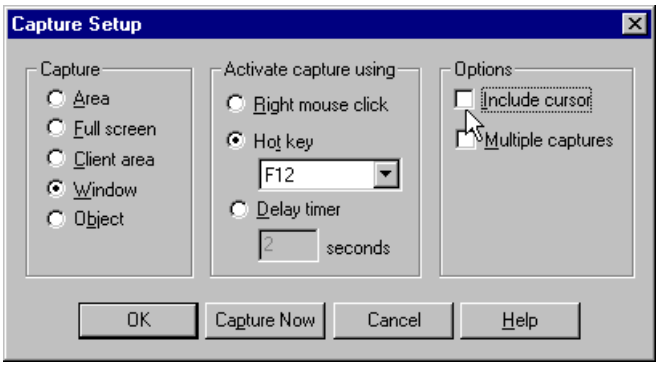
```
sso type -text "{tab 3}1{0 4}"
```

To indicate that a specified string should be typed as is, without interpretation of curly braces and special character representations, use the -literal y (y for yes) option. For example, the following command will type the string "{tab 3}1{0 4}", rather than trying to interpret and convert the string to the "Tab Tab Tab 10000" sequence:

```
sso type -literal y -text "{tab 3}1{0 4}"
```

In most instances, the Microsoft Windows user interface enables the use of keyboard procedures in place of direct (mouse driven) manipulation of desktop elements. As a result, most of the tasks that are carried out using windows-level and text-level extensions can also be done using the type extension.

For example, suppose you want to select the Include cursor check box in the following window:




You can do this either by using the check extension, by specifying the caption of the field:

```
sso check -label "Include cursor"
```

or by using the type extension to first simulate using tabs to focus on the desired field and then simulate typing a space to change the status of the field:

```
sso type -text "{tab}{tab} "
```

However, we strongly recommend that you do not use the type extension when a specific window- or field-level extension can give the required result. Simulating keyboard procedures usually has a number of disadvantages.

For example, using the type extension with a series of tabs to focus on a field has the following drawbacks:

- The resulting script is not as easy to understand.
- The command is more error prone. It is easy to insert three tabs or five tabs, when four are needed.
- The script becomes position-dependent. If the next version of the application adds a new field before the target field, the script has no way of knowing about it. The script will not do what you want it to do, and usually you will not receive any error message.

On the other hand, when you use the check extension for the same purpose, you specify the label of the field, and the extension looks for a field with this label in the current window. If that field is not found, you get an error message that will alert you to a script problem.

# Network Extensions

Scripts often need to perform network-related tasks. eTrust SSO provides extensions to handle these functions.

## General Network Extension

The *net_use* extension enables the script to connect to or disconnect from any network provider working with Windows. It also allows the script to capture a network resource and map a network volume. This extension is very similar to the net use command of Windows.

**Note:** The end user workstation does not need the NetWare client in order to use this extension.

# NetWare Extensions

eTrust SSO also provides specific NetWare extensions—nw_attach, nw_logintree, nw_logout, nw_map, nw_capture, nw_endcap, and nw_setpass—which work, basically, like the analogous NetWare commands.

Refer to Novell NetWare documentation for a detailed explanation of the NetWare commands and operations.

For example, the nw_attach and nw_map extensions would be used when a script has to connect an end user to an application on a Novell volume to which the user is not permanently connected.

The Novell client must be installed on the user's workstation for the NetWare extensions to function. (The Microsoft client for NetWare that is included with Windows 2000, XP, or 2003 is not compatible with the eTrust SSO NetWare extensions.)

# Login Extensions

The SSO Server stores the login information for an application and it sends the necessary information to the user's workstation according to the SSO Client request when the user wants to log in to an application. This login information includes the Tcl script to log in to the application and the login variables.

## Login Variables

Login variables are Tcl variables that contain SSO login information set by eTrust SSO. They are made available to the script during its execution. _LOGINNAME and _PASSWORD contain the name and the password (or ticket) with which the SSO Client logs the user into the target application.

_NEXTPWD contains the new password for the current user and application. It provides an indication that the user password must be changed on the application server.

Assume the following situation:

Terri selects the CICS_TEST application in the Application window (or the eTrust SSO application menu). The application record for CICS_TEST in the eTrust SSO database points to a script named CICS.TCL, the _LOGIN_TYPE variable contains the value TICKET, and the host for this application is MVS_TEST.

The login record in the eTrust SSO database indicates that Terri's login name for CICS_TEST is UTST021. There is no password and next password information in the login record since CICS_TEST is a ticket-based application.

Once Terri selects the CICS_TEST application:

- The CICS.TCL script is sent from the SSO Server to the SSO Client and executed.
- The login name UTST021 is sent to the workstation and stored there in the variable _LOGINNAME, to be available during the script's execution.
- A dynamically generated ticket is sent instead of a real password, and stored in the variable _PASSWORD, so that it will be available during the execution of the script.
- The SSO Client executes the script.

## Using Login Extensions

eTrust SSO login extensions are used to manipulate login variables. For example, to get the login information of applications other than the one you are currently working on, or in order to update the login information.

### chlogin

Use the *chlogin* extension to change part of the login information of the current SSO user for a specific application. This is useful when the script has to handle an application password change. In a case like this, after the script has changed the password in the application itself, the chlogin extension has to update the new password in the SSO Server, to keep it synchronized.

For example:

```
# get new password by using the pwdbox extension
sso pwdbox -prompt "Please enter password"
# change the password in the application itself
sso chlogin -loginname $_LOGINNAME -password $_PASSWORD -appname $_APPNAME
```

To perform this update successfully, the user must be authorized to use the specified application.

## getlogin

There are times when you have logged into one application and you need to access another application. The *getlogin* extension allows you to fetch the login variables to log into the second application.

Consider a case where you want to log into the Pine application but you have to log into the Reflection X application first. The SSO Client displays an icon or a menu item for Pine, and the login variables are set for Pine. However, the script needs the login variables for Reflection X as well. In this case, you can use the getlogin extension to get the login variables to log into Reflection X:

```
sso getlogin -appname ReflectionX
```

After the getlogin has been invoked, the script has another set of login variables, and they are all prefixed with the name of the application. In the preceding example, the script will have variables named ReflectionX_LOGINNAME, ReflectionX_PASSWORD, ReflectionX_HOST, etc. These variables pertain to the second application. The default set of login variables (_LOGINNAME, _PASSWORD, _HOST, etc.) pertains to the first application.

## notify

Besides the login information that was already mentioned, the SSO Server keeps more information related to the specific user and application. For example, the SSO Server keeps some internal counters (such as the login counter) and the audit trail of SSO activities. In some contexts you must inform the SSO Server about certain events and their completion status (in other cases, this is optional). Use the notify extension to allow the server to keep track of these events.

As an example, if your script is responsible for periodic password changes in a password-based application, it is important to let SSO know when the script has successfully changed your password. This is done using the following command (the 0 parameter signifies a successful operation):

```
sso notify -event pwdchange -status 0 -appname Purchasing
```

Another important context is the OTP (One Time Password) context. If the target application that your script logs into supports OTP, you must let the SSO Server know when the old password has been transmitted. SSO needs this information to know that it should calculate the next OTP generation and store it. This is done using the following command:

```
sso notify -event login -status 0 -appname Billing
```

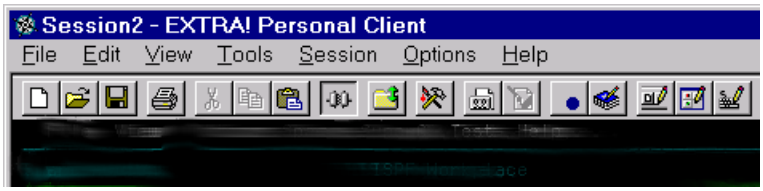
For OTP applications, the most obvious event to wait for before sending the "sso notify" command (that is, the user's prompt appearing) is not the most prudent. Once the old password has been transmitted, it is best to send the notify message as soon as possible. Waiting for things that are partially under user control (such as seeing their prompt, after their .profile or equivalent has been executed) increases the window of opportunity for "man in the middle" attacks. Indeed, perfectly innocent changes the user might make, such as changing their prompt character, might break the login script to the point where the notify message is never sent. It is far safer to send the message at a fixed duration after sending the password, keeping control completely in the hands of the tcl script from when the password is transmitted to when the notify message is sent.

The *getmode* extension retrieves the current state of the SSO Client such as whether the SSO Client is offline. This extension is useful when you want to specify certain actions in a script that depend on the state of the SSO Client. For example, you should avoid making a password change when the SSO Client is offline.

The *logoff* extension terminates the user's SSO session and logs them off the SSO Client.

# 3270 Emulation Extensions

A 3270 emulation program allows you to run mainframe applications from a PC. The program enables a PC to emulate a 3270 dumb terminal by converting the mainframe presentation protocol (the 3270 data stream) to a graphic image in a window on the PC. Here is a typical 3270 emulation window:



In the same way, a 5250 emulation program allows you to run AS/400 programs from a PC.

For the user and the mainframe, these are computer terminal screens showing meaningful data in discrete fields and text, but from the point of view of Microsoft Windows, the graphic images in the emulation window are only bitmaps.

eTrust SSO enables the script developer to manipulate the 3270 or 5250 data on the field or text level with the following Windows extensions:

- sso getfield
- sso getscrape
- sso refreshsso setfield
- sso type
- sso waittext

Before a script uses these extensions with emulation windows, it should set the _MODE variable to console:

```
set _MODE console
```

A script for mainframe programs can also use all the general extensions, although the sso run extension cannot be used from within the emulation program.

When there is a need to provide specialized support for manipulating 3270/5250 data on the field or text level, eTrust SSO scripts can use hllapi_ extensions, the special 3270/5250-specific extensions.

**More information:**

eTrust SSO Extensions for HLLAPI (see page 235)

# Chapter 4: Writing eTrust SSO Scripts

This section contains the following topics:



## Beginning to Write Scripts

Any workstation on the network can write and run basic scripts if it has the SSO Client installed.

**Important!** *In some of the examples in this book, pathnames have been split over two lines in order to fit the page. Pathnames in actual scripts should be written on one line and not divided.*

### Basic Scripting Environment

You can begin writing scripts and testing them if ssointerp.exe is installed on your workstation. You can install the ssointerp.exe by installing the SSO Client. Use a text editor, such as Notepad, to write the script and save the script in the same directory as ssointerp.exe. This enables you to run only those scripts that do not use login extensions, login variables, or hllapi extensions.

You can use ssointerp.exe to run a script from either a Run dialog box or from a shortcut.

To run a script from a Run dialog box, run the script interpreter executable from a Run dialog box with the file name as an argument:

*path*/ssointerp.exe -file *path*/*filename*

Give the full pathname both to ssointerp.exe and to the script file. The script can have either a .tcl or a .txt extension, or no extension at all.

You can also run one SSO command as follows:

*path*/ssointerp.exe -cmd *SSO_extension*

For example:

```
path/ssointerp.exe -cmd sso msgbox -msg "Hello John"
```

To run a script from a shortcut, set up a shortcut with the following parameters:

Target: ssointerp.exe -file *fileName.*

Start in: *pathName* (full pathname of the directory where ssointerp.exe is located).

## Naming Scripts

Script names use the convention *filename*.tcl. While they are being developed, they should be stored in the same folder that contains the SSO Client applications and files. (If for any reason script files are kept in another folder, then the paths in the appl.ini file have to be changed correspondingly.)

To change the location of the scripts on Windows see the following Windows Registry:

```
HKLM\SOFTWARE\ComputerAssociates\eTrustSSO\Server\ssod\"ScriptPath"
```

## Documentation

The script that you write today may have to be modified in the future to accommodate new versions of the application or the OS, and so on. Therefore, it is important that the script itself should be self-documenting, which means that it should contain essential script-specific explanations. This is done by extensive use of comments, such as head comments and line comments. Head comments describe the purpose of the script and its main features. Line comments explain specific commands or groups of commands.

## Production

Once scripts are running successfully, they should be transferred to the SSO Server using scp or sftp and then undergo final testing in remote mode (with the SSO Server).

### Maintenance

You should remember that eTrust SSO scripts use and interact with many variables and elements of the computing environment. Changes in the environment affect the operation of scripts. For example:

- Changes in hard disk organization that change the location of applications may cause sso run commands to fail because the pathname argument is no longer correct.
- Upgrading an application may result in many changes, such as a new executable name or new login windows with different titles and field labels. eTrust SSO extensions that refer to these elements might no longer function as expected.
- Upgrades and changes to operating systems may have similar effects.

Because of this, the administrator supporting eTrust SSO should coordinate with the personnel responsible for version control and be in the loop regarding system environmental changes.

## Screen Scraping Modes

eTrust SSO uses a variety of screen scraping procedures (techniques that allows a PC to intercept character-based data) to follow application behavior during the login process and to identify and manipulate application elements. The method of operation of these procedures is determined by the value of the _MODE variable: win, console, or dos_window, which are all case-sensitive.

When working in Windows mode, which is the default mode, extensions perform screen scraping of controls and the contents of controls.

When working in console mode, extensions perform screen scraping of text.

In DOS window mode all the extensions operate on controls except sso waittext, which retrieves screen contents by cut and paste operations. When the _MODE variable is set to dos_window, additional variables must be set to specify the area that the extension monitors and to set the controls that activate its functions. These additional variables include:

- _BEGIN_CUT
- _CUT_OFFS_BOTTOM
- _CUT_OFFS_LEFT
- _CUT_OFFS_RIGHT

- _CUT_OFFS_TOP
- _CUT_PROC
- _END_CUT
- _HIDE_CUT

# Error Handling

Scripts have to check for errors and then provide instructions for performing appropriate responses.

## Trapping Errors

The following script is an example specifying which window is the active window:

```
sso run -path "C:\\Program Files\\ComputerAssociates\
                \\eTrust SSO Client\\Demo\\Demo_App.exe"
sso window -title "Demo_App - Login"
sso setfield -label "User Name:" -value $_LOGINNAME
sso setfield -label "Password:" -value $_PASSWORD
sso click -label OK
```

However, this check does not result in a significant change for the end user. If the window extension signals an error, the script aborts with a message that Demo_App was not found. Without this check, if the Demo_App - Login window did not appear, the setfield extension, which puts data in the username field, aborts the script in the same manner, but with a message indicating the User Name field was not found.

To make better use of the window extension, the script needs to react to an error in some way other than aborting.

To do this, add the following line at the start of the program:

```
set _ERRORMODE resume
```

This sets a mode that allows the script to continue executing when an error occurs, rather than aborting.

The following script is an example, to explain the concept of allowing the script to continue executing when an error occurs:

```
set _ERRORMODE resume
sso run -path "C:\\Program Files\\ComputerAssociates\
                \\eTrust SSO Client\\Demo\\Demo_App.exe"
sso window -title "Demo_App - Login"
sso setfield -label "User Name:" -value $_LOGINNAME
sso setfield -label "Password:" -value $_PASSWORD
sso click -label OK
```

Once the _ERRORMODE variable is set to resume, the script continues past the second line of the script (the one containing the window extension) even if a window has not appeared. Then the script can execute the additional commands that should be provided to handle the error.

## Handling Completion Code

Most SSO extensions assign a value to the variable _SSOERR according to the status of the action. This value is known as a completion code. A zero value means the extension ran and completed normally. A non-zero value for _SSOERR indicates that the extension did not function as expected. Therefore, by checking for a nonzero value, the script can determine whether an error has occurred.

When an extension runs in stop or trace_stop mode, a non-zero value normally stops the script execution. When an extension runs in resume mode or msg mode, the script should check its completion code (_SSOERR value) to verify success.

Each extension that runs overwrites the previous value of _SSOERR. Therefore, if you want to find out the completion code of an extension after another extension has run, you will have to save the completion code in a different variable.

To illustrate the use of completion code checking, a number of changes will be made to the script example used at the start of this section. The following example illustrates completion code checking:

```
# run Demo_App. Since the default value of _ERRORMODE
# is stop, if Demo_App does not run, then the script
# terminates and the default error message box is shown
sso run -path "C:\\Program Files\\eTrust    \
               \\SSO\\Client\\Demo\\Demo_App.exe"

# set _ERRORMODE to msg
set _ERRORMODE msg
# wait for Login window to appear
sso window -title "Demo_App - Login"

# restore the default value of _ERRORMODE (stop)
 # from now on, if an extension fails, the default error
# message box is displayed and the script is terminated
set _ERRORMODE ""


# examine the value of the _SSOERR variable
if {$_SSOERR != 0} {

   # If the value of _SSOERR is non-zero, display a
   # message box at the user's workstation with the
   # message:  "Demo_App window is not found."
   sso msgbox -msg "Demo_App window is not found."
   # after displaying the message box,
   # terminate script execution
   exit
}


# if the value of the _SSOERR variable was zero, then
# continue executing the program
sso setfield -label "User Name:" -value $_LOGINNAME
sso setfield -label "Password:"  -value $_PASSWORD
sso click -label OK
```

The exit command, like the set and the if commands, is a standard Tcl command, not an eTrust SSO extension.

# Changing Passwords

Scripts have to change the application password when needed. The following example uses Demo_App, which is a password-based application, but the same general method works with ticket-based, One Time Password, Kerberos, and DCE applications.

The following script segment is a continuation of the script in the preceding section:

```
# check that the main Demo_App window is on the screen
sso window -title "Demo_App"
# check if the password is to be changed
# if $_NEXTPWD is empty the if clause will not execute
if { $_NEXTPWD != "" } {
   # push password button
   sso click -label "Change Password"
   # check that the Change Password window appeared
   sso window -title "Demo_App - Change Password:"
   # fill in fields in the Change Password window
   sso setfield -label "User Name:" -value $_LOGINNAME
   sso setfield -label "Password:" -value $_NEXTPWD
   sso setfield -label "Verify Password:" -value $_NEXTPWD

   # and push OK button
   sso click -label OK
   # notify the SSO Server on successful password change
   sso notify -event pwdchange -status 0 \
        -appname Demo_App
}
```

The execution of the if clause depends on the value of the login variable _NEXTPWD sent by the SSO Server. The _NEXTPWD variable has a non-empty value when the password has to be changed. If the _NEXTPWD login variable is empty (has a null value), then the script does not need to change the password.

# Scripting for Browser-Based Applications

eTrust SSO users in many companies have to access password-protected applications and sites on company intranets and on the Web.

Using the SSO Client, the process requires that the script:

- Start up a browser
- Navigate to the required site (if the browser was not invoked with a URL argument)
- Enter login data in the appropriate fields.

In general, scripts for these applications and sites are developed in the same way as a script for a network or client/server application.

When the browser-based application provides a freestanding authentication dialog, the application login script can use the eTrust SSO general and window extensions with default variable values.

When edit boxes are embedded in the browser page (rather than in a freestanding dialog box), you still use standard extensions. However, it may be necessary to use extensions in the console mode to identify text in the browser window. This is done by setting the _MODE variable to console before the extension is called.

Microsoft's Internet Explorer requires the use of a specific set of SSO extensions, called *html_ extensions*.

# Scripting for 3270/5250 Applications

Develop scripts for 3270/5250 applications on a workstation with a 3270 emulator, a working 3270 connection, and the SSO Client enabled.

Use the following eTrust SSO Windows extensions to manipulate data inside the 3270 or 5250 emulation window:

```
sso getfield
sso getscrape
sso refresh
sso setfield
sso type
sso waittext
```

These extensions do not use HLLAPI and the type of emulation used does not affect them. However, the _MODE variable has to be set to console before the extensions are used in the script.

If a particular environment or application requires specialized support, the script developer can use the eTrust SSO HLLAPI extensions.

# Pre- and Post-Commands

In some cases, different scripts require identical sections of code. To reduce production effort and to reduce errors, eTrust SSO can use a standard pre-command, a standard post-command, and a global pre-command file together with the individual application script.

### Application Pre-Command and Post-Command

The application record in the eTrust SSO database has fields named SCRIPT_PRECMD and SCRIPT_POSTCMD, which point to pre-command and post-command files. By using these fields, the eTrust SSO administrator can define initialization and termination processing for a specific application, regardless of the specific script that will be executed to log into this application.

The pre-command is normally used to initialize application-specific variables. These variables, like the general login variables, are available to the script as it begins execution. The pre-command and the post-command are prefixed and appended to the script and become part of the script before the script reaches the SSO Client workstation.

### Global Pre-Command File

In addition to the application-specific pre-command and post-commands, the administrator can create an installation-wide pre-command file that will be executed as part of any script. This file is normally used to define installation-wide Tcl procedures that can be used to perform common operations that most of the application scripts need.

Using this method, all the application scripts can call these procedures, rather than using their own internal code each time. This method can save development and maintenance time, as well as disk space. There is also less chance for script error, because the global file is usually checked more extensively than each of the scripts.

## Scripting Tips

The following tips may prove useful in developing scripts.

### User Name and Password as run Extension Parameters

Some applications, such as cc:Mail, let you specify user name and password as command-line parameters. We recommend that you use user name and password arguments with the run extension, rather than writing screen scraping code to set user name and password fields.

For this reason, when you begin to prepare a logon script for a new application, first examine the command line options available when starting up that application.

## Session Profile as run Extension Parameters in 3270 Programs

Some 3270 emulation programs, such as Host Explorer, let you build a predefined session profile, save it locally, and use it as a command line parameter. These session profiles generally include host name, computer type, session short name, and possibly additional data.

In emulation programs that allow the use of a session profile as an argument in the sso run command, you can bypass an Open New Session or equivalent screen and simplify script development and execution. For example, you can start Host Explorer with a predefined profile called HOST-A by using the following:

```
sso run -args "-A -P HOST-A" \
       -path "c:/HOSTEX/PROGRAMS/TN3270.EXE"
```

## Checking Pathnames

Do not assume that programs and files will always be found where a standard installation puts them and do not take for granted that the pathname to an executable will be the same on all of the workstations in an organization or even in one department. Custom installations, restoring backups, and disconnecting and remapping drives can all relocate files intentionally or inadvertently. As a result, the pathname that the script needs to run a program, may differ from user to user; for example: C:/Program Files/Notes/notes.exe for one user and C:/Notes/notes.exe for another.

When the run extension fails to find its target application, it displays an error message and terminates the script.

One way to reduce the scope of this problem is to have the script append to the user's PATH with additional possible locations for the target application.

For example, a script to open Lotus Notes could begin:

```
append env(PATH) {;C:\Notes}
append env(PATH) {;C:\Program Files\notes}
append env(PATH) {;C:\Program Files\Lotus\Notes}
append env(PATH) {;D:\Notes}
append env(PATH) {;D:\Program Files\notes}
append env(PATH) {;D:\Program Files\Lotus\Notes}
. . .
 sso run -path notes.exe
```

**Note:** The change to PATH is in effect only as long as ssointerp.exe is running. There is no effect on the user's environment once the script has terminated.

Another way to handle the problem is to check the pathname in the script before using the run extension and then branch according to the result of the check.

Tcl has a native command, file exists, which verifies the pathname to a file. If the pathname is found, the command returns 1. If it is not found the command returns 0. The command syntax is:

```
file exists pathname
```

In the following example, the user is asked to contact the system administrator if the script's pathname is incorrect.

```
set pathname "C:/notes/notes.exe"
if {[ file exists $pathname ] == 0 } {
     sso msgbox -msg "Lotus Notes may not be \n\
                  properly installed on your \n\
                  workstation. \n\
                  Call the system administrator at \n\
                  extension 2073 for instructions \n\
                  on how to proceed."
    exit 0}
sso run -path $pathname
```

The script could also ask the user to input the correct pathname by using the following as an argument of if:

```
set pathname [sso inputbox -prompt "If you know the \
               correct pathname, enter it below and\
               click OK. Otherwise, click Cancel."]
```

# Troubleshooting

We recommend that you thoroughly test your scripts. The following steps can help with scripts that fail to do what you expect:

- Verify that the script you think you are testing is really the one that is executing. (An easy way to do so is to use sso msgbox.)
- To temporarily eliminate problems resulting from variable substitutions, experiment by replacing variables with hard-coded values.
- Insert msgbox extensions in suspect code sections of your script to display variable values and to record branching.
- If there is a possibility that the problem is timing-related, set the _PAUSE variable to a reasonable value, for example, try one second. Run the script again. If the problem is gone, it probably was a timing problem. Try to use sleep at key points to see where the problem exists.

If the SSO Client runs on a computer that is faster than the host it communicates to is running, the script commands will run very quickly. This may cause errors. To fix the problem, use the _PAUSE variable to set a small pause (usually a couple of hundred milliseconds) between each SSO command. This can be of help for other timing problems.

- Other timing problems can be resolved by increasing the _TIMEOUT variable value.
- Try to replace failing code with different code that does the same thing. For example, if you have problems with the menu extension, try the type extension, using menu access keys.

  If you have problems with Tcl syntax, try adding curly braces around the code that is causing problems. If that does not help, try adding square braces around it.
- If you have problems with Windows extensions when trying to access a field, verify that you have identified the actual elements of the target window. Also verify that the script is using the actual attributes and identifiers of the target window or field.

  In some cases, application sign-on windows and even parts of operating systems were written according to different guidelines. For example, in some fields, disk and folder names are associated with icons in a way that prevents normal text matching.

  It is often helpful to use a simple Tcl program (like the one below) to verify the identity of window elements. Utilities from various vendors might also help.

  For a practical example, suppose you are writing a script segment to map a network drive using the following Windows dialog:


  You do not want the drive to reconnect at logon, but before clicking the Reconnect at logon checkbox off, you first have to determine its state.

Use the following command to determine the state of the checkbox:

```
set state [sso getbtnstate -label "Reconnect at logon"]
```

You receive the following error message:

```
Script execution error: \
Cannot find item "Reconnect at logon"
```

Since the checkbox belongs to the Button class, you can run the following Tcl program to identify its ordinal number:

```
sso window –title "Map Network Drive"
for {set i 1} {$i <= 4} {incr i} {
     set btn [sso getfield -ord i -class Button –pos self]
     sso msgbox -msg "$i Button  <$btn>"
}
```

This program eventually displays a message box containing:

```
2 Button <Reconnect at logon>
```

Now you can go back to your original script and rewrite the sso check command to read:

```
set state [sso getbtnstate -ord 2 –class Button]
```

- When using field specification by both position and class, (-ord n -class className), you may experience problems with the ordinal number you specify, for example, if you wrote sso click -ord 2 -class button, but the first button is pushed. This could be caused by the presence of invisible fields in the window. If there are invisible fields, SSO treats them as regular fields, that is, it considers them during the counting, enables access to them, and so on. In order to find the invisible fields in the window, you should use an appropriate utility, such as Spy.
- A script may fail because it cannot find the target window. One way to make sure that focus is not lost is to make the target window the current window. You should set the _TARGET_WIN variable to current at the start of a section that encounters this problem:

```
set _TARGET_WIN current
```

# Chapter 5: SSO Extensions

This section contains the following topics:



## Overview

This chapter provides a detailed reference for the eTrust SSO extensions to the Tcl language.

The chapter includes:

- Syntax conventions for designating windows and fields, as used by all extensions that access windows and fields
- String matching conventions, which are used by all extensions that take strings as arguments
- A list of all eTrust SSO extensions by category (general, Windows, HLLAPI, network, HTML, and login extensions)
- Extension reference—including syntax, description, return value, and example for each extension; extensions are listed alphabetically

**Note:** This reference makes considerable use of typographical conventions to explain and illustrate the syntax of eTrust SSO extensions. A backslash indicates that a command continues on the following line.

**More information:**

Conventions (see page 14)

## Special Syntax Conventions

This chapter uses special syntax descriptions to describe arguments that include window specifications and field specifications. These descriptions are explained here in detail in order to avoid repeating the full syntax specifications and explanations for each of the relevant extensions in this reference chapter.

## Specifying Windows

The syntax description of an SSO extension that has to designate a specific window or subwindow includes the notation:

*windowSpec*

This notation is a placeholder for one of the following syntax segments for the different ways the target window can be designated. Please note each occurrence of either –title|-titleglob|-titlexact, -class or –target is treated as a potential search target and the search will be conducted in the order they are specified in the command. The first window found will be returned:

- Designation by the title (caption) of the window or subwindow:

  `-title|-titleglob|-titlexact` *targetText*

  A window's title is not always predictably fixed. Applications that create their own window have complete control over their window title, and may modify it to reflect the state of the program - the name of a loaded document, for instance. Applications that run from the command line (such as telnet) have no control over their command line; instead, Windows determines the window's title. This can change depending on how the command is invoked; starting the command from the "Start" menu might result in a different window title to when the same application is launched from the SSO tools menu, for example.

  If you want to specify a window that may have more than one title, you can construct the window command to search for one of several different titles.

  In the following example, the interpreter can use either Telnet or "OtpTelnet". It will use the first match it finds.

  ```
  sso window -titlglob "*Telnet*" -titlexact "OtpTelnet"
  ```

| Key | Key Value | Description |
|---|---|---|
| -title | *targetText* | Indicates that eTrust SSO searches for the specified title text as the first part of the text in the window title bar. This also allows for an exact match. |
| -titleglob | *targetText* | Indicates that the title text provided is a pattern formatted in globbing format. eTrust SSO searches for a title that matches the pattern specified. |
| -titlexact | *targetText* | Indicates that eTrust SSO has to match the label text provided to the whole title and not just to the first part. |

| Key | Key Value | Description |
|---|---|---|
| *targetText* | | Is a text string that provides the search criteria. eTrust SSO matches this string to the title in the title bar of the windows that it surveys in the search for the target window. Text matching is case-sensitive and follows these rules:<br>■ The ellipsis (...) at the end of a caption string is ignored.<br>■ The first of multiple ampersands (&) in a caption string is ignored.<br>■ The special symbols \n, \r, \t, \\, (space), \\, and \ are treated as literals, rather than as escape symbols. |

- Designation by the class of the window or subwindow:

  `-class` *`className`*

| Key | Key Value | Description |
|---|---|---|
| -class | *className* | Specifies the name of the target window's class in Microsoft Windows notation. |

You can identify the class of a window by using an appropriate utility, such as Spy (bundled with Microsoft Visual C++).

- Designation by the title and the class of the window or subwindow:

  `-target {-title|-titleglob|-titlexact` *`targetText`* `-class` *`className`*`}`

  The -target option can be used to specify the title of a window as well as its class. You can use any combination of "-title|-titleglob|-titlexact" and "-class". The following example will only find a Notepad document that is called "word.txt".

  ```
  sso window -target {-class "Notepad" -title "word.txt - Notepad"}
  ```

  If any criterion within the -target curly brackets is not met, the error message "window not found" occurs.

**More information:**

Specifying Windows (see page 52)
Globbing Format (see page 98)

## Globbing Format

A string that is the criterion for glob-style matching can contain both characters to be matched literally and the following special characters:

| Special Characters | Function |
|---|---|
| * (asterisk) | Matches any string of characters, any one character, or the absence of a character in the target string. |
| ? (question mark) | Matches any one (single) character in the target string. |
| [ ] (square brackets) | Matches any one character that is the same as one of the characters in the character list delimited by the brackets. For example, [abcd] matches a, b, c, or d in the target. When you use brackets, you must put braces around your string. Additional information on character lists follows this table. |
| \ (backslash) | Allows the use of special characters for matching. The character after the backslash is treated like an alphanumeric character, rather than a special character. For example, * matches only the asterisk, rather than matching other characters (as the asterisk normally does in glob-style matching). |

## Character Lists

When a character list is composed of one or more than one discrete and explicit characters (for example [abcdefg]), then the match is with any single character in the list. If the list within the brackets is preceded by a caret (^), then the match is with any single character that is *not* in the list.

When the character list is given as a range (for example, [a-g]), then the match is with any single character within that range, inclusively. If a caret (^) precedes the range notation, then the match is with any single character not in the range. You can specify both ends of the range, or only its first or last character.

When you use a character list as part of a switch (option) argument, put the argument in braces. For example:

```
sso window -titleglob {[A-Z]ntitled}
```

The following table describes the character lists that can be used. The expressions *ch1*, *ch2*, and *chN* each stand for a single character; *ch2*...*chN* stands for all the characters from ch2 to ch5 inclusive.

| List in Brackets | Matches |
|---|---|
| [*ch1ch2...chN*] | Any single character in the list enclosed by the square brackets. |
| [**^***ch1ch2...chN*] | Any single character that is *not* in the list enclosed by the square brackets. |
| [*ch1***-***ch2*] | Any single character in the range, inclusive. |
| [**^***ch1-ch2*] | Any single character that is *not* in the inclusive range. |
| [**-***ch2*] | Any single character with an ascii value lower than or equal to the specified character (*ch2*). |
| [**^-***ch2*] | Any single character with an ascii value *higher* than the specified character (*ch2*). |
| [*ch1*-] | Any single character with an ascii value equal to or higher than the specified character (*ch1*). |
| [**^***ch1*-] | Any single character with an ascii value *lower* than the specified character (*ch1*). |

The following table contains examples of globbing format and character lists:

| Specify... | To match... | But not... |
|---|---|---|
| *i*.c | lis.c, list.c, quatrain.c | box.c, list.b |
| st*.h | st.h, str.h, string.h | list.h, string.b |
| * | any string | |
| ?i?.c | lis.c, min.c | list.c, is.c |
| st?.t | sta.t, stb.t, st7.t | string.t, str.c |
| mmc04.? | mmc04.a, mmc04.1 | mmc04.doc, mmc04. |

| Specify... | To match... | But not... |
|---|---|---|
| doc[pqr] | docp, docq, docr | docx, doc1 |
| arc[r-x] | arcr, arcu, arcx | arca, arcz |
| fax[^d-] | faxa, faxb, faxc | faxd, faxx |

## Specifying Fields

The syntax description of an SSO extension that uses a field specification includes the notation:

*FieldSpec*

This notation is a placeholder for the following alternatives:

- Field specification by label (caption):

```
-label|-labelglob|-labelexact labelText\
 [-pos right|left|above|below|self]
```

| Key | Key Value | Description |
|---|---|---|
| -label | *labelText* | Indicates that eTrust SSO will search for the specified label text as the first part of the label text. This also allows for an exact match. |
| -labelglob | *labelText* | Indicates that the text provided is a pattern formatted in globbing format. eTrust SSO searches for a label that matches the pattern specified. |
| -labelexact | *labelText* | Indicates that eTrust SSO has to match the label text provided to the whole label, not just the first part. |
| *labelText* | | The text (caption) of the label. |
| -pos | right | Optional. Specifies that the target field is located to the right of the specified label (caption). |
| | left | Optional. Specifies that the target field is located to the left of the specified label (caption). |
| | above | Optional. Specifies that the target field is located above the specified label (caption). |
| | below | Optional. Specifies that the target field is located below the specified label (caption). |

| Key | Key Value | Description |
|---|---|---|
| | self | Optional. Specifies that the label itself is the target field. |

**Note:** The key -pos is not relevant in _MODE console. In all cases the search is case-sensitive.

- Field specification by the class of the field:

```
-class className \
 [-pos right|left|above|below|self]
```

| Key | Key Value | Description |
|---|---|---|
| -class | *className* | Specifies the name of the target subwindow's class in Microsoft Windows notation. |

You can identify the class of a field by using an appropriate utility, such as Spy (bundled with Microsoft Visual C++).

Other parameters are as for Field specification by label.

**Note:** In all cases the search is case-sensitive.

- Field specification by the ordinal number of one of a number of fields of the same class:

```
-ord number -class className
```

| Key | Key Value | Description |
|---|---|---|
| -ord | *number* | Specifies the ordinal number of the target field among all the fields of the specified (or default) class within the target message box or dialog box. The ordinal number of the first field is 1. If there are any invisible fields present, SSO treats them as if they were visible fields (includes them in the count, allows access to them, etc.). To reveal invisible fields in a message box or dialog box, use a utility like Spy, which comes bundled with Microsoft Visual C++.<br><br>Using -ord 0 means that the default will work as -ord 1. |
| -class | *className* | Specifies the name of the target window's class in Microsoft Windows notation. You can identify the class of a field by using an appropriate utility, such as Spy (bundled with Microsoft Visual C++). |

**Note:** In all cases the search is case-sensitive.

- Field specification by the ordinal number of a specified class and title (the extension searches for a specific instance of fields of the same class with the same label):

  ```
  -ord number -class className \
  -label|-labelglob|-labelexact labelText \
   [-pos right|left|above|below|self]
  ```

  The parameters are as described in the preceding specifications.

**More information:**

Specifying Fields (see page 60)
Globbing Format (see page 98)

# Extension List by Categories

## General Extensions

The following table summarizes the eTrust SSO General extensions:

| General Extensions | Description |
|---|---|
| askyesno | Displays a dialog box with the specified question and Yes and No buttons; returns the selected button as the return value. |
| inputbox | Displays a message box; receives and returns text input from the user. You can use the -wrap option with this extension (specify Y or N)*. |
| msgbox | Displays a message box; returns the value of the button that was selected by the user. |
| pwdbox | Gets a password from the user; returns the value typed by the user. Can require verification by retyping. You can use the -wrap option with this extension. |
| run | Runs a program at the user's workstation. |
| sleep | Suspends execution of the script for a specified amount of time. |
| statusbox | Displays a message box with text and a Cancel button. The script continues to run while the message is displayed (unlike msgbox). |
| terminate | Terminates the current application. |

* If you specify Y for the –wrap option long text is wrapped properly and words will not be broken across two lines.

## Windows Extensions

The following table summarizes the eTrust SSO Windows extensions:

| Windows Extensions | Description |
|---|---|
| check | Checks or clears check boxes. |
| click | Simulates one or more mouse clicks at a specified location. |
| getbtnstate | Gets the selection status of a button or check box. |
| getfield | Retrieves the contents of a specified field. |
| getmsgtext | Returns the text of the specified message box. |
| getscrape | Returns the contents of the screen scrape. |
| lockinput | Locks input to the workstation from the keyboard and the mouse. |
| menu | Selects a menu item. |
| push | Pushes a specified button. |
| refresh | Refreshes the screen scrape. |
| screensize | Gets the dimensions of the display area in pixels and puts them in variables. |
| selectitem | Selects an item from a list. |
| selecttab | Selects a tab in a property sheet (Windows 2000/XP/2003). |
| selecttree | Selects directories and files in Windows Explorer (Windows 2000/XP/2003). |
| setfield | Sets the contents of a specified field. |
| shutdown | Shuts down the Window |
| subwindow | Finds the appropriate document window of the target window. |
| type | Types text, including special characters. This can also be use to send control characters to the target window. |
| waittext | Waits for a string, or one of a series of strings, to appear on the screen. |

| Windows Extensions | Description |
|---|---|
| window | Finds a specified target window. |
| wintitle | Gets the title of a window. |
| unlockinput | Removes the lockinput command so that users can use the keyboard and the mouse. |

## Network Extensions

The following table summarizes the network extensions:

| Network Extensions | Description |
|---|---|
| net_use | Attaches or disconnects from any network provider running with Windows. Also does capture and mapping. |
| nw_attach | Attaches to a NetWare server. |
| nw_capture | Captures printer output to a NetWare queue. |
| nw_endcap | Ends a printer capture. |
| nw_logintree | Logs into a directory tree of NDS. |
| nw_logout | Logs out from a server. |
| nw_map | Maps a NetWare volume to a local drive. |
| nw_setpass | Sets a password on a NetWare server. |

You do not need the NetWare client to use sso net_use with NetWare resources. However, to use the NetWare extensions, you must install the appropriate Novell client for Windows. (The Microsoft Client for NetWare bundled with Windows 2000/XP/2003 is not compatible with the eTrust SSO NetWare extensions.)

## HTML Extensions

Windows extensions described in the previous section, work with all Netscape browsers and MS Internet Explorer browsers through version 3.*x*. However, in order to work with Internet Explorer 4.0 and up, scripts must use the following special HTML extensions:

| HTML Extensions | Description |
|---|---|
| html_browse | Opens an Internet Explorer browser window. |

| HTML Extensions | Description |
|---|---|
| html_connect | Establishes a connection to an already opened browser window with the specified window title. |
| html_disconnect | Disconnects from an open Internet Explorer window. |
| html_grabpage | Invokes an active Internet Explorer window. |
| html_getfield | Retrieves the contents of a specified field in the browser window. |
| html_getselecteditem | Selects the text in an edit box or list box specified by the html_selectitem extension. |
| html_getframesnum | Selects the specified frame in an Internet Explorer window |
| html_navigate | Opens a new site in an Internet Explorer browser at a specified URL. |
| html_push | Pushes a designated button in the browser window. |
| html_search | Puts text in the search mechanism of the Web page and pushes the search button. |
| html_selectitem | Selects a specified item in an edit box or listbox in the browser window. |
| html_setfield | Enters a value in a specified edit box in the browser window. |

**Note:** html text outside html tags is invalid syntax for html extensions.

## Login Extensions

The following table summarizes the eTrust SSO login extensions:

| Login Extensions | Description |
|---|---|
| chlogin | Sets login information for a specific application. |
| getlogin | Gets login information (login variables) for a specific application. |
| getmode | Determines the current state of the SSO Client. Especially useful with offline functionality. |
| logoff | Terminates the user's SSO session. |
| notify | Notifies the SSO Server of a login related event. |

**Note:** The login extensions access and update the database on the SSO Server.

## hllapi Extensions

The following list summarizes the eTrust SSO hllapi extensions:

| hllapi Extensions | Description |
|---|---|
| hllapi_connect | Connects to a 3270 session. |
| hllapi_disconnect | Disconnects the last connected 3270 session. |
| hllapi_getcursor | Returns the current location of the cursor in the current 3270 session. |
| hllapi_getfield | Returns the contents of the specified field in the current 3270 session. |
| hllapi_getscreen | Returns the contents of the screen in the current 3270 session. |
| hllapi_setcursor | Sets the cursor at a specified location in the current 3270 session. |
| hllapi_setfield | Sets the value of a specified input field in the current session. |
| hllapi_type | Types characters to the current 3270 session. |
| hllapi_waitsys | Waits for the host system to return to input mode, so that the terminal can accept keystrokes. |
| hllapi_waittext | Waits for a specified text to appear on the current session's screen. |

**Note:** All hll_extensions have been removed. There is an equivalent hllapi_ extension for every old hll_ extension so you should now use hllapi_ extensions instead.

# Extension Details

## askyesno

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The askyesno extension displays a message box with a text string, which is generally a question, together with Yes and No command buttons. The extension returns the label of the selected button as the return value.

The window size is determined by the length of the prompt text as well as the title text.

**Syntax**

```
sso askyesno -prompt promptText [-title titleText]
```

| Key | Key Value | Description |
|---|---|---|
| -prompt | *promptText* | The text string that is displayed in the message box. |
| -title | titleText | The text of the message box title. Optional; the default is eTrust SSO. |

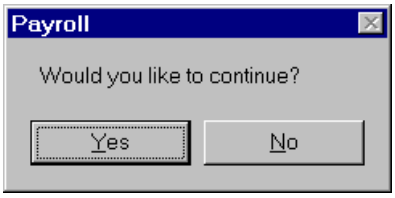
### Return Value

A return value of no indicates the user selected the No button. A return value of yes indicates the user selected the Yes button.

### Example

The following example displays a box with the question "Would you like to continue?" and Yes and No command buttons:

```
set opt [sso askyesno  \
     -prompt "Would you like to continue?" -title Payroll]
 if {$opt == "no"} {
  # user selected No
  ...
} else {
  # user selected Yes
  ...
 }
```




### More information:

inputbox (see page 147)
msgbox (see page 152)

## check

### Compatibility

Windows 2000/XP/2003.

### Purpose

The check extension selects or clears a check box or option button. If the box is not selected (empty) it selects it (puts a check in the box); if it is selected, the extension clears it. The extension fails if the specified field is disabled. eTrust SSO looks for the specified field within the target window.

When you specify the target field by position and you omit the class parameter, the check extension defaults to the button class.

With other classes, you must use the option -class.

### Syntax

sso check *FieldSpec*

***FieldSpec***

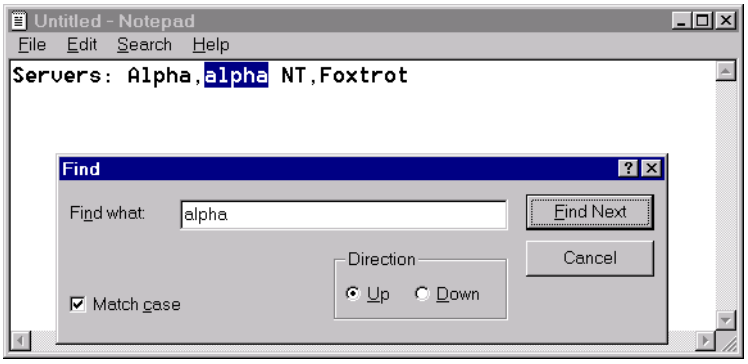
Specifies the target field (check box or option button). For more information, see the section, Specifying Fields in this chapter.

### Example

This example shows the use of the check extension, together with the functionally similar click extension. First, the example opens Notepad, types in some text, and opens the Find box:

```
sso run -path notepad.exe
sso window -titleglob "*Notepad*"
sso type -text "Servers: Alpha,alpha NT,Foxtrot"
sso menu -item "Edit/Find"
sso window -title "Find"
```

This displays the following window:




The code below enters search criteria, puts a check in the Match case check box, clicks the Up button, and starts the Find function:

```
sso setfield -label "Find what" -value "alpha"
sso check -label "Match case"
sso click -label "Up"
sso click -label "Find Next"
```

**Note:** The previous script segment will run equally well if the check extension is replaced with a click extension.

### More information:

click (see page 111)

## chlogin

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The chlogin extension modifies an existing SSO user record by passing login variable values to the SSO Server. The next time the user starts the login process, the login variables that have been updated by chlogin will be retrieved and used by the SSO Client.

The _NEXTPWD variable is subject to quality checks by the SSO Server. If the -check n option is used, then no quality check is carried out. However, if an application has password synchronization enabled and the -sync n option is not used, then the next password for the original application (as stored in _NEXTPWD) is propagated as a next password to all synchronized applications and a quality check is run.

The -check y option checks consolidated password policies. The –check option only has effect if the _NEXTPWD variable is used.

The extension can also initiate password sync. As a default, the chlogin extension initiates a password change in all synchronized applications. If you do not want password synchronization to take place, use -sync n.

**Note:** This extension will fail if the client is in an offline state. The online status may be determined using the getmode extension. See the getmode extension documentation for further details.

**Syntax**

```
sso chlogin -loginname name -password pwd -appname AppName \
            [-check y|n] [-sync y|n] [-nextpwd pwd]
```

| Key | Key Value | Description |
|---|---|---|
| -loginname | *name* | Specifies that the login-name variable will be changed and provides a text string for the new value. |
| -password | *pwd* | Specifies that the password variable will be changed and provides a text string for the new value. |
| -appname | *AppName* | Specifies the name of the application for which the login variables should be set. |

| Key | Key Value | Description |
|---|---|---|
| -check | y\|n | Optional. Specifies whether to run a quality check on the password.<br>The default is y (does quality check).<br>The –check option only has effect if the _NEXTPWD variable is used. |
| -nextpwd | *pwd* | Optional. Specifies that the next-password variable will be changed and provides a text string for the new value.<br>If the -nextpwd option is omitted from the -chlogin extension no quality checks are performed. |
| -sync | y\|n | Optional. Specifies whether the next passwords will be changed on other synchronized applications.<br>The default is y (to synchronize passwords). |

**Example**

This example changes the user's password for the "telnet" application in the SSO database:

```
sso chlogin -loginname $AppLoginName -password $new_password -appname telnet
```

**More information:**

_LOGINNAME (see page 207)
_NEXTPWD (see page 209)
_PASSWORD (see page 209)

## click

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The click extension simulates a mouse click (or clicks) in the specified field. eTrust SSO searches for the specified field within the target window. You can specify the exact coordinates of the click.

The extension fails if the specified field is disabled.

The click extension can be used in place of the check extension.

**Syntax**

```
sso click [-numclicks number] [-button left|middle|right] [-offsx x] [-offsy y]
FieldSpec
```

| Key | Key Value | Description |
|---|---|---|
| *FieldSpec* | | Specifies the target field. |
| -button | left\|middle\|right | Optional. Simulates one or more clicks with the mouse button indicated.<br>The default is left. |
| -numclicks | *number* | Optional. Simulates the number of clicks indicated.<br>The default is a single click. |
| -offsx | *x* | Optional. Specifies the x click coordinate within and relative to the target window. If you also include *FieldSpec* parameters, then the x coordinate is within and relative to the field specified. |
| -offsy | *y* | Optional. As for -offsx. |

**Examples**

To click the OK button:

```
sso click -label OK
```

To click coordinates 200(x)/300(y):

```
sso click -offsx 200 -offsy 300
```

To click the middle of the target window:

```
sso click -offsx ""  -offsy ""
```

**More information:**

push (see page 163)
type (see page 186)
Specifying Fields (see page 100)

## getbtnstate

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The getbtnstate extension returns the selection status of an option button.

When an option button is already selected, clicking it will cause the selection to be switched off. It is advisable to use getbtnstate to check the selection status of the button before clicking the button.

**Syntax**

```
sso getbtnstate FieldSpec
```

***FieldSpec***

Specifies the target field. For more information, see Specifying Fields in this chapter.

**Return Value**

If the button is selected, the return value is 1. If the button is not selected, the return value is 0.

**Example**

The following example determines if the MyButton button is selected and if it is not, selects it:

```
if { [sso getbtnstate -label MyButton ] == 0 } {

sso click -label MyButton

}
```

**Note:** If the MyButton button is disabled, the getbtnstate extension fails.

## getfield

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The getfield extension looks for the specified field in the visible area of the target window. If the specified field is found, it returns the field's contents.

Generally, the getfield extension is used with a text box (sometimes referred to as an edit field) to retrieve its contents. However, you can also use this extension with other types of fields. For example, if you use the getfield extension with a command button, the getfield extension returns the label (caption) of the button.

The interpreter disregards the -maxwords and -maxlines options if they have zero value.

In this case, the -maxlines default (one line) works. The option -maxwords does not have a default.

**Note:** -maxlines and -maxwords function are only in console mode. Either the -maxlines option or the -maxwords option can be used, but not both in the same getfield command.

**Syntax**

```
sso getfield [-maxlines number]|[-maxwords number][-timeout TimeValue]
             FieldSpec
```

| Key | Key Value | Description |
|---|---|---|
| *FieldSpec* | | Specifies the target field. The option -pos is relevant only in _MODE win.<br>If you do not specify a -pos option, the getfield extension defaults to -pos right. |
| -maxlines | *number* | Functional only in console mode.<br>Optional. The maximum number of lines to be retrieved.<br>The default is one line. |
| -maxwords | *number* | Functional only in console mode.<br>Optional. The maximum number of words to be retrieved. |

| Key | Key Value | Description |
|---|---|---|
| -timeout | *TimeValue* | Optional. Specifies the time to wait for the specified field to be found, in seconds or milliseconds depending upon the suffix used (s and sec designate seconds; ms and msec, milliseconds). When no suffix is used, the time is in seconds. If you do not specify the amount of time, the extension will wait 5 seconds. |

**Example 1**

This example assigns the contents of a text box labeled "User Name:" to a variable named text:

```
set text [sso getfield -label "User Name:"]
```

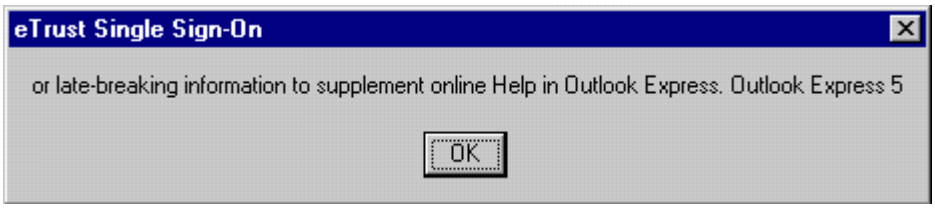
**Example 2**

This is an example on the -maxlines option.

We are looking for the specific text that provides "complementary" in the following window:


```
set _MODE console
sso window -title "Example.txt"
sso msgbox -msg [sso getfield -label "complementary" -maxlines 2]
```

As a result, the following message box appears:



**Example 3**

This is an example on the -maxwords option.

We are looking for the specific text that provides "document" in the text document from Example 2.

Issue the commands:

```
set _MODE console
sso window -title "Example.txt"
sso msgbox -msg [sso getfield -label "document" -maxwords 2]
```

As a result, the following message box appears:


**More information:**

setfield (see page 174)
type (see page 186)
Specifying Fields (see page 100)

## getlogin

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The getlogin extension retrieves the login variables for a specific application. It is useful when you need to log into a preliminary application before you can log into the target application. For example, suppose the end user has to log into telnet before accessing an e-mail program. The getlogin extension can also be used to retrieve and execute the script associated with a given application.

Generally, when the getlogin extension encounters problems such as an expired password or undefined login data, it displays a prompt asking for the user to enter a password.

By default the login variables are stored in the Tcl variables [application name]_LOGINNAME and [application name]_PASSWORD, and the contents of the script gets stored in [application name]_SCRIPT, unless –out option is used to specify a different prefix.

The getlogin extension will not function if the current user does not have permission to log into the specified application.

**Note:** This extension will fail if the client is in an offline state. The online status may be determined using the getmode extension. See the getmode extension documentation for further details.

**Syntax**

```
sso getlogin [-out prefix] -appname AppName
```

| Key | Key Value | Description |
|---|---|---|
| -appname | *AppName* | Specifies the name of the application for which the login variables should be fetched. |
| -exec | *yes*\|*no* | Optional. This has to be used in conjunction with –script. Default = yes. This is only used to prevent the application script from being executed when the –script option is used. |
| -script | *yes*\|*no* | Optional. Default = no. If yes, any script associated with that application will be executed, unless the value –exec is set to no. If used, it also stores the contents of the script in the Tcl variable, in the manner similar to that used by login variables. |

| Key | Key Value | Description |
|---|---|---|
| -out | *prefix* | Optional. Specifies the prefix for the new set of login variables. If you omit the prefix, each login variable name will include the specified application name as its prefix. |

**Examples**

The following example logs the user into CHECKING_ACCOUNT via telnet:

```
# get login variables for CHECKING_ACCOUNT:
 sso getlogin -appname CHECKING_ACCOUNT

# run telnet to CHECKING_ACCOUNT:
 sso run -path telnet

# fill in login data:
sso waittext -text "login:"
 sso type -text "$CHECKING_ACCOUNT_LOGINNAME{enter}"
sso waittext -text "Password:"
 sso type -text "$CHECKING_ACCOUNT_PASSWORD{enter}"
```

The following example will display a message box with the contents of the script associated with NotepadApp application.

```
sso getlogin –appname NotepadApp –script yes –exec no

sso msgbox –msg “Script for NotepadApp is $NotepadApp_SCRIPT”
```

## getmode

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The getmode extension determines the current state of the SSO Client. This is useful because you may create scripts that will take different courses of action according to the state of the SSO Client. For example, the script should not let a user change their password if the SSO Client is offline. The getmode extension can be used to determine if the client is online before proceeding with a password change.

The states of the SSO Client that will be returned:

_CLIENT_STATE_OFFLINE - Whether the client is operating in offline mode.

_CLIENT_STATE_SIGNING_ON - Whether a sign-on is in progress.

_CLIENT_STATE_SIGNED_ON - Whether the user is signed on to the SSO Server.

_CLIENT_STATE_REAUTHENTICATING - Whether a reauthentication is in progress.

_CLIENT_STATE_SIGNING_OFF - Whether a sign-off is in progress.

_CLIENT_STATE_LOCKED - Whether the workstation is locked locked.

A return value of 0 indicates false.

A return value of 1 indicates true.

**Syntax**

```
sso getmode
```

**Example**

```
#set the client state variables
sso getmode

# if the client is not currently offline...
if { !$_CLIENT_STATE_OFFLINE }{
    # Notify the server about an application password change
    sso notify -appname $_APPNAME -event pwdchange -status 0
}
```

## getmsgtext

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The getmsgtext extension returns the text of a specified message box. When the box is found, the extension returns the text of the message from it. This extension is useful when you know that the previous actions of the script might result in a message from the application and you want to know the contents of this message.

If a window with the specified caption fails to appear within the time-out period, the extension fails.

### Syntax

```
sso getmsgtext WindowSpec
```

***WindowSpec***

Specifies the window containing the text to be retrieved. For more information, see Specifying Windows in this chapter.

### Return Value

The extension returns a text string from the specified message box. If the message box is not found, the extension returns an empty string.

### Example

This example retrieves text from a message box that is expected to appear after a command has executed and then uses the text retrieved as a branching criterion:

```
...
 sso click -label OK

# check result of action
set msgtext [sso getmsgtext -title "msgbox_caption"]
switch -glob [string toupper $msgtext] {
   *SUCCESS* {
      # action succeeded
      ...
      }
   *FAIL* {
      # action failed
      ...
      }
      }
```

### More information:

getfield (see page 114)
window (see page 192)

## getplatform

### Compatibility

Windows 2000/XP/2003.

### Purpose

The getplatform extension retrieves the type of OS.

**Syntax**

```
sso getplatform
```

**Return Value**

These are the possible return values:

- Windows XP
- W2K WS - Win2000 professional (workstation)
- W2K Srv - Win2000 server
- W2K DC - Win2000 domain controller (advanced server)

## getscrape

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The getscrape extension does a screen scrape and returns the contents. This extension is useful for debugging scripts.

**Syntax**

```
sso getscrape
```

**Example**

To display the result of the screen scrape in a message box:

```
sso msgbox -msg [sso getscrape]
```

**More information:**

refresh (see page 165)

## hllapi_connect

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The hllapi_connect extension establishes a HLLAPI-connection to an already running 3270 session. This session becomes the current session and all the HLLAPI extensions that follow will relate to this session.

In most emulators, you can connect to a session even if it does not have an open emulation window on the screen (that is, without a previous run extension). This is useful for background scripts that are intended to serve an application program rather than a human operator.

**Syntax**

```
sso hllapi_connect -session SessionName
```

| Key | Key Value | Description |
|---|---|---|
| -session | *SessionName* | The one-character code that identifies the session. |

**Example**

This example starts a 3270 emulation and HLLAPI-connects to session a:

```
sso run -path pcsws.exe -args
"C:\\Personal\\Communications\\PRIVATE\\ca.WS"
sso hllapi_connect -session a
...
```

**More information:**

_HLLAPI_FUNC_NO (see page 205)
_HLLAPI_RC (see page 205)
_MODE (see page 208)
hllapi_disconnect (see page 122)

## hllapi_disconnect

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The hllapi_disconnect extension terminates the HLLAPI connection to the current 3270 session that was connected by the last hllapi_connect extension.

You should put the command sso hllapi_disconnect at the end of every script in which a hllapi_connect extension appears. There is no need to issue hllapi_disconnect for every hllapi_connect that was issued somewhere in the script, because hllapi_connect performs an automatic disconnect for the previous connect.

### Syntax

```
sso hllapi_disconnect
```

### Example

This example shows HLLAPI connecting a session and then HLLAPI disconnecting it:

```
sso hllapi_connect –session a
...
...
 sso hllapi_disconnect
```

### More information:

_HLLAPI_FUNC_NO (see page 205)
_HLLAPI_RC (see page 205)
_MODE (see page 208)
hllapi_connect (see page 121)

## hllapi_getcursor

### Compatibility

Windows 2000/XP/2003.

### Purpose

The hllapi_getcursor extension returns the current location of the cursor in the current 3270 session. The variables _ROW and _COL are set to the current cursor location, which is also the return value.

### Syntax

```
sso hllapi_getcursor
```

### Return Value

The current location of the cursor is the return value, formatted as *rr ccc*. *rr* is the two-digit row number, and *ccc* is the three-digit column number (both padded with leading zeroes).

The extension also sets the variable _ROW to the cursor's current row number and _COL to the current column.

### Examples

This example gets the location of the cursor in the current 3270 session and displays two message boxes: the first with the cursor location as a formatted string, and the second with it as a row number and a column number:

```
set csr_loc [sso hllapi_getcursor]
 sso msgbox -msg "The current cursor location \
                 is $csr_loc"
sso msgbox -msg "The cursor is located \
                 at row $_ROW and column $_COL"
```

### More information:

_COL (see page 201)
_HLLAPI_FUNC_NO (see page 205)
_HLLAPI_RC (see page 205)
_MODE (see page 208)
_ROW (see page 212)
hllapi_getfield (see page 124)
hllapi_setcursor (see page 127)
hllapi_setfield (see page 128)
hllapi_waittext (see page 132)

## hllapi_getfield

### Compatibility

Windows 2000/XP/2003.

### Purpose

The hllapi_getfield extension returns the contents of a specified field in the current session.

Although you can specify only part of the caption, and use the -pos parameter, you should use the full caption, in order to reduce the possibility that another field will match the caption in the command.

Once the specified caption is located, the contents of the next field are returned as the result of the extension, whether or not the next field is an input field.

The hllapi_getfield extension sets the variables _ROW and _COL to the row and column where the specified caption was found. If the specified caption isn't found, _ROW and _COL are each set to -1.

The parameters -from* and -pos* cannot be used together.

**Syntax**

You can use either one of the following syntax versions:

```
sso hllapi_getfield -label Caption[-fromx x -fromy y]
```

or

```
sso hllapi_getfield -label Caption [-posx x -posy y]
```

| Key | Key Value | Description |
|---|---|---|
| -label | *Caption* | Specifies the caption (label) of the target field. The caption is assumed to be located immediately before the target field. You do not have to specify the whole caption; the first letter of the caption will suffice if it is unique. The search is case-sensitive |
| -fromx | *x* | Optional. Specifies the row number in the 3270 screen where the caption search should begin. Not specifying -fromx is equivalent to specifying -fromx 1 (that is, to begin the search at the top left of the screen).<br>The default is 1. |
| -fromy | *y* | Optional. Specifies the column number in the 3270 screen where the caption search should begin. Not specifying -fromy is equivalent to specifying -fromy 1 (that is, to begin the search at the top left of the screen).<br>The default is 1. |
| -posx | *x* | Optional. Specifies a specific location (row) of the caption, or the location of the field's contents. If a non-null caption is specified, a verification of the caption is performed for the specified location. This is a safe getfield, because it eliminates the chance that the caption found is not the required one but a similar string somewhere else on the screen. The contents of the field at the specified location are retrieved without a search or verification of the contents when a null caption ("") is specified |
| -posy | *y* | Optional. Specifies a specific location (column) of the caption, or the location of the field's contents. If a non-null caption is specified, a verification of the caption is performed for the specified location. This is a safe getfield, because it eliminates the chance that the caption found is not the required one but a similar string somewhere else on the screen. The contents of the field at the specified location are retrieved without a search or verification of the contents when a null caption ("") is specified |

**Return Value**

The contents of the specified field are the return value.

**Examples**

This example retrieves the contents of Userid from the pcomm4 logon panel:

```
set uid [sso hllapi_getfield -label "Userid" \
-posx 14 -posy 5]
 sso msgbox -msg "$uid"
```

**More information:**

_MODE (see page 208)
_TIMEOUT (see page 216)
_ROW (see page 212)
_COL (see page 201)
_HLLAPI_FUNC_NO (see page 205)
_HLLAPI_RC (see page 205)
hllapi_getscreen (see page 126)
hllapi_setfield (see page 128)
hllapi_waittext (see page 132)

## hllapi_getscreen

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The hllapi_getscreen extension returns the contents of the screen in the currently connected 3270 session. The length of the returned string varies according to the model type of the currently connected 3270 session.

**Syntax**

```
sso hllapi_getscreen [-attrb y|n]
```

| Key | Key Value | Description |
|---|---|---|
| -attrb | y | Specifies that the field attributes contained in the 3270 screen are to be left unchanged, so you can use them to get information about fields. A field attribute is one byte, whose value is X'C0' or greater. For a complete description of the available attributes and their meanings, refer to the IBM 3270 documentation, or to your 3270 emulator documentation. |

| Key | Key Value | Description |
|---|---|---|
| | n | Not specifying -attrb causes the field attribute bytes to be converted to blanks (just as the user sees them on a regular 3270 terminal).<br><br>The default is n. |

**Return Value**

The current content of the screen is the return value, formatted as a long string concatenating all the screen rows.

**Example**

This example shows the contents of a 3270 screen (without attributes) in a message box:

```
set screen [sso hllapi_getscreen]
 sso msgbox -msg "$screen"
```

**More information:**

_MODE (see page 208)
_HLLAPI_FUNC_NO (see page 205)
_HLLAPI_RC (see page 205)
hllapi_getfield (see page 124)

## hllapi_setcursor

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The hllapi_setcursor extension positions the cursor at a specified location in the currently connected 3270 session. You should use the hllapi_setfield extension rather than a combination of the hllapi_setcursor and hllapi_type extensions whenever possible.

**Syntax**

```
sso hllapi_setcursor -offsx x -offsy y
```

| Key | Key Value | Description |
|---|---|---|
| -offsx | *x* | Specifies a row number in the 3270 screen. |
| -offsy | *y* | Specifies a column number in the 3270 screen. |

**Example**

The example sets the cursor to row 13 column 5:

```
sso window -title "Session A"
sso hllapi_setcursor -offsx 13 -offsy 5
```

**More information:**

_MODE (see page 208)
_HLLAPI_FUNC_NO (see page 205)
_HLLAPI_RC (see page 205)
hllapi_getcursor (see page 123)
hllapi_setfield (see page 128)
hllapi_type (see page 130)

## hllapi_setfield

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The hllapi_setfield extension inserts a text string into a specified input field in the current session.

Although a partial caption can be specified, it is advisable to use the full caption, in order to minimize the chances that another field will match the caption search criteria.

The hllapi_setfield extension sets the variables _ROW and _COL to the row and column where the specified caption was found. If the specified caption isn't found, _ROW and _COL are each set to -1.

The hllapi_setfield extension cannot be used to enter special characters. To do so, you have to use the hllapi_type extension.

The parameters -from* and -pos* cannot be used together.

**Syntax**

You can use either one of the following syntax versions:

```
sso hllapi_setfield \
-label Caption -value Value [-fromx x -fromy y]
```

or

```
sso hllapi_setfield \
-label Caption -value Value [-posx x -posy y]
```

| Key | Key Value | Description |
|---|---|---|
| -label | *Caption* | Specifies the caption (label) of the target field. The caption is assumed to be located immediately before the target field. You do not have to specify the whole caption; the first letter of the caption will suffice if it is unique. The search is case-sensitive |
| -value | *Value* | Specifies the new text value for the target field. The length of the text is limited by the length of the target field. |
| **-**fromx | *x* | Optional. Specifies the row number in the 3270 screen where the caption search should begin. Not specifying -fromx is equivalent to specifying -fromx 1 (that is, to begin the search at the top left of the screen).<br>The default is 1. |
| **-**fromy | *y* | Optional. Specifies the column number in the 3270 screen where the caption search should begin. Not specifying -fromy is equivalent to specifying -fromy 1 (that is, to begin the search at the top left of the screen).<br>The default is 1. |
| **-**posx | *x* | Optional. Specifies either the specific location (row) of the caption or the specific location of the target field. If a null caption argument is used (""), no search or verification is performed and the field at the specified location is set to value. If a non-null caption argument is specified, the extension verifies the presence of the specified caption at the location. This is a safe setfield, because it ensures that the caption located is the correct one and not a similar string someplace else on the screen. |

| Key | Key Value | Description |
|---|---|---|
| -posy | *y* | Optional. Specifies either the specific location (column) of the caption or the specific location of the target field. If a null caption argument is used (""), no search or verification is performed and the field at the specified location is set to value. If a non-null caption argument is specified, the extension verifies the presence of the specified caption at the location. This is a safe setfield, because it ensures that the caption located is the correct one and not a similar string someplace else on the screen. |

**Example**

The example inserts a login name in the appropriate input field.

```
sso hllapi_setfield -label "Userid" -value "Name"
```

**More information:**

_MODE (see page 208)
_TIMEOUT (see page 216)
_ROW (see page 212)
_COL (see page 201)
_HLLAPI_FUNC_NO (see page 205)
_HLLAPI_RC (see page 205)
hllapi_getfield (see page 124)
hllapi_type (see page 130)
hllapi_waittext (see page 132)

## hllapi_type

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The hllapi_type extension simulates typing done by a user to the current 3270 session.

To type special characters, use special character mnemonics surrounded by curly braces. For example:

```
sso hllapi_type -text "{tab}abc{tab}"
```

In order to type the @ sign, type @@.

There is no need to use a hllapi_waitsys extension before the hllapi_type extension because the hllapi_type extension has an internal procedure for the wait function.

**Syntax**

```
sso hllapi_type -text String
```

| Key | Key Value | Description |
|---|---|---|
| -text | *String* | Specifies one or more characters. Use special mnemonics for special characters, such as {enter} for the Enter key. |

**Examples**

To type a string:

```
sso hllapi_type -text "Peterson, Terri"
```

To tab to the next field:

```
sso hllapi_type -text "{tab}"
```

To type a variable value, type one of the following:

```
sso hllapi_type -text "$_PASSWORD"
```

or

```
sso hllapi_type -text $_PASSWORD
```

To type a string that includes a dollar sign:

```
sso hllapi_type -text {$22}
```

To type "hello" and press Enter:

```
sso hllapi_type -text "hello{enter}"
```

**More information:**

_MODE (see page 208)
_HLLAPI_FUNC_NO (see page 205)
_HLLAPI_RC (see page 205)
hllapi_setcursor (see page 127)
hllapi_setfield (see page 128)
3270 Keyboard Mnemonics (see page 227)

## hllapi_waitsys

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The hllapi_waitsys extension waits until the terminal session is ready to accept keystrokes, or until the timeout value is reached.

Usually there is no need to use waitsys, because it is an internal procedure of most of the HLLAPI extensions. However, in some situations, such as when combining a Windows application and a 3270 application in the same script, this extension is necessary.

**Syntax**

```
sso hllapi_waitsys
```

**Example**

To type Enter, wait, and then display a message box:

```
sso hllapi_type "{enter}"
sso hllapi_waitsys
sso msgbox -msg "Now you may start working"
```

**More information:**

_MODE (see page 208)
_TIMEOUT (see page 216)
_HLLAPI_FUNC_NO (see page 205)
_HLLAPI_RC (see page 205)
hllapi_waittext (see page 132)

## hllapi_waittext

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The hllapi_waittext extension waits for specified text to appear on the current session's screen. If the text already appears on the screen, script execution continues without a wait period.

The last step of hllapi_waittext execution sets the variables _ROW and _COL to the location where the specified text was found on the screen.

The parameters -from* and -pos* cannot be used together.

**Syntax**

You can use either of the following syntax versions:

```
sso hllapi_waittext -text String [-fromx x -fromy y]
```

or

```
sso hllapi_waittext -text String [-posx x -posy y]
```

| Key | Key Value | Description |
|---|---|---|
| -text | *String* | Specifies one or more characters. Use special mnemonics for special characters, such as {enter} for the Enter key. |
| **-**fromx | *x* | Optional. Specifies the row number in the 3270 screen where the caption search should begin. Not specifying -fromx is equivalent to specifying -fromx 1 (that is, to begin the search at the top left of the screen).<br>The default is 1. |
| **-**fromy | *y* | Optional. Specifies the column number in the 3270 screen where the caption search should begin. Not specifying -fromy is equivalent to specifying -fromy 1 (that is, to begin the search at the top left of the screen).<br>The default is 1. |
| **-**posx | *x* | Optional. Specifies a specific location (row) where the text is to appear. The -posx key verifies that the text is at the specified location. This is a safe waittext, because it ensures that the text string located is the correct one and not a similar string someplace else on the screen. A null text is not allowed. |
| **-**posy | *y* | Optional. Specifies a specific location (column) where the text is to appear. The -posy key verifies that the text is at the specified location. This is a safe waittext, because it ensures that the text string located is the correct one and not a similar string someplace else on the screen. A null text is not allowed. |

**Return Value**

The extension returns the completion code.

**Example**

```
sso text_var [sso hllapi_waittext -text "or"]
sso msgbox -msg $text_var
```

**More information:**

_MODE (see page 208)
_TIMEOUT (see page 216)
_ROW (see page 212)
_COL (see page 201)
_HLLAPI_FUNC_NO (see page 205)
_HLLAPI_RC (see page 205)
hllapi_getfield (see page 124)
hllapi_setfield (see page 128)
hllapi_waitsys (see page 132)

## html_browse

**Compatibility**

MS Internet Explorer 4.0 and higher.

**Purpose**

The html_browse extension opens an Internet Explorer browser window and navigates to either the current home page or the page specified by the -location option.

**Syntax**

```
sso html_browse [-location "targetURL"]
```

It is good practice to use quote marks around the target URL.

| Key | Key Value | Description |
|---|---|---|
| -location | *targetURL* | (Optional) Specifies which URL Internet Explorer will open. If this value is not specified or an empty string is specified, the html_browse will navigate to the current home page. |

**Example**

To connect to an Internet Explorer window with the Atlantic City Press home page, use:

```
sso html_browse
sso html_navigate –location "http://www.pressplus.com/"
```

The following example will achieve the same effect as the above example:

```
sso html_browse -location "http://www.pressplus.com/"
```

You can specify an empty value using just the quotation marks or you can simply remove the -location key altogether. For example the following two example will work in the same way:

```
sso html_browse -location ""
```

or

```
sso html_browse
```

## html_connect

**Compatibility**

MS Internet Explorer 4.0 and higher.

html_connect does not function with a browser instance launched inside a Citrix MetaFrame ICA client, as the ICA client does not provide all of the standard Windows hooks. An alternative in this case is to instead use the *html_browse* extension to launch the browser, as *html_browse* functions correctly in a MetaFrame ICA environment.

**Purpose**

The html_connect extension establishes a connection to an already opened Internet Explorer browser window with the specified window title. The value you use in -title should uniquely identify the Web page.

**Syntax**

```
sso html_connect –title targetText
```

| Key | Key Value | Description |
|---|---|---|
| -title | *targetText* | Indicates that the specified title text will be treated as part of the text in the window title bar that SSO will search for. This also allows for an exact match. |

**Example**

To connect to an Internet Explorer window with the Atlantic City Press home page, use:

```
sso html_connect –title  "PressPlus"
```

## html_disconnect

**Compatibility**

MS Internet Explorer 4.0 and higher.

**Purpose**

The html_disconnect extension disconnects from the active Internet Explorer browser window. html_disconnect will not take any key or key-value.

**Syntax**

```
sso html_disconnect
```

## html_getfield

**Compatibility**

MS Internet Explorer 4.0 and higher.

**Purpose**

The html_getfield extension returns the contents of a specified field in the current browser window. The edit box is identified by a label in the browser window and an ordinal number. The extension locates the label, counts the edit boxes that appear after the text, and retrieves whatever text is found in the *n*th edit box.

If the fieldIndex value is 1, then the extension searches the browser window for the first edit box that contains the designated label.

Note that the extension searches the browser window line by line from top to bottom and from left to right on each line.

**Syntax**

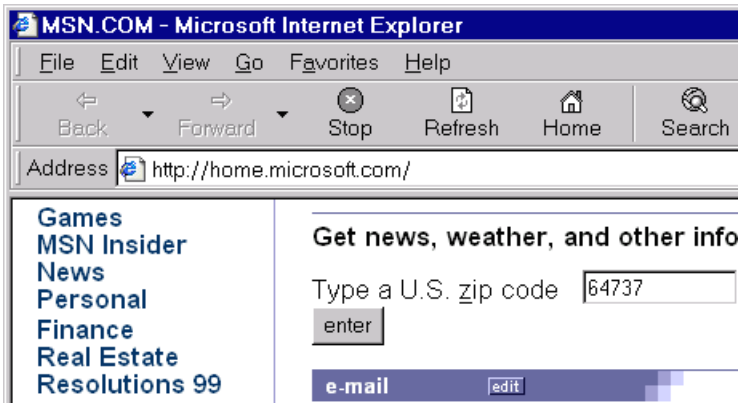
```
sso html_getfield -label labelText [-ord fieldIndex]
```

| Key | Key Value | Description |
|---|---|---|
| -label | *labelText* | The text, in the target field, which serves as the starting point of the search for the target field. |
| -ord | *fieldIndex* | The ordinal number of the target field that follows the label text. If the *fieldIndex* value is 1, then the extension searches the browser window for the first edit box that contains the specified text.<br><br>0 and 1 can be used interchangeably when using this option.<br><br>The default value is 1. |

**Example**

The following command retrieves the contents of the edit box with the request to type a U.S. zip code label:

```
sso html_getfield -label "code"
```




**Note:** Don't use labelText values that contain underlined letters, like the z in zip. They cannot be represented by conventional alphanumeric characters and will not be recognized in the browser window.

To retrieve the hotmail logon name, use one of the following commands:

```
sso html_getfield -label "Email address"
```

```
sso html_getfield -label "Email address" -ord 1
```

To retrieve the contents of the hotmail password field, you can use either of the following:

```
sso html_getfield -label "Password" -ord 2
```

```
sso html_getfield -label "Email address" -ord 2
```

**Note:** Since the html_getfield extension searches line by line, it doesn't make any difference if the labelText value is Hotmail Password or Logon Name. However, it is essential to provide the correct ordinal number.

**More information:**

html_setfield (see page 146)

## html_getframesnum

**Compatibility**

MS Internet Explorer 4.0 and higher.

**Purpose**

The html_getframesnum extension retrieves the number of frames on an active Internet Explorer browser window.

**Syntax**

```
sso html_getframesnum
```

**Example**

The following displays the number of the frames in the active window:

```
sso msgbox -msg [sso html_getframesnum]
```

## html_getselecteditem

**Compatibility**

MS Internet Explorer 4.0 and higher.

**Purpose**

The html_getselecteditem extension complements the html_selectitem extension. While html_selectitem selects the item in the combobox, html_getselecteditem will retrieve the selected text of that item.

**Syntax**

```
sso html_getselecteditem –label labelName [-ord fieldIndex]
```

| Key | Key Value | Description |
|---|---|---|
| -label | labelText | The text, in the browser window, which serves as the starting point for the search for the target listbox. |
| -ord | fieldIndex | The ordinal number of the list box or combo box that follows the label text. If the fieldIndex value is 1, then the extension searches the browser window for the first edit box that contains the specified text.<br><br>The default value is 1. |

**Example**

For example we have a html combobox with a label Search

To retrieve the current text on it:

```
sso msgbox -msg [sso html_getselecteditem -label "Search" -ord 1]
```

**More information:**

html_selectitem (see page 144)
html_getfield (see page 136)
selectitem (see page 169)

## html_grabpage

**Compatibility**

MS Internet Explorer 4.0 and higher.

**Purpose**

The html_grabpage extension invokes an active Internet Explorer browser window.

**Syntax**

```
sso html_grabpage
```

**Example**

```
sso html_grabpage
```

## html_link

**Purpose**

The html_link extension can be used to click on hyperlinks that are defined by the Action: tag or client side image maps. An Action: tag that only defines a fragment identifier is not treated as a hyperlink and is not counted when searching for the target hyperlink.

**Note**: This extension is not supported for Java applications.

**Syntax**

```
sso html_link -label labelText|-name NameText [-ord fieldIndex]
```

**-label**

Text in the browser window that serves as the starting point for the search for the target hyperlink.

**-ord**

Ordinal number of the target hyperlink that follows the label text. If the text of -label designates a hyperlink, it is counted as hyperlink number 1.

**Default:** 1

**-name**

Name of the target hyperlink. If the hyperlink is defined by an Action: tag the name can be one of the following:

- The text content of the Action: tag
- If there is no text content and there is an <img> tag within the enclosing Action: tag, the name is the alt attribute value, if one exists.
- If there is no text content and there is no alt attribute in the enclosed <img> tag, then this hyperlink cannot be identified by the -name option.

If the hyperlink is defined by a client side image map the name of the tag can be one of the following:

- The value of the alt attribute of the <area> tag that defines the hyperlink.
- If there is no alt attribute, then this hyperlink cannot be identified by the -name option.

**Notes:**

- You cannot use the -label and the -name options together.
- To display the ALT value of a hyperlink in a tooltip, you must hover the cursor over the hyperlink.

## html_navigate

**Compatibility**

MS Internet Explorer 4.0 and higher.

**Purpose**

The html_navigate extension opens a new site in an Internet Explorer browser at a specified URL.

**Syntax**

```
sso html_navigate -location URL [-grab Y|N]
```

| Key | Key Value | Description |
|---|---|---|
| -location | *URL* | Specifies the URL to be linked to. |
| -grab | Y\|N | Connects to the site and recognizes all controls on the specified URL.<br>The default is Y. |

## html_push

**Compatibility**

MS Internet Explorer 4.0 and higher.

**Purpose**

The html_push extension pushes a specified command button in the browser window. When the fieldIndex value is 1 (the default), then the extension searches the browser window for the first button that contains the designated label.

When the fieldIndex value is a positive integer, the extension locates the label text in the browser window, counts the pushable controls that appear after the text, and pushes the *n*th control.

You can use either the -label option or the -name option, you can not use these options together.

**Note:** The html_push extension cannot be used to click URLs in the browser window.

**Syntax**

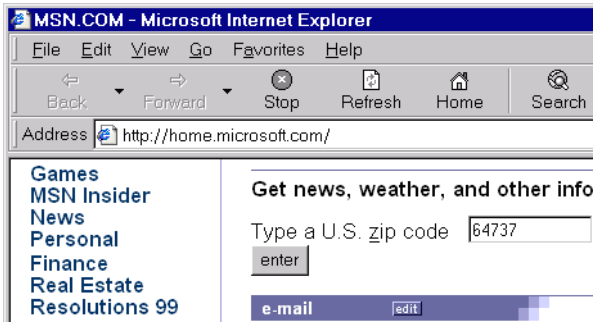
```
sso html_push -label labelText/-name NameText [-ord fieldIndex]
```

| Key | Key Value | Description |
|---|---|---|
| -label | *labelText* | The text in the browser window that serves as the starting point for the search for the target field. |

| Key | Key Value | Description |
|---|---|---|
| -ord | *fieldIndex* | The ordinal number of the target field that follows the label text or name. If the *fieldIndex* value is 1, then the extension searches the browser window for the first edit box that contains the specified text.<br><br>0 and 1 can be used interchangeably when using –ord option.<br><br>The default value is 1. |
| -name | *NameText* | The name of the target button or target field. |

**Example**

In the following example:




you can push the enter button by using:

```
sso html_push -label "hotmail" -ord 1
```

**More information:**

click (see page 111)
push (see page 163)

## html_search

**Compatibility**

MS Internet Explorer 4.0 and higher.

**Purpose**

The html_search extension puts text in the search mechanism of an active Internet Explorer browser window and pushes the search button.

**Syntax**

```
sso html_search -text Text -engine pageName -engineURL search URL
```

| Key | Key Value | Description |
|---|---|---|
| -text | *Text* | Specifies the text that is being searched for. |
| -engine | *pageName* | Specifies the search engine web page. This is not case-sensitive. You can only use one of the following search engine web pages with this key: Google, AltaVista, AOL, Excite, HotBot, Lycos, Yahoo!, MSN |
| -engineURL | *search URL* | Specifies the search engine web page for a search engine not listed for -engine.<br>To get the URL string for this key:<br>1. Go to the search engine web page.<br>2. Perform a search, for example "frogs".<br>3. From the results page, copy the URL in the address line.<br>4. Replace the original search text with %s, for example replace 'frogs' with '%s' |

**Example**

To find all the information about frogs on Google (or one of the other specified search engines) use:

```
sso html_search -text frogs -engine "Google"
```

To find all information about frogs on a search engine not supported by –engine, for example 'Looksmart,' use:

```
sso html_search -text frogs -engineUrl
http://search.looksmart.com/p/search?tb=dir&qt=%s
```

## html_selectitem

**Compatibility**

MS Internet Explorer 4.0 and higher.

**Purpose**

The html_selectitem extension selects an item in a specified list box or combo box in the browser window.

The list box is designated by a text string in the browser window and an ordinal number. After locating the *n*th listbox after the label text according to the fieldIndex value, the extension then selects the *n*th item in the list box according to the value of itemIndex.

Note that the extension searches the browser window line by line from top to bottom and from left to right in each line.

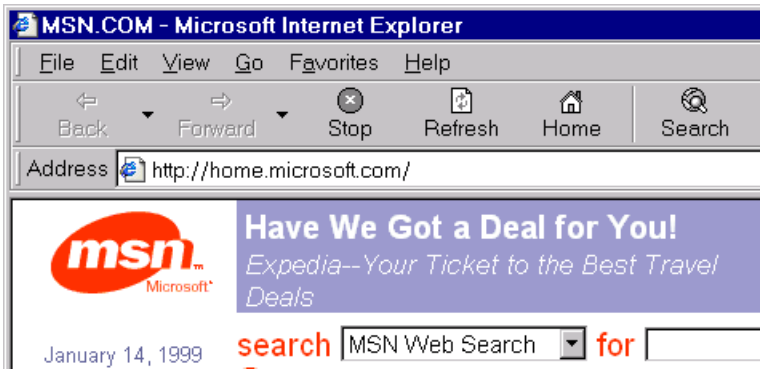
**Syntax**

```
sso html_selectitem -label labelName [-ord fieldIndex]\
                    -index itemIndex | -item itemName
```

| Key | Key Value | Description |
|---|---|---|
| -label | *labelText* | The text, in the browser window, which serves as the starting point for the search for the target listbox. |
| -ord | *fieldIndex* | The ordinal number of the list box or combo box that follows the label text. If the *fieldIndex* value is 1, then the extension searches the browser window for the first edit box that contains the specified text.<br>0 and 1 can be used interchangeably when using –ord option.<br>The default value is 1. |
| -index | *itemIndex* | The index number of the item in the listbox |
| -item | *itemName* | Specifies the full name or the first part of the name of the item to be selected. |

**Example**

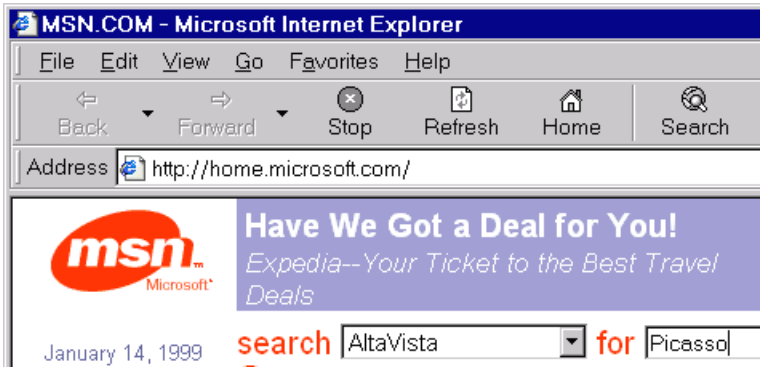
If you want to search for Picasso from the following browser window and use the AltaVista search engine:




use the following commands:

```
sso html_selectitem -label search -ord 1 -index 2
sso html_setfield -label for -ord 1 -value Picasso
```

This gives the following result:




**More information:**

html_getframesnum (see page 138)
selectitem (see page 169)

## html_setfield

### Compatibility

MS Internet Explorer 4.0 and higher.

### Purpose

The html_setfield extension sets a value in a specified edit box in the browser window. The edit box is designated by a label and an ordinal number. The extension locates the text, counts the edit boxes that appear after the text, and enters the designated value into the *n*th edit box.

Note that the extension searches the browser window line by line from top to bottom and from left to right in each line.

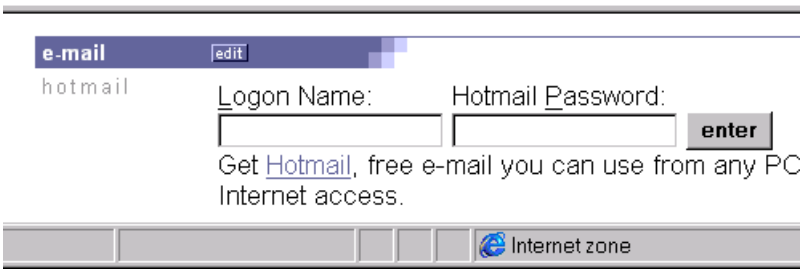
### Syntax

```
sso html_setfield -label labelName [-ord fieldIndex]\
                  -value newValue
```

| Key | Key Value | Description |
|---|---|---|
| -label | *labelText* | The text, in the browser window, which serves as the starting point for the search for the target field. |
| -ord | *fieldIndex* | The ordinal number of the field that follows the label text. If the *fieldIndex* value is 1, then the extension searches the browser window for the first edit box that contains the specified text.<br><br>0 and 1 can be used interchangeably when using –ord option.<br><br>The default value is 1. |
| -value | *newValue* | The value to be entered into the target edit box. |

### Example

To enter the Hotmail logon name and password and then press the enter button in the window below:



use the following commands:

```
sso html_setfield -label "Logon Name" -value markl
sso html_setfield -label "Hotmail Password" -ord 2 \
                  -value hj41B43
sso html_push -label "enter" -ord 1
```

**More information:**

html_getfield (see page 136)
setfield (see page 174)

## inputbox

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The inputbox extension displays a dialog box containing a prompt, a text box for input from the user, and OK-Cancel command buttons.

The dialog box size is determined by the length of the text in -prompt promptText. The window size is not affected by the length of the specified title, and titles that are too long for the box are truncated.

If the Cancel command button is pushed, then script execution stops, unless the _ERRORMODE setting is resume or msg.

**Syntax**

```
sso inputbox -prompt promptText [-default defaultText]\
 [-title titleText] [-input num|all] [-x x] [-y y]
```

| Key | Key Value | Description |
|---|---|---|
| -prompt | *promptText* | Specifies the text that is displayed in the dialog, above the edit box (input field). |
| -default | *defaultText* | Optional. Specifies the text that appears in the edit box of the input window. This text can be replaced by the user. |
| -x | x | Optional. The x coordinate on the screen of the dialog. Default value -1 which means center the dialog on the screen. Any negative value defaults to -1. Any positive value that is too large will automatically reposition so that the entire dialog is visible on the screen. |
| -y | y | Optional. The y coordinate on the screen of the dialog. See the description for -x. |

| Key | Key Value | Description |
|---|---|---|
| -title | *titleText* | Optional. Specifies the title of the dialog. The default is eTrust SSO. |
| -input | num\|all | Optional. -input num indicates that only numeric input will be accepted.<br>The default is all (any character input). |

**Return Value**

The inputbox extension returns the text string typed by the user.

**Example**

The following example displays a dialog box that asks the user for the name of a host and then invokes telnet for that host:

```
set host [sso inputbox \
                -prompt "Enter the name of the host:"
if {$host == ""} {
   exit
}
# run telnet for the selected host
sso run -path wktelnet.exe -args $host
```

**More information:**

askyesno (see page 107)
msgbox (see page 152)

## lockinput

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The lockinput extension locks input to the workstation from the keyboard and the mouse. The station remains locked until the script stops running or the unlockinput extension is used.

All keyboard and mouse input is blocked, except to the interpreter itself so that if you put in another sso command, such as sso msgbox, the user would be able to use the keyboard and mouse to interact with the message box.

Some SSO Tcl functions such as "sso click" and "sso type" override the lockinput function so that the script can simulate user input on the screen. When these overriding functions are being executed there is a small window of opportunity for the end user to interfere with the script. For example, if a user clicks a mouse button it could move the focus of the cursor. For this reason, when the lockinput is active, it is generally better practice to use the "sso setfield" command instead of "sso click" and "sso type".

Unfortunately "sso setfield" is not appropriate for every applicaiton, for example Java applications and terminal emulator programs will not accept the "sso setfield" command.

To minimise the chance of the end user inadvertently affecting the logic in the script by clicking the mouse or typing a keystroke at the exact same time as the "sso type" command executes, you may want to display a Messagebox dialog that tells the end user to wait until the script has completed.

**Syntax**

```
sso lockinput
```

**Example**

The following example would lock user input for 10 seconds:

```
sso lockinput
sso sleep -time 10
```

## logoff

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The purpose of the Logoff extension is to terminate the SSO user's SSO session. This extension terminates the Token that is created when the SSO user authenticates and shuts down SSO correctly. This extension can also be used to log off the current Windows user's logon session.

**Syntax**

sso logoff [-system y|n] [-force y|n]

| Key | Key Value | Description |
|---|---|---|
| -system | y\|n | If this flag is set to y, the Windows user will be logged off from the system. The default value for this flag is n. |
| -force | y\|n | If this flag is set to y, all the processes running in the logon session of the Windows user will be forced to terminate. This can cause application to lose data. The default value for this flag is n. |

**Example**

The following example logs the current user off from the SSO Client, therefore terminating their SSO session.

```
sso logoff
```

The following example logs the current user off from the SSO Client, therefore terminating their SSO session and logs them off their Windows session.

```
sso logoff -system y
```

The following example logs the current user off from the SSO Client, therefore terminating their SSO session and logs them off their Windows session. All the running processes will be forced to terminate.

```
sso logoff -system y -force y
```

## menu

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The menu extension selects a menu item from the menu of the target window. The extension fails if the specified menu item is disabled.

menuItem is written as a string with item elements separated by slashes. For example, the Paste item in the Edit menu would be designated Edit/Paste, and Find Next in Notepad's Search menu would be Edit/Find Next.

Even when the -itemexact menuItem option is used, text matching ignores an ellipsis (¼) in menu item names.

You can also select menu items with the type extension, using access keys (like Alt+F and then X for File/Exit) or keyboard shortcuts (like Ctrl+S for Save).

**Syntax**

```
sso menu -item|-itemglob|-itemexact menuItem
```

| Key | Key Value | Description |
|---|---|---|
| -item | *menuItem* | Indicates that eTrust SSO searches for the specified string as the first part of the menu item. This also allows the string to be the exact menu item name. |
| -itemglob | *menuItem* | Indicates that the string provided is actually a pattern formatted in the globbing format. eTrust SSO searches for a menu item that matches the pattern specified by in the argument. |
| -itemexact | *menuItem* | Indicates that the string provided in the caption argument represents the *complete* string that eTrust SSO searches for in the window's menu. |

**Example**

The example selects Save As from the target window's menu.

```
sso menu -item "File/Save As"
```

**More information:**

type (see page 186)

## msgbox

### Compatibility

Windows 2000/XP/2003.

### Example

The msgbox extension displays a message box with text, a command button or buttons, and an optional severity icon at the user's workstation. The info, warning, and error icons are mutually exclusive. If no value is specified, no icon appears in the message box.

After the message box is displayed, script execution is suspended until the user selects one of the buttons.

The -button option determines the button set displayed in the box. If the option is not used, the default is a single OK button.

The message box size is determined by the length of the text that appears in the message box. The window size is not affected by the length of the specified caption and long captions are truncated.

### Syntax

```
sso msgbox [-title titleText] [-icon info|warn|error] \
      [-buttons okcancel|yesnocancel|abortretryignore|ok]\
      -msg msgText [-x x] [-y y]
```

| Key | Key Value | Description |
|---|---|---|
| -msg | *msgText* | The text that will appear in the message box. |
| -x | x | Optional. The x coordinate on the screen of the dialog. Default value -1 which means center the dialog on the screen. Any negative or blank value defaults to -1. Any positive value that is too large will automatically reposition so that the entire dialog is visible on the screen. |
| -y | y | Optional. The y coordinate on the screen of the dialog. See the description for -x. |
| -title | *titleText* | Optional. Sets the text of the message box title.<br>The default is eTrust Single Sign-On. |
| -icon | info | Optional. Displays an info icon (the letter i) in the message box. |

| | | |
|---|---|---|
| | warn | Optional. Displays a warning icon (an exclamation mark) in the message box. |
| | error | Optional. Displays an error icon (a Stop sign) in the message box. |
| -buttons | okcancel | Optional. Two buttons will be displayed: OK and Cancel. |
| | yesnocancel | Optional. Three buttons will be displayed: Yes, No, and Cancel. |
| | abortretryignore | Optional. Three buttons will be displayed: Abort, Retry, and Ignore. |
| | ok | The default. One button is displayed: OK. |

**Return Value**

sso msgbox returns the name of the button selected by the user.

The msgbox extension does not set the value of the _SSOERR variable.

**Example**

The following example displays a message box with the message "An error occurred" and Abort, Retry, and Ignore command buttons:

```
set opt [sso msgbox -icon error \
    -buttons abortretryignore -msg "An error occurred"]
switch $opt {
    abort  {
            # abort selected
          ...
            }
    retry  {
            # retry selected
          ...
            }
    ignore {
            # ignore selected
          ...
            }
            }
```

**More information:**

askyesno (see page 107)
inputbox (see page 147)

## net_use

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The net_use extension connects to, or disconnects from, any network provider that works with Windows. It also captures a network resource and maps a network volume to a local drive name or deletes a mapping.

Although pathnames are specified using OS conventions, Tcl requires that backslashes be doubled (to identify them as literals) and that pathnames be enclosed in quotation marks if they contain spaces. For example:

```
\\BOXCAR\\VOL1:c\\APPS
```

Tcl syntax does allow other valid ways to express pathnames. For more information, see the section Strings Containing Special Characters. The previous example can also be expressed:

```
{ BOXCAR\VOL1:c\APPS} or
  BOXCAR/VOL1:c/APPS
```

**Note:** The end user workstation does not need the NetWare client in order to use this extension.

**Syntax**

```
sso net_use {-device deviceName|-net netResource} \
            [-del y|n] [-user userName] \
            [-password password] [-persistent y|n]
```

| Key | Key Value | Description |
|---|---|---|
| -net | *netResource* | Defines which network resource to connect to, including printers. |
| -device | *deviceName* | Optional. Defines which drive to map to. This accepts any free disk drive or printer (LPT1: through LPT3:).<br><br>Type an asterisk instead of a specific device name to assign the next available device name. |
| -del | y\|n | Optional. Disconnects<br><br>This must be used with either -net or -device to specify which mapped drive to delete.<br><br>The default is n. |

| Key | Key Value | Description |
|---|---|---|
| -user | *userName* | Optional. The current user's domain name and user name. If the domain name is omitted, the current domain is assumed. If the user name is omitted, the currently logged in user's user name is used. |
| -password | *password* | Optional. If a user-specified password value is not provided, the current user's password is used. To avoid providing a password, use an empty string (""). |
| -persistent | y\|n | Optional. Restore the connection at the next login.<br>The default is n. |

**Return Value**

When -device is used, the name of the redirected local device is returned. This is an efficient way of obtaining the redirected local name when an asterisk value (next available drive) is used with -device.

When -device is not used, the network resource name is returned.

**Examples**

To log into a specific server and to reconnect at the next login:

```
sso net_use -net {\\acc3_nw410} -user $_USERNAME \
            -password $_PASSWORD
sso net_use -net {\\acc3_nw410} -del yes
```

To map a network volume to G:

```
sso net_use -device G: -net {\\acc3_nw410\sys} \
            -user $_USERNAME -password pokerface  \
            -persistent yes

sso net_use -device G: -del yes
```

To connect to a printer:

```
sso net_use -device lpt2 -net \\\\net3\\hpljet5m-bc \
            -user $_USERNAME -password $_PASSWORD

sso net_use -device lpt2: -del yes
```

## notify

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The notify extension notifies the SSO Server of an event, either a login or a password change and about the result of that event. This directs the SSO Server to update its internal counters and audit trails and, in the event of a successful password change, to change the current password and next password values in the database.

**Note:** This extension will fail if the client is in an offline state. The online status may be determined using the getmode extension. See the getmode extension documentation for further details.

**Syntax**

```
sso notify -event login|pwdchange [-status value] \
           -appname ApplName
```

| Key | Key Value | Description |
|---|---|---|
| -appname | *ApplName* | Specifies the application for which a login or password change was attempted |
| -event | login\|pwdchange | Specifies the name of the event, which is either login or pwdchange |
| -status | *value* | Optional. Specifies the completion status of the specified event:<br>0—Success<br>A non-zero value—Failure<br>If status is omitted, it is assumed that the specified event completed successfully, |

**Example**

To notify the SSO Server about a failed password change:

```
sso notify -event pwdchange -status 3 -appname notes
```

## nw_attach

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The nw_attach extension attaches the current user to a NetWare server.

**Syntax**

```
sso nw_attach -server serverName -user userName \
              [-password password]
```

| Key | Key Value | Description |
|---|---|---|
| -server | *serverName* | Specifies the name of the NetWare server to attach to. |
| -user | *userName* | Specifies the user name for the new connection. |
| -password | *password* | Optional. Specifies the password of the specified user to the specified server. If the specified user has no password on the specified server, this parameter is not used. |

**Example**

To attach the user to NetWare server NOV011:

```
sso nw_attach -server NOV011 -user $_LOGINNAME \
              -password $_PASSWORD
```

**More information:**

nw_logout (see page 159)
nw_map (see page 160)

## nw_capture

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The nw_capture extension captures printer output and directs it to a NetWare queue. This is the same function as the NetWare capture command.

**Syntax**

```
sso nw_capture -lpt lptName -queue queue [-server \ serverName]
```

| Key | Key Value | Description |
|---|---|---|
| -lpt | *lptName* | Specifies the number of the port to be captured. Specify 1 for LPT1, 2 for LPT2, etc. |
| -queue | *queue* | Specifies the name of the NetWare queue to which the captured output should be directed. |

| Key | Key Value | Description |
|---|---|---|
| -server | *serverName* | Optional. Specifies the name of the NetWare server on which the specified queue is located. If the NetWare server name is omitted, the default server is assumed. |

**Example**

The following example captures printer output to LPT1 and directs it to the Q_HPJET4 queue on server NOV011:

```
sso nw_capture -lpt 1 -queue Q_HPJET4 -server NOV011
```

**More information:**

nw_attach (see page 156)
nw_endcap (see page 158)

## nw_endcap

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The nw_endcap extension ends a printer capture previously established using the nw_capture extension. It performs the same function as the NetWare endcap command and the NetWare capture command with the /endcap switch.

**Syntax**

```
sso nw_endcap -lpt lptName
```

| Key | Key Value | Description |
|---|---|---|
| -lpt | *lptName* | Specifies which port to end the capture for. Specify 1 for LPT1, 2 for LPT2, and so forth. |

**Example**

The following example ends the printer capture of LPT1:

```
sso nw_endcap -lpt 1
```

**More information:**

nw_capture (see page 157)

## nw_logintree

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The nw_logintree extension logs the user into a directory tree of NDS (Network Directory Services).

NDS is supported in Novell 4.*x* and above.

The nw_logintree extension keeps the binderies connections intact.

**Syntax**

```
sso nw_logintree -tree treeName -user userName \
                 -password password
```

| Key | Key Value | Description |
|---|---|---|
| -tree | *treeName* | The name of the directory tree to which the user will be logged in. |
| -user | *userName* | The user's login name for the directory tree. |
| -password | *password* | The user's password for the directory tree. |

**Example**

To log the user into the directory tree ACME_CO:

```
sso nw_logintree -tree ACME_CO -user $_LOGINNAME \
                 -password $_PASSWORD
```

**More information:**

nw_attach (see page 156)

## nw_logout

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The nw_logout extension logs the user out of a NetWare server or out of an NDS directory tree. It performs the same function as the NetWare logout command.

The nw_logout extension keeps the binderies connections intact.

**Syntax**

```
sso nw_logout { -server serverName | -tree treeName }
```

| Key | Key Value | Description |
|---|---|---|
| -server | *serverName* | Specifies the name of the NetWare server from which to log out. |
| -tree | *treeName* | Specifies that the user will be logged out of a NDS directory tree and gives the name of the directory tree to which the user was attached. |

**Example**

To log out of NetWare server NOV011:

```
sso nw_logout -server NOV011
```

**More information:**

nw_attach (see page 156)

## nw_map

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The nw_map extension maps a NetWare volume to a local drive or deletes a mapping.

Note that although pathnames are specified using OS conventions, Tcl requires that backslashes be doubled (to identify them as literals) and that pathnames be enclosed in quotation marks if they contain spaces. For example:

```
NOV011\\VOL1:\\APPS
```

Tcl syntax does allow other valid ways to express pathnames. For more information, see the section Strings Containing Special Characters. The previous example can also be expressed:

```
{NOV011\VOL1:\APPS}
```

```
NOV011/VOL1:/APPS
```

You must already be attached to a server in order to use the nw_map extension.

**Syntax**

To map a drive:

```
sso nw_map [-drive drive|next] -resource path
```

To delete a mapping:

```
sso nw_map -del drive
```

| Key | Key Value | Description |
|---|---|---|
| -drive | *drive*\|next | Type either:<br>drive—Specifies the letter of a local drive. Specify one letter, such as G: (you must include the colon)<br>next—Specifies that the next available drive will be mapped to the specified path. |
| -resource | *path* | Specifies the NetWare volume and directory that is to be mapped. The path must begin with a server and volume specification, followed by a directory. For example: NOV011\\VOL1:\\APPS. |
| -del | *drive* | Specifies that the drive mapping should be deleted for the designated drive. |

**Return Value**

The nw_map extension returns the drive letter of the mapped drive.

**Examples**

The following example attaches a user to server NOV011 and then maps the mail directory to drive G: on the user's workstation:

```
sso nw_attach -server NOV011 -user mailuser
sso nw_map -drive G: -resource "NOV011\\VOL1:\\mail"
```

The following example uses the -next option to map NOV011\VOL1:\mail to the next available drive on the user's workstation and set the variable drive_name to that drive:

```
set drive_name [sso nw_map -drive next \
                   -resource "NOV011\\VOL1:\\mail"]
```

**More information:**

nw_attach (see page 156)

## nw_setpass

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The nw_setpass extension changes a password on a file server. It also can synchronize passwords on a set of file servers.

If the workstation is attached to more than one server, and the user has the same password on all of the servers, the nw_setpass extension can synchronize the passwords by using the -sync switch when it changes the password on one file server.

**Syntax**

```
sso nw_setpass [-sync y|n] [-user userName]  \
          [-server serverName] -oldpass oldPassword \
          -newpass newPassword
```

| Key | Key Value | Description |
|---|---|---|
| -oldpass | *oldPassword* | The old password. |
| -newpass | *newPassword* | The new password. |
| -sync | y\|n | Optional. Signifies that passwords will be synchronized on all of the servers to which the user is attached.<br>The default is n. |
| -user | *userName* | Optional. Signifies that the extension is to change a password for another user and gives that user's name. |
| -server | *serverName* | Optional. Signifies that the password will be set on a specific server and gives the name of the server where the password will be changed. |

**Example**

The following example changes the user's password on server NOV011 and synchronizes the user's passwords on all the servers that the user has permission to attach to:

```
sso nw_setpass -sync y -server NOV011 \
               -oldpass $_PASSWORD -newpass $_NEXTPWD
```

**More information:**

nw_attach (see page 156)

## push

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The push extension simulates activating a command button. The extension will fail if the specified button is disabled. The specified button is searched for within the target window.

By default, the push extension works only with the button class. For other classes of buttons, you must add the option:

```
-class NameOfClassOfButton
```

**Examples**

To push an OK button:

```
sso push -label OK
```

To push the second command button in a dialog box:

```
sso push -class Button -ord 2
```

**More information:**

click (see page 111)
type (see page 186)

## pwdbox

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The pwdbox extension displays, at the user's workstation, a dialog box containing a text box for password entry. It includes a duplicate-entry option. The pwdbox dialog box always remains on top as the active window.

If -retype y is used, an additional text box is displayed for password verification. As the user types in the text box, every character is represented by an asterisk (*).

The extension can specify a prompt, which will appear over the Password text box, and a dialog box title.

**Syntax**

```
sso pwdbox [-retype y|n] [-title titleText] \
           [-prompt promptText]
```

| Key | Key Value | Description |
|---|---|---|
| -retype | y\|n | Optional. Specifies that an additional text box (input field) for input verification will be displayed.<br>The default is y. |
| -wrap | y\|n | Optional. Specifies whether to wrap a long prompt. |
| -pwdlen | *number* | Optional. Specifies the minimum length for a new password. |
| -title | *titleText* | Optional. Specifies the title of the dialog box.<br>The default is eTrust Single Sign-On. |
| -prompt | *promptText* | Optional. The text that will be displayed within the dialog box, above the text box (input field).<br>The default is Enter Password. |

**Return Value**

If the user pushes OK, the pwdbox extension returns 0.

The string that the user enters in the Password text box is set as the value of _PASSWORD.

If the user pushes Cancel, the pwdbox extension returns 200 and if the _ERRORMODE is stop or trace_stop, script execution stops.

### Example 1

The following example displays a dialog box with a long prompt and two text boxes. After the user enters a password (text), it assigns the text that was entered to the variable _PASSWORD and checks password length:

```
sso pwdbox -retype y
while {[string length $_PASSWORD] < 10} {
   sso msgbox -msg "Your password is too short."
   sso pwdbox -retype y
}
sso msgbox -msg "Your new password has been accepted."
```

### Example 2

The following example displays a dialog box with a long prompt specified, that uses the wrap text command and has two text boxes. After the user enters a password (text), it assigns the text that was entered to the variable _PASSWORD and checks password length:

```
sso pwdbox -retype y -prompt "Enter your password.  This should be at least ten digits long. A secure password should contain both numbers and letters, and should not be a dictionary word, a name, or a birthday. Do not write your password down." -wrap y
while {[string length $_PASSWORD] < 10} {
   sso msgbox -msg "Your password is too short."
   sso pwdbox -retype y
}

sso msgbox -msg "Your new password has been accepted."
```

### More information:

askyesno (see page 107)
inputbox (see page 147)
msgbox (see page 152)
chlogin (see page 110)
getlogin (see page 117)

## refresh

### Compatibility

Windows 2000/XP/2003.

### Purpose

The refresh extension refreshes the screen scrape performed by the previous extension. It is useful for debugging scripts and also for production scripts.

**Syntax**

```
sso refresh
```

**Example**

To refresh the screen scrape of the previous extension:

```
sso refresh
```

**More information:**

getscrape (see page 121)

## run

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The run extension starts the specified program and continues executing.

In Windows, sso run with the option -getpid y sets the value of the _PID variable thereby identifying the current program. The window extension will search only those windows that belong to the current program.

If the path to the executable is defined in the user's environment, then the executable filename is sufficient, otherwise a full pathname starting with the drivename is required. If the pathname is not correct, the extension fails with error 80 (invalid path or specified path not found).

Sometimes, different copies of the same executable are located on different disks, directories, or subdirectories on different computers within the organization. If this is the case, the same run extension will execute on some workstations and fail on others.

Although pathnames are specified using OS conventions, Tcl requires that backslashes be doubled (to identify them as literals) and that pathnames be enclosed in quotation marks if they contain spaces. For example:

```
"C:\\Program Files\\eTrust\\Ssointrp.exe"
```

Tcl syntax does allow other valid ways to express pathnames. Two pathnames that are equivalent to the previous example are:

```
{C:\Program Files\eTrust\Ssointerp.exe}
```

```
"C:/Program Files/eTrust/Sssointerp.exe"
```

Pathnames must always be written on one line. Tcl will *not* accept a pathname with a return in it, even if the return is prefaced with a backslash or the pathname is enclosed in quotation marks or braces.

Generally, the window extension is used immediately after the run extension to check that the program has opened the desired window (usually a login window).

**Note:** The _PID variable is set by sso run –getpid y to identify the current program.

### Syntax

```
sso run -path pathname [-oneinstance y|n] [-getpid y|n] \
 [-dir directory] [-args "commandLineParameters"] [-mainwin y|n] \
 [-checkprocess NameOfProcess]
```

| Key | Key Value | Description |
|---|---|---|
| -checkprocess | *NameOfProcess* | Specifies the process name for applications which executable and process have two different names.<br>This key is relevant if it is used with –oneinstance y. |
| -path | *pathname* | Specifies the full filename of the program to run, including the file name extension; for example: MYPGM.EXE. If the program is available through the PATH setting, you can specify the program name only; otherwise, the full OS path is required. Pathnames in scripts should be written on one line and not split over two lines with a backslash. |

| Key | Key Value | Description |
|---|---|---|
| -oneinstance | y\|n | Optional. y indicates that the specified program will not be run if there is an instance of the program already running on the workstation.<br><br>The default is n. |
| -getpid | y\|n | Optional. In Windows only. y specifies that the extension will retrieve process identification and store the PID in the _PID variable. n specifies the program will run without retrieving the process identification.<br><br>The default is n. |
| -dir | directory | Optional. Specifies a working directory for the program to be run. The format for the directory is the full pathname according to the OS convention. |
| -args | *commandLine Parameters* | Optional. Provides command line parameters needed by the program to be run. |
| -mainwin | y\|n | Optional. -mainwin y specifies that the run extension will try to find the main window of the application being run, without regard for the title of the window.<br><br>The default is n. |

**Example**

The following example starts Notepad.exe, which is located in the WINNT directory, with the file Tips.txt.

```
sso run -dir "C:\\WINNT" -args "C:\\WINNT\\Tips.txt" \
        -path "C:\\WINNT\\Notepad.exe"
```

**More information:**

nw_map (see page 160)
_PID (see page 211)
Strings Containing Special Characters (see page 26)
Checking Pathnames (see page 90)

## screensize

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The screensize extension is used to retrieve the dimensions of the local primary display monitor in pixels and set height and width variables. This information can be used by the script to position the cursor.

**Syntax**

```
sso screensize
```

**Return Value**

The screensize extension returns the height and width of the display area of the local ordinary monitor by setting two variables:

- _SCREEN_HEIGHT, the display area height in pixels.
- _SCREEN_WIDTH, the display area width in pixels.

**Example**

The following is an example of using screensize extension.

```
sso screensize
sso msgbox -msg "display area height: $_SCREEN_HEIGHT \
display area width: $_SCREEN_WIDTH"
```

## selectitem

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The selectitem extension selects an item in the specified list box or combo box in the target window by simulating a mouse click on that item. Selectitem can select item by name even if it has an icon at the beginning.

If you specify the target field by its position, any field there that is either a list box or a combo box will be considered.

The -control switch limits the search to either a list box or a combo box.

When the -class switch appears in the FieldSpec, the -control option has higher priority than the -class option. It means that, to work with a combo box, you must use -control combo (for list boxes -control list is the default). In case you use the -class option, you must use the option -pos self as shown in the following examples:

```
sso selectitem -class SysListView32 -pos self -item {DEV_DOMAIN1\admin}
sso selectitem -class SysListView32 -pos self -index 6
sso selectitem -control combo -class ComboBox -pos self -item Maximized -numclicks
2
sso selectitem -control combo -class ComboBox -pos self -index 3 -numclicks 2
```

When you specify the target field by its caption, you specify the location of the target field using the -pos option. If you do not specify an option, the default is below.

The item in the list is specified with either the -item or the -index option. One must be used but not both. In some list boxes, selection by name (-item) will not work, but selection by index number will.

You must use the variable class _BOUNDS when there is a list box or combo box inside the frame.

If you want to first select an item from a combo box, you must use the –numclicks option. For example:

```
sso selectitem -control combo -label "Screen Saver" -numclicks 2 0
```

Using -numclicks 2 will first open the combo box and then close it.

If the list box or combo box has one of the following owner-drawn styles:

```
LBS_OWNERDRAWFIXED/CBS_OWNERDRAWFIXED,
LBS_OWNERDRAWVARIABLE/CBS_OWNERDRAWVARIABLE
```

without the style:

```
LBS_HASSTRINGS/CBS_HASSTRINGS
```

then, the selection is enabled only by index instead of by item.

These list boxes or combo boxes with drawn text can be recognized by using Microsoft Spy++.

**Syntax**

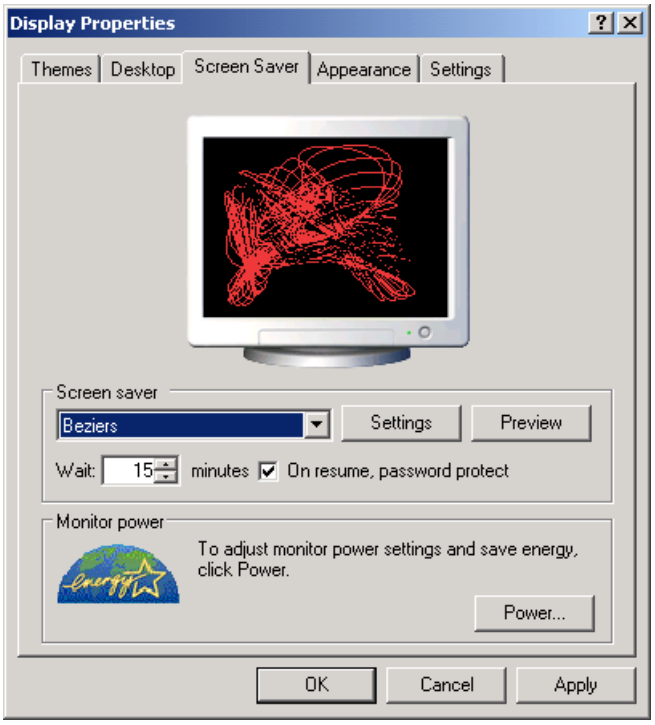
```
sso selectitem [-control list|combo] \
               [-button left|middle|right] \
               [-numclicks number] \
               [FieldSpec] /
               {-item itemName/-index indexNumber}
```

| Key | Key Value | Description |
|---|---|---|
| *FieldSpec* | | Optional. Specifies the target field in which the item to be selected is found. See Specifying Fields in this chapter. When it is not specified, the extension will find the first control on the target window that is either a "SysListView32", a "ListBox" or a "ComboBox". |
| -item | *itemName* | Specifies the full name or the first part of the name of the item to be selected. If the item option is not used, the first item will be selected. |
| -index | *indexNumber* | Selects the nth item in the list in the list or combo box. The first (uppermost) item in the list is 1.<br>The default is 1. |
| -control | list\|combo | Optional. Specifies that the item to be selected is from a combo box or a list box. If -class is used in *Field*Spec, -control has to match what is specified in the -class. The extension will use what is in *FieldSpec* to find the target field and then compare that field with –control option to make sure the right field is located by *FieldSpec*. |
| -button | left\|middle\|right | Optional. Specifies which mouse button will be used to select the item.<br>The default is left. |
| -numclicks | *number* | Optional. Specifies the number of clicks to be simulated when selecting the item.<br>The default is 1. |

### Examples

The following example selects Beziers from the Screen saver combo box in the Display properties window.

```
set _BOUNDS 20
sso window -title "Display Properties"
sso selectitem  -control combo -item "Beziers" \
-label "ScreenSaver"
...
 sso selectitem -control combo -item "Beziers"\-label "Screen Saver" -numclicks
2
```




## selecttab

### Compatibility

Windows 2000/XP/2003.

### Purpose

The selecttab extension selects a tab (tabbed page) in a properties sheet.

### Syntax

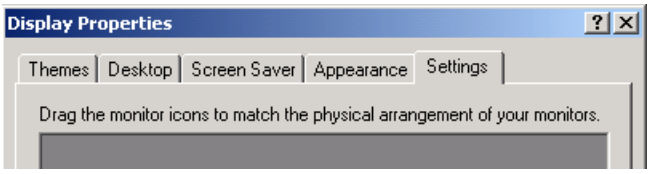
```
sso selecttab -tab | -tabglob | -tabexact tabLabel \
-index indexNum
```

| Key | Key Value | Description |
|---|---|---|
| -tab<br>-tabglob -tabexact | *tabLabel* | Specifies the label of the tab to be selected. |
| -index | *IndexNum* | Specifies the index number of the tab.<br>The default is 1 |

### Example

The following example selects the Settings tab in the Display properties sheet:

```
sso selecttab -tab Settings
```



## setfield

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The setfield extension searches for the specified field within the visible area of the target window. If the specified field is found, setfield sets the field's contents to the given value.

When you designate the target field by its label, you can specify the location of the target field in relation to the label by using a -pos option. If you do not specify a -pos option, the setfield extension defaults to -pos right.

In some cases, the default method of entering a string value into a field causes problems. If this occurs, you can use the -type y option to enter the value as if it were a series of keystrokes.

As a default, when the -autonext option is not used, the extension simulates a keystroke to move to the next field or control. The type of keystroke used is determined by the mode in which the extension is operating:

- In win mode, the extension simulates a tab
- In console mode, the extension simulates an Enter

The type of keystroke used by the default can be changed by setting the _AUTO_NEXT variable to the desired key before the extension is used.

When -autonext *key* is used, the specified functional keystroke will be simulated by the extension. For example, to advance to the next control with a tab key, use -autonext {tab}.

The key to be used must be specified using special character mnemonics as they appear in the appendix "Special Character Mnemonics."

**Syntax**

```
sso setfield [-type y|n] [-autonext key|none] \
             [-timeout TimeValue]
             FieldSpec  -value newValue
```

| Key | Key Value | Description |
|---|---|---|
| *FieldSpec* | | Specifies the target field. The option -pos is relevant only in _MODE win. |
| -value | *newValue* | Specifies the new value for the field. The value can be any string and it must contain at least one character. |

| Key | Key Value | Description |
|---|---|---|
| -type | y\|n | Optional. y specifies that the value will be entered into the field as if it were a series of keyboard actions.<br><br>The default is n.<br><br>This key is relevant only in win mode. |
| -autonext | *key*\|none | Optional. *key* specifies that the extension advances with the indicated key after setting the target field. none specifies that the extension does not advance after setting the target field.<br><br>If the -autonext option is not used the extension advances after setting the target field:<br><br>In win mode, it advances with a tab.<br><br>In console mode, it advances with an Enter. |
| -timeout | *TimeValue* | Optional. Specifies the time to wait for the specified field to be found, in seconds or milliseconds depending upon the suffix used (s and sec designate seconds; ms and msec, milliseconds). When no suffix is used, the time is in seconds. If you do not specify the amount of time, the extension will wait 5 seconds. |

**Examples**

To set user name:

```
sso setfield -label "User Name:" -value $_LOGINNAME
```

To set value to a field labeled Description:

```
sso setfield -label Description \
             -value "Security Administrator"
```

To use a Ctrl (Control) key to advance fields:

```
set _AUTO_NEXT "{^}"
sso setfield -label "User Name:" -value $_LOGINNAME
sso setfield -label "Password:" -value $_PASSWORD
```

**More information:**

_TIMEOUT (see page 216)
getfield (see page 114)
type (see page 186)

## shutdown

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The shutdown extension will either shutdown or restart the computer. The -force flag can be used to forcefully terminate all the currently running processes.

If you want to log off from SSO before the system is shut down, please use the logoff extension.

**Syntax**

```
sso shutdown [-force y|n] [-reboot y|n]
```

| Key | Key Value | Description |
|---|---|---|
| -force | y\|n | If this flag is set to y, the extension forces all running processes to terminate. This can cause the applications to lose data. Therefore, you should use this flag sparingly.<br><br>**Note:** On Windows XP, if the computer is locked and this flag is not specified, the shutdown extension will fail.<br><br>The default value for this flag is n. |
| -reboot | y\|n | If this flag is set to y, the system is shut down and restarted again. The default value for this flag is n. |

**Example**

```
# This command will reboot the system
sso shutdown -reboot y
```

## selecttree

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The selecttree extension selects a folder or object in a box where trees are used to show file or network structure such as Windows Explorer. The selecttree options allow you to toggle a selection on or off, to expand a selection (that is to show the nested folders within the selected folder), or to collapse a selection (that is, to hide the display of nested folders within a selection).

The pathname argument must contain the full pathname to the required element.

**Syntax**

```
sso selecttree [-action toggle|expand|collapse] \
               -path fullPathname
```

| Key | Key Value | Description |
|---|---|---|
| -action | toggle | -action toggle indicates that the specified object will be expanded if it is collapsed or collapsed if it is expanded. |
| -action | expand | -action expand indicates that the specified object will be expanded (that is the nested objects will be displayed). |
| -action | collapse | -action collapse indicates that the specified object will be collapsed (that is the nested objects will be hidden). |
| -path | fullPathname | The full pathname to the specified object, beginning with My Computer up to and including the specified object. Path elements are separated with a slash (not a backslash). |

**Example**

The following example will display the folders within the System folder of Win95:

```
sso selecttree -action expand \
          -path "Desktop/My Computer/C:/Win95/System"
```

## sleep

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The sleep extension suspends execution of the script and passes control to other programs for a specified amount of time. The default unit is seconds.

**Syntax**

```
sso sleep -time timeValue
```

| Key | Key Value | Description |
|---|---|---|
| -time | *timeValue* | Specifies the amount of time that the script will be suspended in seconds or in milliseconds, depending upon the suffix used. The suffixes s and sec designate seconds; ms and msec designate milliseconds. When no suffix is used, the time is in seconds. |

**Example**

The following examples all specify suspending script execution for two seconds:

```
sso sleep -time 2
sso sleep -time 2s
sso sleep -time 2sec
sso sleep -time 2000msec
sso sleep -time 2000ms
```

**More information:**

PAUSE (see page 210)

## statusbox

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The statusbox extension displays a box with a message. Optionally, it displays a message and Cancel button, which when pressed terminates the script. It can also close the box or change the text in the box.

It is used to display information to the end user on the progress of the login process.

If a statusbox box is open, running sso statusbox with a new -msg statusboxText argument will change the text in the box.

If the -button Cancel option is used, the box displays a Cancel button which, when pushed, terminates the script if the _ERRORMODE is stop or trace_stop.

Running sso statusbox -close y will close any open statusbox box.

The script continues to run when the message is being displayed (unlike the msgbox extension, which suspends script execution until the user presses the OK button).

If you choose to use the -button variable with the status box, the only extensions that can actually be cancelled by the user when they click the Cancel button are:

- window
- subwindow
- waittext
- setfield
- getfield
- sleep
- run (when used with –getpid y)
- terminate

If the user clicks the cancel button with any other extensions, the script will just wait for the extension to run to completion or timeout.

### Syntax

```
sso statusbox –msg statusboxText [-button Cancel] [-close –y|n] [-wrap y|n]
[-persistent y|n] [-x x] [-y y]
```

| Key | Key Value | Description |
|---|---|---|
| -msg | *StatusBox Text* | Specifies the text to be displayed inside the statusbox box. The -msg option is not used when -close y is used to close an open statusbox. |
| -x | x | Optional. The x coordinate on the screen of the dialog. Default value -1 which means center the dialog on the screen. Any negative or blank value defaults to -1. Any positive value that is too large will automatically reposition so that the entire dialog is visible on the screen. |
| -y | y | Optional. The y coordinate on the screen of the dialog. See the description for -x. |
| -wrap | y\|n | Optional. Specifies whether the text should wrap so that the message box is not too wide. |
| -persistent | y\|n | Optional. Specifies whether the statusbox should remain open, even if the user clicks the cancel button, until the script has finished or the command "statusbox -close y" is parsed. The -persistent option is therefore only relevant in conjuction with the "-button Cancel" option. |
| -button | Cancel | Optional. Displays a Cancel button.<br>When the user presses this button the currently running extension is terminated. Whether this terminates the entire script or just that extension is determined by the what the _ERRORMODE variable is set to. |
| -close | y\|n | Optional. -close y closes an open status box.<br>The default is n. |

**Example**

The following example opens Notepad and displays messages about the process being carried out:

```
sso run -path "Notepad.exe"
sso statusbox -msg "Notepad application is opening"
sso sleep -time 5
sso window -titleglob "*Notepad*"
if {$_SSOERR == 0 } {
      sso statusbox -msg "Notepad application is open"
      sso sleep -time 5
}
sso statusbox -close y
```

**More information:**

msgbox (see page 152)

## subwindow

### Compatibility

Windows 2000/XP/2003.

### Purpose

The subwindow extension detects document windows or subwindows of a top-level application window. The subwindow extension is useful for handling Multiple Document Interface (MDI) applications, as it allows you to work with MDI windows that are document windows of the window detected by the sso window extension. Usually, the subwindow extension can be used to work with subwindows of other types (non-MDI windows) as well.

The subwindow extension waits for the specified document window to appear. Script execution is suspended until the subwindow is found or until the time-out period expires.

By using the -hwnd option, you specify a subwindow by its window handle value.

You can limit the search to subwindows of a specific class by specifying the class in the *WindowSpec*.

By default, the document window found by the subwindow extension becomes the current window. Then, when the _TARGET_WIN variable is set to current, the subwindow found is the window used for all the field-related extensions that follow. For example, the click extension will search for the specified field only within that subwindow.

When the -focus n option is used, then the document window found by the subwindow extension is not made the active window. This is useful if you want to check whether a subwindow appears but do not want to make it the active window.

The subwindow extension is case-sensitive.

**Note:** Titles (captions) may be changed when subwindows are minimized or maximized.

### Syntax

To look for a single target:

```
sso subwindow [-focus y|n] [-waitfocus time] \
         [-size min|max|same|open] \
         [-move y|n] [-lockinput y|n] \
         [-hwnd windowHandle] \
         [-window active|current] windowSpec
```

To look for several targets:

```
sso subwindow [-focus y|n] [-waitfocus time] \
        [-size min|max|same|open] \
        [-move y|n] [-lockinput y|n] \
        [-hwnd windowHandle] \
        [-window active|current] \
        WindowSpecA WindowSpecB
```

| Key | Key Value | Description |
|---|---|---|
| -focus | y\|n | Optional. -focus n specifies that the extension just checks for the presence of the subwindow without making it the active subwindow.<br><br>The default is y. |
| -waitfocus | *time* | Optional. Sets the time that the extension waits for the designated subwindow to become the active subwindow. The option takes positive integers as arguments; the units are seconds: -waitfocus 10 means wait 10 seconds for focus.<br><br>The default value is 5sec. |
| -size | min | Optional. Indicates that the specified subwindow should be minimized. |
| | max | Optional. Indicates that the specified subwindow should be maximized. |
| | same | Optional. Indicates that the specified subwindow should be kept at the same size it was. |
| | open | Optional. Indicates that the specified window should be restored. |
| -move | y\|n | Optional. y indicates that the subwindow is to be moved so that it is entirely within the bounds of the screen (in order to facilitate screen scraping).<br><br>The default is y. |
| -lockinput | y\|n | Optional. -lockinput y locks user input while the extension is locating the target window and making it active.<br><br>The default is y. |
| -hwnd | *windowHandle* | Optional. Specifies a subwindow by a window handle value stored in a variable. The value is hexadecimal. |

| Key | Key Value | Description |
|---|---|---|
| -window | active | Optional. -window active specifies that the active subwindow be used as the target window, overriding the _TARGET_WIN variable. |
| | current | Optional. -window current specifies that the current subwindow be used as the target window, overriding the _TARGET_WIN variable. |
| *WindowSpec* | | Specifies the subwindow to look for. See Specifying Windows in this chapter.<br>You can specify no more than 32 target subwindows using this command. |
| -target | *WindowSpec* | When the target option uses the options -title and -class, the search is performed only if both options are true. If any of these options is incorrect, the search fails.<br>For example:<br>sso subwindow -target {-title<> -class<>}<br>If in the sso window there are several -target parameters defined, the search returns the first correct occurrence found. |

**Return Value**

The return value is the subwindow title.

**Example**

The following example finds subwindow Book1 on the Microsoft Excel window:

```
sso window -title "Microsoft Excel"
sso subwindow -title "Book1"
```

**More information:**

_PID (see page 211)
_TARGET_WIN (see page 215)
_WIN_TITLE (see page 220)
_WINARRAY (see page 218)
_WINDOW (see page 220)
run (see page 166)
setfield (see page 174)

## terminate

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The sso terminate extension kills the running application.

**Note:** The termination will fail if the user does not have the privileges to terminate the process.

**Syntax**

```
sso terminate -processname ProcessName | -pid ProcessID | -title|\
-titleglob| -titlexact targetText
```

| Key | Key Value | Description |
|---|---|---|
| -processname | *ProcessName* | The name of the process, which can be found in Windows Task Manager. |
| -pid | *ProcessID* | The process ID number, which can be identified by using Microsoft Spy++.<br>The process ID number must be in hexadecimal notation. |
| -title<br>-titleglob<br>-titlexact | *targetText* | The title of the main window of the process to be terminated. For information about the keys -title, -titleglob, and –titlexact, see Specifying Windows in this chapter. |

**Example**

```
sso run -path calc.exe -getpid y
sso terminate -pid $_PID
```

## type

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The type extension simulates typing done by the user into the target window.

Everything that can be done through the keyboard can be accomplished using the type extension, including accessing menu items, activating keyboard shortcuts, pushing buttons, and selecting check boxes.

| Use these symbols... | To denote these keyboard functions |
|---|---|
| % (percent sign) | Alt |
| ^ (caret sign) | Ctrl |
| + (plus sign) | Shift |

To simulate Alt+F, X (select the File/Exit menu), type:

```
sso type -text "%FX"  # select File/Exit
```

Use special character mnemonics surrounded by braces inside quotation marks for special characters. For example:

```
sso type -text "{tab}abc{tab}def{enter}"
```

You can simulate pressing Alt+F* (F1-F12) keys simultaneously. The syntax is:

```
"%{F*}"
```

where F* represents the appropriate F1-F12 key.

You can also repeat a character or a special character using the following syntax:

```
{character n}
```

For example:

```
sso type -text "{tab 4}{a 7}{enter 2}"
```

**Note:** Place the string argument inside quotation marks, even when not required by the rules of syntax.

**Syntax**

```
sso type [-literal y|n ] -text string
```

| Key | Key Value | Description |
|---|---|---|
| -literal | y\|n | Optional. y indicates that the specified string should be typed as-is, without interpreting special characters like {enter}.<br><br>The default is n. |
| -text | *string* | Specifies one or more characters. Use special mnemonics for special characters, such as {enter} for the Enter key. See the appendix "Special Character Mnemonics" for a complete list of special characters mnemonics. |

**Examples**

To type a string:

```
sso type -text "Peterson, Terri"
```

To tab to the next field:

```
sso type -text "{tab}"
```

To type a variable value:

```
sso type -text "$_PASSWORD"
(or sso type -text $_PASSWORD)
```

To type a variable name:

```
sso type -text {$_PASSWORD}
(or sso type -text \$_PASSWORD)
```

To select by pressing the spacebar:

```
sso type -text " "
```

To type Alt+F, X (For selecting File/Exit):

```
sso type -text "%FX" (or sso type -text %FX)
```

To escape from Telnet using the type extension:

```
sso type -text "%{F4}"
```

To type a string including a $ (dollar sign):

```
sso type -text {$22 charge}
```

which gives $22 charge

To type a string that includes special characters as literals use one of the following examples:

```
sso type -literal y \
         -text {Press "p" then press {enter}}
```

or

```
sso type -literal y \
         -text "Press \"p\" then press \{enter\}"
```

both of which give the text:

```
Press p then press {enter}
```

**More information:**

check (see page 108)
menu (see page 151)
setfield (see page 174)

## unlockinput

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The *unlockinput* extension unlocks input from the keyboard and the mouse to the workstation. The *unlockinput* is used in conjunction with the lockinput extension.

**Syntax**

```
sso unlockinput
```

## waittext

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The waittext extension suspends script execution until the target application displays a specified text. It then waits for the application that owns the target window to write the text to the screen. This extension is usually used with text-based applications like telnet emulations. The extension also checks whether the specified text already appears on the screen. The waittext extension is case-sensitive.

This extension intercepts target application APIs such as textout and drawtext. It will also wait for text that appears in a DOS window and a 3270/5250 window.

When the _MODE variable is set to dos_window, the waittext extension searches for the target text by copying the relevant area of the current/active window and then searching for the target text on the clipboard. The area to be copied is determined by setting the various _CUT variables in the script before the waittext extension is invoked.

**Note:** This extensions matches text by line, so if you have text that wraps across a line, you must limit the waittext search to one line only.

**Syntax**

```
sso waittext [-timeout value]  -text targetText
```

to wait for more than one specified text:

```
sso waittext [-timeout value] -text "TextA" -text "TextB"
```

| Key | Key Value | Description |
|---|---|---|
| -timeout | *value* | Optional. Specifies the time to wait for the specified string to appear, in seconds or in milliseconds, depending upon the suffix used (s and sec designate seconds; ms and msec, milliseconds). When no suffix is used, the time is in seconds. If you do not specify the amount of time, the extension will wait 5 seconds. |
| -text | *targetText* | Specifies the text you are waiting for. If the text contains spaces or special characters, enclose it within quotes. The extension, which is case-sensitive, will look for the specified text.<br><br>The maximum number of "-text" options you can specify is ten. Any more than that are silently discarded.<br><br>If you have duplicate values for "-text" options it will cause an error. For example the following will fail:<br><br>sso waittext -text "Fish" -text "Fish" |

**Return Value**

The extension returns the text that was found.

**Example**

This example invokes telnet, waits for a login prompt, types in the login name, waits for a password prompt, and types in the password:

```
sso run -path telnet.exe -args ares
sso window -title "Telnet"
sso waittext -text "login:"
 sso type -text "$_LOGINNAME {enter}"
sso waittext -text "Password:"
 sso type -text "$_PASSWORD {enter}"
```

**More information:**

_MODE (see page 208)
_HIDE_CUT (see page 204)
_PAUSE (see page 210)
_TIMEOUT (see page 216)
_WINDOW (see page 220)

## wintitle

**Compatibility**

Windows 2000/XP/2003.

**Purpose**

The wintitle extension returns the title of a specified window. The target window may be either the active window or the current window, depending on the value of the _TARGET_WIN variable. Wintitle will also return the title of a disabled window.

If the -window active|current option is used it overrides the _TARGET_WIN variable.

**Syntax**

```
sso wintitle [-window active|current]
```

| Key | Key Value | Description |
|---|---|---|
| -window | active\|current | Optional. -window active returns the title of the active window. -window current returns the title of the current window. The current window is set by the window or subwindow extensions. This window is represented by the _WINDOW variable. |

**Return Value**

The title (caption) of the window that was found is returned.

**Example**

This example finds the Exploring window and stores its exact title:

```
set _TARGET_WIN CURRENT
sso window -titleglob "*Exploring*"
set prog_title [sso wintitle]
```

**More information:**

_WINDOW (see page 220)
selectitem (see page 169)
window (see page 192)

## window

### Compatibility

Windows 2000/XP/2003.

### Purpose

The window extension determines if a specified window is present and changes its status, its display, or both. The window extension waits for the specified window to appear and script execution is suspended until a window is found or until the time-out period expires.

All the top-level windows on the desktop are searched. If more than one window with the same *WindowSpec* exists and the option -getpid y was used with the run extension, then the window extension selects the window that was created by the current program (the program set by the most recent run extension).

By default, the window found by the window extension is made the current window. If the _TARGET_WIN variable is set to current, this window will be the window used by all the field-related extensions that follow. For example, the click extension will search for the specified button only within this window.

By using the -hwnd option, you specify a window by its window handle value. When it is used, the –window and WindowSpec are ignored.

By using the –window option, you specify the window as either the current window or the active window. When it is used, the WindowSpec is ignored.

You can also specify the window by using WindowSpec.

You have to use either –hwnd, -window or WindowSpec to specify the window. When all of them are missing, syntax error will be reported.

Note that sometimes when a window is minimized or maximized, the window title is changed (for example, truncated when the window is minimized).

The window extension is case-sensitive.

### Syntax

To search for a single target:

```
sso window [-focus y|n] [-waitfocus time] \
     [-size min|max|same|open] \
     [-move y|n] [-lockinput y|n] \
     [-hwnd windowHandle] \
     [-window active|current] windowSpec
```

To search for several targets:

```
sso window [-focus y|n] [-waitfocus time] \
    [-size min|max|same|open] \
    [-move y|n] [ -lockinput y|n] \
    [-hwnd windowHandle] \
    [-window active|current] \
    windowSpecA windowSpecB
```

| Key | Key Value | Description |
|---|---|---|
| -focus | y\|n | Optional. -focus n specifies that the extension just checks for the presence of the window without making it the active window.<br>The default is y. |
| -waitfocus | *time* | Optional. Sets the time that the extension waits for the designated window to become the active window. The option takes positive integers as arguments; the units are seconds: -waitfocus 10 means wait 10 seconds for focus.<br>The default value is 5sec. |
| -size | min | Optional. Indicates that the specified window should be minimized. |
| | max | Optional. Indicates that the specified window should be maximized. |
| | same | Optional. Indicates that the specified window should be kept at the same size it was.<br>The default is same. |
| | open | Optional.Indicates that the specified window should be restored. |
| -move | y\|n | Optional. y indicates that the window is to be moved so that it is entirely within the bounds of the screen (in order to facilitate screen scraping).<br>The default is y. |
| -lockinput | y\|n | Optional. -lockinput y locks user input while the extension is locating the target window and making it active.<br>The default is y. |

| Key | Key Value | Description |
|---|---|---|
| -hwnd | *windowHandle* | Optional. Specifies a window by a window handle value stored in a variable. The value is hexadecimal. When the key -hwnd is issued in the script, this key finds the window. The *WindowSpec* and -window issued in the same script are ignored. |
| -window | active | Optional. -window active specifies that the active window be used as the target window, overriding the _TARGET_WIN variable. When it is used, the WindowSpec is ignored. |
| | current | Optional. -window current specifies that the current window be used as the target window, overriding the _TARGET_WIN variable. When it is used, the WindowSpec is ignored |
| *WindowSpec* | | Specifies the window to look for. See Specifying Windows in this chapter.<br>You can specify no more than 32 target windows using this command. |
| -target | *WindowSpec* | When the target option uses the options -title and -class, the search is performed only if both options are true. If any of these options is incorrect, the search fails.<br>For example:<br>sso window -target {-title <> -class <>}<br>If in the sso window there are several -target parameters defined, the search returns the first correct occurrence found. |

**Return Value**

The return value is the window title.

**Examples**

To find the Windows Explorer window:

```
sso window -titleglob "*Explor*"
```

To make a window of the #32770 class with the title Find: Computer the active window:

```
sso window -target {-class #32770 -title "Find: Computer"}
```

To store a window handle value and use it later to make the window the current window:

```
sso window -title calculator
set CALC_WND $_WINDOW
.
.
.
sso window -title Untitled
.
.
.
sso window -hwnd $CALC_WND
```

To wait for either of two windows:

```
set MyWin [sso window -title “title1" -title “title2"]

switch $MyWin {
   title1 {
      # handle title1 window
      ...
      }
   title2 {
      # handle title2 window
      ...
      }
   default {
      sso msgbox –msg "Error: $_SSOERR"
      exit
      }
   }
```

**More information:**

_PID (see page 211)
_WINARRAY (see page 218)
_WINDOW (see page 220)
_WIN_TITLE (see page 220)
_TARGET_WIN (see page 215)
_TIMEOUT (see page 216)
run (see page 166)
setfield (see page 174)
subwindow (see page 182)

# Chapter 6: Script Variables

This section contains the following topics:

# Script Variables

The variables in eTrust SSO scripts affect the behavior of some or all of the eTrust SSO extensions. Variable values can be changed as needed and default values have been chosen so that it will be easier to develop scripts. Certain variables are updated by some or all of the eTrust SSO extensions.

eTrust SSO uses three classes of script variables:

- Operational variables
- Login variables
- HLLAPI variables

The following section describes these variables, their value ranges and defaults, and their interaction with the eTrust SSO extensions.

# Variable Classes

eTrust SSO scripts use three classes of script variables: operational variables, hllapi variables, and login variables.

The operational variables define the operating environment of the SSO extensions. They can be used by an eTrust SSO script during its execution.

The login variables hold information needed for, and affected by, application login. These variables are fetched from the eTrust SSO database on the SSO Server host. Some of them pertain to the current application, some of them are specific to the current user in relation to the current application, and some of them hold installation-wide data.

The HLLAPI variables are used only by the HLLAPI extensions.

# _APPNAME

**Variable Type**

Login variable.

**Purpose**

This variable contains the name of the current application.

One script can serve many applications. For example, you might have one application called TELNET_A and another application called TELNET_B that both use the same script (TELNET.TCL) but with different HOST values (A and B). You use the _APPNAME variable when there is a need for the script to know the name of the current application.

**Source**

The APPL record in the eTrust SSO database.

# _AUTO_NEXT

**Variable Type**

Operational variable.

**Purpose**

Sets the default keystroke used by sso setfield to advance after setting a field.

**Default Value**

Default value depends on the mode under which sso setfield is operating: in win mode—"{tab}", in console mode—"{enter}".

**Extensions Affected**

sso setfield

**Example**

To use a Ctrl (Control) key to advance fields with sso setfield:

```
set _AUTO_NEXT "{^}"
sso setfield -label "User Name:" -value $_LOGINNAME
sso setfield -label "Password:" -value $_PASSWORD
```

# _BEGIN_CUT

**Variable Type**

Operational variable.

**Purpose**

Used by sso waittext in dos_window mode. Sets the command set that invokes the mark command in a DOS window, emulation window, or other similar window. The command set is the string of keyboard actions and can be different for each different application. In a DOS window, this is Alt ek.

**Default Value**

Alt ek (DOS Window values)

**Extensions Affected**

sso waittext

# _BOUNDS

**Variable Type**

Operational variable.

**Default Value**

Default or script.

**Purpose**

Defines the range in which a Windows field-level extension will search for a target field relative to a given label (caption).

**Extensions Affected**

All Windows field-level extensions.

# _COL

**Set By**

This variable is set by:

- hllapi_getcursor
- hllapi_getfield
- hllapi_setfield
- hllapi_waittext

**Default Value**

Default value –1 (variable not in use).

**Variable Value**

A positive integer from 1 (often from 1 to 80).

# _CUT_OFFS_BOTTOM

**Variable Type**

Operational variable.

**Purpose**

This variable is used by sso waittext in dos_window mode. It sets the bottom boundary of the area to be marked and copied by the extension. The value is relative to the outer bounds of the window (including the title bar).

**Extensions Affected**

sso waittext

# _CUT_OFFS_LEFT

**Variable Type**

Operational variable.

**Purpose**

This variable is used by sso waittext in dos_window mode. It sets the left boundary of the area to be marked and copied by the extension. The value is relative to the outer bounds of the window (including the title bar).

**Extensions Affected**

sso waittext

# _CUT_OFFS_RIGHT

**Variable Type**

Operational variable.

**Purpose**

This variable is used by sso waittext in dos_window mode. It sets the right boundary of the area to be marked and copied by the extension. The value is relative to the outer bounds of the window (including the title bar).

**Extensions Affected**

sso waittext

# _CUT_OFFS_TOP

**Variable Type**

Operational variable.

**Purpose**

This variable is used by sso waittext in dos_window mode. It sets the top boundary of the area to be marked and copied by the extension. The value is relative to the outer bounds of the window (including the title bar).

**Extensions Affected**

sso waittext

# _END_CUT

**Variable Type**

Operational variable.

**Purpose**

This variable is used by sso waittext in dos_window mode. It sets the command set that invokes the copy command in a DOS window, emulation window, or other similar window and completes the cut operation. The command set is the string of keyboard actions and can be different for each different application. In a DOS window, this is Alt key.

**Default Value**

Alt key (DOS Window values)

**Extensions Affected**

sso waittext

# _ERRORMODE

**Variable Type**

Operational variable.

**Purpose**

This variable determines the behavior of eTrust SSO extensions when they fail to perform their functions.

| Variable Value | Extension Behavior |
|---|---|
| resume | The extension sets the _SSOERR variable with a return code and resumes execution at the next command. |
| msg | The extension displays an error message box at the user workstation, sets the _SSOERR variable, and resumes execution at the next command. |
| stop | The extension displays an error message box at the user workstation and terminates the script. |
| trace_stop | The extension displays a detailed error message at the user workstation and terminates the script. |

**Description**

This variable affects only eTrust SSO extensions. It has no effect on native Tcl commands and other extensions. If there is an error within a native Tcl extension, the script aborts with the stop message (or with a trace_stop message if a trace stop mode is set).

**Default Value**

stop

**Extensions Affected**

All eTrust SSO extensions

**Example**

```
set _ERRORMODE resume
sso run -path mypgm.exe
if { $_SSOERR != 0 } {
   sso msgbox -icon error \
               -msg "The mypgm application did not start"
exit
   }
#set _ERRORMODE back to default
set _ERRORMODE stop
```

# _HIDE_CUT

**Variable Type**

Operational variable.

**Purpose**

This variable is used by sso waittext in dos_window mode. It hides the mark rectangle, which is white in a DOS window, from the user, who just sees the cursor move.

**Default Value**

yes (hides the mark rectangle)

**Extensions Affected**

sso waittext

**More information:**

_MODE (see page 208)

# _HIDE_CUT

**Variable Type**

Operational variable.

**Purpose**

This variable is used by sso waittext in dos_window mode. It hides the mark rectangle, which is white in a DOS window, from the user, who just sees the cursor move.

**Default Value**

yes (hides the mark rectangle)

**Extensions Affected**

sso waittext

# _HLLAPI_FUNC_NO

**Variable Type**

HLLAPI variable.

**Set By**

All HLLAPI extensions.

**Purpose**

This variable contains the number of the last HLLAPI function that was executed by the last HLLAPI extension. It is set, along with _HLLAPI_RC, by all the HLLAPI extensions. Refer to this variable only when the _SSOERR variable was set to 70 by the last HLLAPI extension.

| Variable Value | Extension Behavior |
|---|---|
| Positive integers | None. |

**Default Value**

Default value –1.

# _HLLAPI_RC

**Variable Type**

HLLAPI variable.

**Set By**

All HLLAPI extensions.

**Purpose**

This variable contains the HLLAPI standard return code from the last HLLAPI function that was executed by the last HLL extension. This return code indicates a specific reason if the script did not perform its expected action. It is set, along with _HLLAPI_FUNC_NO, by all the HLL extensions. Use this variable only when the _SSOERR variable was set to 70 by the last HLL extension.

**Variable Value**

0 and positive integers

**Default Value**

Default value –1.

**More information:**

Specialized HLLAPI Extensions (see page 231)

# _HOOK_MODE

**Variable Type**

Operational variable.

**Purpose**

This variable is relevant to extensions that use a scraping technique (sso waittext and sso getscrape).

| Variable Value | Extension Behavior |
|---|---|
| native | The default value for all operating systems. |
| winnt | In Win 9x when a combination of Win 9x and WinNT techniques for scraping is needed (especially for Java applets). |

**Description**

_HOOK_MODE controls the NT scraping mechanisms usage. The variable values are *native* and *winnt*. The variable is set to *winnt* to enable NT scraping in Windows 9x.If you work in Windows 9x with Netscape Communicator (netscape.exe) running Java applets, Java SDK Applet Viewer 1 (appletviewer.exe) and others, it is recommended to set the variable value to *winnt* .

**Default Value**

Native

**Extensions Affected**

sso waittext and sso getscrape

# _HOST

**Variable Type**

Login variable.

**Purpose**

This variable contains the identifier of the host on which the current application is located.

One script can serve many applications. For example, you might have one application called TELNET_A and another application called TELNET_B, both using the same script (TELNET.TCL) but with different HOST values (A and B).

**Source**

The APPL record in the eTrust SSO database.

# _LOGINCOUNT

**Variable Type**

Login variable.

**Purpose**

This variable contains the number of successful logins by the user into the current application.

If a script uses the notify extension to notify login events, then it will increment the _LOGINCOUNT variable each time a login is successful.

The variable is set by the getlogin extension.

**Source**

The login record in the eTrust SSO database.

**More information:**

Writing eTrust SSO Scripts (see page 81)
SSO Extensions (see page 95)

# _LOGINNAME

**Variable Type**

Login variable.

**Purpose**

This variable contains the login name of the current user of the current application.

This login name is specific to the current application, because the current user may have different login names for different applications. The login name can be different from the user name, which is the name used to identify the current user within the eTrust SSO database.

The variable is set by the getlogin extension.

If the application name consists of more than one word, then you must use braces { } to see the application data in the message box.

**Source**

The login record in the eTrust SSO database.

**Example**

The following example shows how to use braces when the application name consists of more than one word.

```
sso getlogin -appname "hag sameah"
sso msgbox -msg ${hag sameah_LOGINNAME}
```

# _MODE

**Variable Type**

Operational variable.

**Purpose**

This variable determines the scripting mode of some of the extensions that follow it in the script. The scripting mode can be either Windows, Windows DOS, or console.

When working in Windows mode, which is the default mode, extensions perform screen scraping of controls and the contents of controls.

When working in console mode, extensions perform screen scraping of text.

In DOS window mode, all the extensions operate on controls, except waittext, which retrieves screen contents by cut and paste operations.

| Variable Value | Extension Behavior |
|---|---|
| win | Extensions perform screen scraping of controls.<br>This is the default value of the variable. |
| console | Extensions perform screen scraping of text. |
| dos_window | All extensions except waittext perform as in win mode.<br>sso waittext functions using cut and paste operations. |

**Default Value**

win

**Extensions Affected**

Most eTrust SSO extensions.

# _NEXTPWD

**Variable Type**

Login variable.

**Purpose**

This variable contains the new password for the current user and application. A non-null value for the variable indicates that the user password must be changed on the application server.

A script should be written to check the value of the _NEXTPWD variable during the login process and to perform a password change if this variable is not empty. When the variable is empty, there is no need for a password change.

The _NEXTPWD variable is not empty when one of the following has occurred:

The end user changes his or her password using SSO Tools or Launchbar (part of SSO Client).

The eTrust SSO administrator changes the Next Password field for an end user.

The end user's password for the application expires and the end user changes the password.

The user's password for the application expires and the SSO Server automatically generates a new password.

The variable is reset by the getlogin extension.

**Source**

The login record in the eTrust SSO database.

# _PASSWORD

**Variable Type**

Login variable.

**Purpose**

This variable contains the password of the current user of the current application.

The value of this variable depends on the login type of the current application, as defined in the eTrust SSO database. For a password-based application, this is the real password of the user. For a ticket-based application, this is a time-based ticket. For an OTP application, this is the One Time Password.

The variable is set by the getlogin extension.

**Source**

For password-based applications: The login record in the eTrust SSO database.

For ticket-based or OTP applications: Dynamic generation by the SSO Server.

# _PAUSE

**Variable Type**

Operational variable.

**Purpose**

This variable holds the amount of time that the script will pause after each eTrust SSO extension is executed. By default, this variable is set to 0, denoting no pause. The amount of time is specified in seconds by default, but you can specify the amount of time in milliseconds.

You can use a pause when your script malfunctions and you suspect it is because of timing problems.

| Variable Value | Extension Behavior |
|---|---|
| Positive numeric values | Specifies that the extension pauses after execution for the time specified. If no units are indicated the time is in seconds. |
| 0 | No pause (default value) |

**Default Value**

0 (no pause)

**Extensions Affected**

All eTrust SSO extensions.

**Examples**

Pauses for 5 seconds:

```
set_PAUSE 5
```

Pauses for 5000 msec (5 seconds):

```
set_PAUSE 5000 msec
```

# _PID

**Variable Type**

Operational variable.

**Set By**

This variable is set by the sso run extension.

**Purpose**

This variable represents the current program. In Windows, it contains the descriptor identifying the current program. If there is more than one window found with the sso window extension, the sso window extension will choose the window belonging to the current program.

Normally, you do not change the value of this variable. In special cases, you can set it to zero, so there will be no current program.

The first part of the script variable _PID is the actual process identifier (pid) in HEX format. And the second part is the thread identification number (tid), which is also in HEX format.

**Default Value**

The default value is an empty string. There is no current program until the run extension is used.

**Extensions Affected**

- window
- subwindow

**Example**

The process started by the last "sso run -getpid y" command can be closed with: "sso terminate -pid $_PID"

The extension will extract the process id out of _PID and will terminate the process properly.

**More information:**

SSO Extensions (see page 95)
subwindow (see page 182)
window (see page 192)

# _POLLDELAY

**Variable Type**

Operational variable.

**Purpose**

This variable controls the number of milliseconds between pollings for input events.

| Variable Value | Extension Behavior |
|---|---|
| 0 | Disabled |
| [Any positive number] | Number of milliseconds between polls. |

**Default Value**

10 (milliseconds)

**Extensions Affected**

Sleep extension.

**Example**

set _POLLDELAY 20

# _ROW

**Variable Type**

HLLAPI variable.

**Set By**

This variable is set by:

- hllapi_getcursor
- hllapi_getfield
- hllapi_setfield
- hllapi_waittext

**Purpose**

This variable represents a row number in the 3270 screen. It is set, along with _COL, by some of the HLL extensions. It is used by SSO 3720 extensions to locate field or cursor position in the 3270 screen.

**Default Value**

–1 (variable not in use).

**Variable Value**

Positive integer from 1 (generally from 1 to 24).

**More information:**

SSO Extensions (see page 95)

# _SCRAPE_TIMEOUT

**Purpose**

This variable determines the maximum time for the sso getscrape extension. The value is expressed in seconds and milliseconds.

When no suffix is used (sec, msec), time is measured in seconds.

**Default Value**

2000 msec.

**Extensions Affected**

sso getscrape

# _SCREEN_HEIGHT

**Variable Type**

Operational variable.

**Set By**

This variable is set by screensize.

**Purpose**

This variable contains the height, in pixels, of the display area of the primary monitor of the local workstation, as retrieved by the last screensize extension.

**Default Value**

There is no default value.

**More information:**

SSO Extensions (see page 95)

# _SCREEN_WIDTH

**Variable Type**

Operational variable.

**Set By**

This variable is set by screensize.

**Purpose**

This variable contains the width, in pixels, of the display area of the primary monitor of the local workstation, as retrieved by the last screensize extension.

**Default Value**

There is no default value.

# _SSOERR

**Variable Type**

Operational variable.

**Set By**

This variable is set by all the eTrust Single Sign-On extensions, except msgbox and sleep.

**Purpose**

The _SSOERR variable is set to a completion code value by all the eTrust SSO extensions except msgbox and sleep, and reflects the completion status of the command. It can be checked by the script to see if the last command completed successfully.

If you want to interrogate the return code (the result of a Tcl command) generated by a command run prior to the last command, you must save the return code in another variable.

The various completion codes are listed in the appendix "Completion Codes."

This variable is relevant only if the current value of _ERRORMODE is either resume or msg. Otherwise, the script aborts if one of the commands fails.

| Variable Value | Extension Behavior |
|---|---|
| 0 | Indicates successful completion. |

| | |
|---|---|
| Positive integer | Indicates warning or failure. |

**Default Value**

The default value is 0.

**Extensions Affected**

All the eTrust SSO extensions, except msgbox and sleep are affected.

**Example**

```
sso window -title "window1"
if { $_SSOERR !=0 } {
sso msgbox -msg "No window found"
exit
}
```

# _TARGET_WIN

**Variable Type**

Operational variable.

**Purpose**

This variable determines whether field-related extensions and the subwindow extension will look for fields within the current window or within the active window. The current window is set by the window and subwindow extensions.

Looking for fields in active windows will result in a shorter script; you will generally be able to move between different windows without writing any specific code.

To look for fields in current windows, you have to set the target window by using the window or subwindow extensions.

If the value of _TARGET_WIN is current and a window is opened as a result of one of your actions, you must make it the current window if you want to search for a field in it.

| Possible Values | Description |
|---|---|
| active | The target window is the active window. |
| current | The target window is the current window. |

**Default Value**

current

**Extensions Affected**

This variable affects:

- All Windows field-related extensions
- window
- subwindow

# _TIMEOUT

**Variable Type**

Operational variable.

**Purpose**

This variable controls the time that some of the eTrust SSO extensions wait before failing. The value is expressed in seconds (the default) or milliseconds (by adding msec).

| Variable Value | Extension Behavior |
|---|---|
| Positive integer | Extension waits before failing |
| 0 | No wait. |
| -1 | Extension waits forever |

**Note:** The value of -1 is not supported by the HLLAPI extensions. If a HLLAPI extension is specified, the default value (5 seconds) is set instead of the -1 value.

**Default Value**

5 seconds

**Extensions Affected**

This variable affects:

- window
- subwindow
- waittext
- All HLLAPI extensions

**Example**

To have the window extension wait 5 seconds for the window to appear before failing:

```
set _TIMEOUT 5
sso window -title "Program Manager"
```

# _USERNAME

**Variable Type**

Login variable.

**Purpose**

This variable contains the name of current user logged into the SSO Client.

**Example**

```
sso msgbox -msg "User's name is $_USERNAME"
```

# _WINARRAY

**Variable Type**

Operational variable.

**Set By**

This variable is set by window and subwindow.

**Purpose**

The _WINARRAY variable stores the window handles of windows or subwindows found by the window or subwindow extension.

When a window or subwindow extension finds more than one window, the interpreter returns an error code of 21 and puts the values of the window handles in the _WINARRAY variable. The script can then examine all of these windows and try to find the target text in them by using the -hwnd option. See the example that follows.

**Extensions Affected**

This variable affects window and subwindow.

**Example**

```
set _ERRORMODE resume
sso window -title "Exploring"
if { $_SSOERR == 21 } {
for { set i 0 } { [lindex $_WINARRAY $i] != "" } {incr i } {
sso msgbox -msg "index: $i\n handle: [lindex $_WINARRAY $i]"}
 }
```

# _WIN_INFO

**Variable Type**

Operational variable.

**Purpose**

This variable contains extra information about the current window, which has been found by the last “window” or “run” command, including:

| Parameter | Description |
|---|---|
| Window class | The name of the class to which the current window belongs. For example: “ConsoleWindowClass” or “#32770” |

| Parameter | Description |
|---|---|
| Show state | Specifies the show state of the window. This element can be one of the following values:<br>SW_HIDE<br>SW_MAXIMIZE<br>SW_MINIMIZE<br>SW_RESTORE<br>SW_SHOW<br>SW_SHOWMINIMIZED<br>SW_SHOWMINNOACTIVE<br>SW_SHOWNA<br>SW_SHOWNOACTIVATE<br>SW_SHOWNORMAL<br>UNKNOWN_SHOW_STATE |
| Visibility | Specifies whether the Windows is visible. This is set to WS_VISIBLE if the window is currently visible, or is set to WINDOW_NOT_VISIBLE if the window is hidden. |
| Left and Top | Specifies the screen coordinates of the window's top-left corner |
| Width and Height | Specifies the size of the window |

**Default Value**

This variable does not have a default value and is set by "sso run –getpid y", "sso window", and "sso subwindow".

**Example**

```
"ConsoleWindowClass" SW_SHOWNORMAL WS_VISIBLE 16 173 669 338
"#32770" SW_SHOWNORMAL WS_VISIBLE 335 234 483 362
"#32770" SW_SHOWNORMAL WINDOW_NOT_VISIBLE 335 234 483 362
"ExploreWClass" SW_MAXIMIZE WS_VISIBLE -4 -4 1160 838
```

# _WIN_TITLE

**Variable Type**

Operational variable.

**Set By**

This variable is set by window and subwindow.

**Purpose**

This variable contains the full title (caption) of the current window, found by the last window or subwindow extension.

The _WIN_TITLE value is exact. If you use Tcl extensions to process this variable, remember that Tcl is case-sensitive.

**Default Value**

There is no default value.

**Extensions Affected**

All the eTrust SSO extensions.

**Example**

To run pifedit.exe and show the full title of the pifedit window:

```
sso run -path pifedit.exe
sso window -title PIF
sso msgbox -msg $_WIN_TITLE
```

# _WINDOW

**Variable Type**

Operational variable.

**Set By**

This variable is set by window and subwindow.

**Purpose**

This variable contains an identifier for the current window. In the Windows environment, it holds the window handle of this window.

If the _TARGET_WIN variable is set to current, the current window is used by all field-related extensions as their target window. For example, the click extension will look for the specified button only within the current window.

As long as the _TARGET_WIN variable is set to active, the _WINDOW variable has no effect.

This variable is set and used by eTrust SSO extensions. Do not change its value.

**Default Value**

The default value is 0.

**Extensions Affected**

This variable affects:

- check
- getfield
- menu
- selectitem
- setfield
- type
- waittext
- wintitle

# Appendix A: Completion Codes

This section contains the following topics:

Completion Code Table (see page 223)

## Completion Code Table

The following table lists the completion codes returned by the various extensions provided by eTrust SSO.

| Type | Code | Meaning |
|---|---|---|
| HTML extension error or warning | -1 | HTML extension encountered problem or error. |
| Successful completion | 0 | The process succeeded. |
| Bad syntax | 1 | The syntax of the extension was invalid. |
| Target not available | 2 | The target field or menu item was not found in the target window. |
| | 3 | The target field or menu item was found but was disabled. |
| Target owner not available | 4 | The target window was not found. |
| | 5 | The target window was found but was disabled. |
| Item in target not available | 6 | The item inside the target window was not found. |
| | 7 | The item inside the target window was disabled. |
| | 8 | The caret was not found. |
| | 10 | The -from item was not found. |
| Target window could not be activated | 20 | The target window could not be activated. |
| | 21 | Detected more than one window that matches the search criteria. |

| Type | Code | Meaning |
|---|---|---|
| HLLAPI error or warning | 70 | A HLLAPI function, used in a HLLAPI extension, returned an error or warning code. Retrieve the values of the _HLLAPI_FUNC_NO and _HLLAPI_RC variables for specific information. |
| Invalid path | 80 | Invalid path or the specified path was not found. |
| Invalid mode | 81 | Invalid mode. The specified mode is not valid for the current DOS window. The valid mode must be dos_window mode. |
| | 90 | Invalid mode. The specified extension is not valid in the current mode (a HLLAPI extension was called in Windows mode or a Window extension was called in 3270 mode). |
| Program fault | 99 | An unexpected result of SSO internal code was detected. Contact your local marketing representative. |
| Fatal error | 100 | A fatal error has occurred. Contact Technical Support at http://supportconnect.ca.com for more assistance. |
| User cancel | 200 | The script execution was canceled by the user. |
| eTrust Access Control miscellaneous failures | 350 | SSO - Selang command failure. |
| | 351 | SSO - password quality control failed. |
| | 352 | SSO - query rejected. |
| Communication errors | 400 | Communications error. |
| | 410 | Communication protocol error. |
| | 411 | Communication connect error. |

# Appendix B: Special Character Mnemonics

This section contains the following topics:



## Windows Keystroke Simulation Table

When writing a script, you can simulate special keys on your keyboard such as <Enter> and <Tab>, as if an operator were typing on a regular keyboard. This can be done using the type extension combined with the mnemonic that represents that special character.

For example, to simulate pressing the **Page Down** key you would type:

```
sso type –text "{pgdn}"
```

This following table shows the windows keystroke or action followed by the corresponding mnemonic:

| Key or Action | Mnemonics |
|---|---|
| Backspace | {backspace} |
| Caps Lock | {capslock} |
| Clear | {clear} |
| Ctrl | {break} |
| Ctrl (right) | {r_ctrl} |
| Delete | {delete} or {del} |
| Down arrow | {down} |
| End | {end} |
| Enter or Return | {enter} |
| Escape | {escape} or {esc} |
| Help | {help} |
| Home | {home} |

| Key or Action | Mnemonics |
|---|---|
| Insert | {insert} |
| Left arrow | {left} |
| NumLock | {numlock} |
| Pause | {pause} |
| PgDn | {pgdn} |
| PgUp | {pgup} |
| Print Screen or Snapshot | {prtsc} |
| Right arrow | {right} |
| Scroll Lock | {scrollock} |
| Tab | {tab} |
| Up arrow | {up} |
| F1 | { F1} |
| F2 | { F2} |
| F3 | { F3} |
| F4 | { F4} |
| F5 | { F5} |
| F6 | { F6} |
| F7 | { F7} |
| F8 | { F8} |
| F9 | { F9} |
| F10 | { F10} |
| F11 | { F11} |
| F12 | { F12} |
| F13 | { F13} |
| F14 | { F14} |
| F15 | { F15} |
| F16 | { F16} |

# Windows Mulitple Keystroke Simulation Table

In addition to the single keystroke mnemonics, you can use the <Shift>, <Ctrl> and <Alt> keys in conjunction with other keystrokes, as if you were holding them down.

For example, to simulate holding down the <Shift> key and typing other keys, to produce **OCSPro**, you would type:

```
sso type –text "+o+c+s+pro}"
```

The following table shows the 'hold-down' key and corresponding mnemonic:

| Key | Mnemonic |
|---|---|
| Shift | + |
| Ctrl | ^ |
| Alt | % |

# 3270 Keyboard Mnemonics

When writing a script, you can simulate special typing keys such as <Enter> and <Tab>, as if an operator were typing on the 3270 terminal keyboard. This can be done using the **hllapi_type** extension, with the special mnemonics that represent the special characters.

The following table shows these special mnemonics, together with the standard HLLAPI representation of the special characters. The HLLAPI representation is given for clarification, and is not intended for use in the SSO scripts.

The maximum string that hllapi_type can send to the connected terminal at one time is 255 characters. Sending a string that is longer than 255 characters will return an error.

Any text passed into a field that exceeds that field's character limit will spill over into the next field.

| HLLAPI Standard Representation | Mnemonic |
|---|---|
| @@ | {@@} |
| @A@Q | {attention} |
| @A@Q | {attn} |
| @< | {backspace} |

| HLLAPI Standard Representation | Mnemonic |
|---|---|
| @B | {backtab} |
| @Y | {capslock} |
| @C | {clear} |
| @V | {cursordown} |
| @L | {cursorleft} |
| @Z | {cursorright} |
| @U | {cursorup} |
| @D | {del} |
| @D | {delete} |
| @V | {down} |
| @S@X | {dup} |
| @q | {end} |
| @E | {enter} |
| @F | {eof} |
| @F | {eraseeof} |
| @A@F | {eraseinput} |
| @F | {ereof} |
| @A@F | {erinput} |
| @O | {home} |
| @I | {insert} |
| @J | {jump} |
| @L | {left} |
| @B | {lefttab} |
| @N | {newline} |
| @t | {numlock} |
| @x | {pa1} |
| @y | {pa2} |
| @z | {pa3} |
| @1 | {pf1} |
| @a | {pf10} |
| @b | {pf11} |

| HLLAPI Standard Representation | Mnemonic |
|---|---|
| @c | {pf12} |
| @d | {pf13} |
| @e | {pf14} |
| @f | {pf15} |
| @g | {pf16} |
| @h | {pf17} |
| @i | {pf18} |
| @j | {pf19} |
| @2 | {pf2} |
| @k | {pf20} |
| @l | {pf21} |
| @m | {pf22} |
| @n | {pf23} |
| @o | {pf24} |
| @3 | {pf3} |
| @4 | {pf4} |
| @5 | {pf5} |
| @6 | {pf6} |
| @7 | {pf7} |
| @8 | {pf8} |
| @9 | {pf9} |
| @R | {reset} |
| @Z | {right} |
| @T | {righttab} |
| @A@H | {sysreq} |
| @T | {tab} |
| @U | {up} |

# Appendix C: Specialized HLLAPI Extensions

This section contains the following topics:



## What Is HLLAPI?

When there is a need to provide advanced support for manipulating 3270 data, eTrust SSO scripts can use the special 3270-specific extensions that incorporate HLLAPI. These extensions are described here and they are detailed in the chapter "eTrust SSO Extensions."

HLLAPI (High Level Language Application Programming Interface) is an IBM application programming interface, supported by all major 3270 emulation products on the market. (It should be noted that there are a number of variants of HLLAPI on the market. eTrust SSO supports the most common of them, EHLLAPI.)

HLLAPI enables a PC application program to interact with a 3270 emulation product. Using HLLAPI, an application can put text into, and get text from, the 3270 window. An application can also manipulate the cursor, wait for events to occur, and perform many other functions.

eTrust SSO HLLAPI extensions are built out of standard HLLAPI calls. Because of this:

- You should become familiar with the basic HLLAPI terminology, as used in this book and in some of the SSO messages.
- You should know, in detail, how the HLLAPI environment is configured on user workstations and how this affects scripts for 3270 applications. For more information, see the *eTrust SSO Administration Guide.*
- You should know how to set your emulation program's options, and sometimes your SSO initialization variables, so that your SSO scripts can use HLLAPI. See the documentation of your emulation product.
- Some of the warning and error conditions you may encounter during script development will be HLLAPI warnings or errors, rather than SSO warnings or errors.

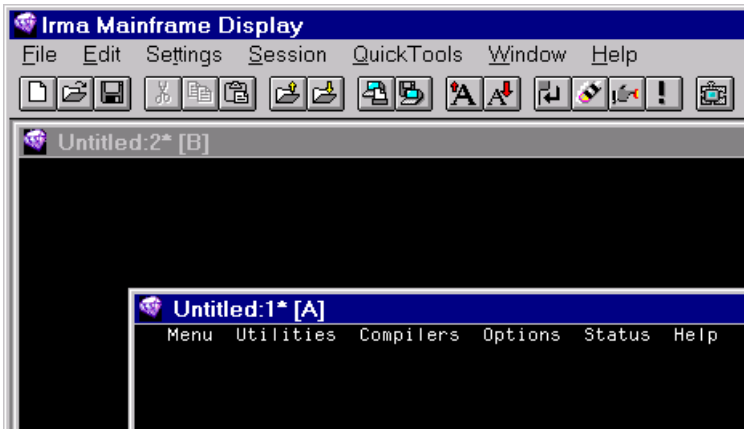
## Basic HLLAPI Terminology

The concepts explained in this section include session, presentation space, session short name, connect session, and disconnect session.

**Session**

Most 3270 emulations let you work concurrently on more than one emulated 3270 terminal. Each virtual terminal is usually represented in a different window and is called a *session*. Each session can connect with a different mainframe and have different attributes. A PC can, of course, run several sessions on the same mainframe.

Here is an example of a 3270 emulation program running two sessions in one open emulation window; each session is in its own document window:




**Presentation Space (3270 Screen)**

A *presentation space*, commonly referred to as the *3270 screen*, is the graphic image that represents the 3270 dumb terminal. It can be the contents of a regular window or the contents of a session document window (that is the document window without the title and frame). The presentation space contains:

- The data area, which is the large upper area and includes the session's text and the fields.
- The Operator Information Area (OIA), which is a single row of symbols and indicators, located across the bottom of the window under the data area.

### Session Short Name (PS ID)

HLLAPI requires that each session be associated with a unique identifier, called a *short name* or, in some emulation programs, a *presentation space ID* (*PS ID*). The short name is a single letter from a to z. The method of assigning short names to sessions differs from one 3270 emulation to another. In some emulations, the short name is assigned automatically when you open a new session, and in others, the short name has to be explicitly assigned. Refer to the documentation of the emulation you are using to obtain the relevant information.

Most emulations also let you give sessions *long names*. However, long names have no meaning in HLLAPI.

### Connect Session

The phrase *connect session* has two different meanings and it is important to distinguish between them:

- The first meaning of the phrase is *emulation connect.* This is the process of opening a new session by the 3270 emulation program. The word connect here refers to the connection between the PC and the mainframe host. In this context, the session is considered to be connected when you see the mainframe data on your workstation. This book will use "emulation connect" when it refers to this meaning.
- The second meaning is *HLLAPI-connect*. This is the initialization process that enables HLLAPI to work with an existing emulation-connected session. The word connect here refers to the connection, occurring within the workstation boundaries, between the application program using HLLAPI (that is, eTrust SSO) and the 3270 emulation session. This is the meaning within the HLLAPI context. In this book, unless otherwise noted, the phrase connect session refers to the second meaning, the HLLAPI-connect meaning of the phrase.

In eTrust SSO, you use the hllapi_connect extension to HLLAPI-connect a session. For example, to HLLAPI-connect the session identified by the short name a, assuming it is already emulation-connected, you type:

```
sso hllapi_connect –session a
```

A typical eTrust SSO script that works with a mainframe application will:

- Invoke the 3270 emulation (using the run extension)
- Emulation-connect a session (that is, create the connection between the PC and the mainframe host). Usually, this connection will be done in the same run extension, by passing parameters to the 3270 emulation.

- HLLAPI-connect the same session, using the hllapi_connect extension (that is, create the connection between the eTrust SSO script and the 3270 opened session).

Some 3270 emulations allow you to HLLAPI-connect a session to SSO using the hllapi_connect extension without first emulation-connecting the session, or even without first executing the 3270 emulation. Actually, this is only a shortcut for convenience. The full process is performed in the background: the emulation is executed, a session is emulation-connected to the host, and then the session is HLLAPI-connected to eTrust SSO.

**Disconnect Session**

The two meanings of connect session, mentioned previously, also apply to disconnect session.

When you *emulation-disconnect* a session, you terminate the connection between the workstation and the mainframe host. Usually, there will be no reason for eTrust SSO to emulation-disconnect a session.

When you HLLAPI-disconnect, you end the connection between your SSO script and the 3270 emulation session. When disconnect session is used in this book, it refers to HLLAPI-disconnect.

The SSO hllapi_disconnect extension performs the HLLAPI disconnection. The hllapi_disconnect extension disconnects the last HLLAPI-connected session (the one with the short name specified in the last hllapi_connect), and therefore it does not require a session short name argument. Here is an example of an hllapi_disconnect command:

```
sso hllapi_disconnect
```

If you run hllapi_connect twice, each command for a different session, you do not need to insert an hllapi_disconnect before the second hllapi_connect. The first session is automatically disconnected when you perform the second hllapi_connect. For example:

```
# execute the 3270 emulation, and open two sessions
sso run -args sess-a \
        -path HLLAPI_emulation_pathname
sso run -args sess-b \
        -path HLLAPI_emulation_pathname
...
 # HLLAPI-connect session a
sso hllapi_connect –session a
...
 # HLLAPI-disconnect session a, and connect session b
sso hllapi_connect –session b
...
 # HLLAPI-disconnect session b
sso hllapi_disconnect
```

### HLLAPI Function Number and Return Code

Standard HLLAPI, as explained above, enables an application program to perform operations such as connecting or disconnecting a session, manipulating text in the presentation space, manipulating the cursor, and waiting for events to occur. Each of these operations is identified by a unique number, called the *HLLAPI function* number. Every HLLAPI function, after being invoked, returns a standard HLLAPI return code that indicates the results of the operation. Most HLLAPI functions give a return code of 0 to indicate a successful completion. A return code greater than zero usually indicates a problem.

## eTrust SSO Extensions for HLLAPI

eTrust SSO provides extensions for specialized manipulation of 3270 data. These extensions, called HLLAPI extensions, use HLLAPI functions.

### HLLAPI Extension Prefix

All the HLLAPI extensions begin with the hllapi prefix. This prefix differentiates HLLAPI extensions from the general or Windows extensions that may have similar names. For example, there is a Windows setfield extension and there is an HLLAPI hllapi_setfield extension. These extensions perform similar functions, but in different environments, and have to be uniquely identified.

### Session-Level HLLAPI Extensions

HLLAPI extensions that operate on the session level relate to the session as a whole, and not to specific elements within the session (fields, text, and cursor).

#### hllapi_connect

Use the hllapi_connect extension to HLLAPI-connect your SSO script to an existing 3270 emulation session. For more information and examples, see Connect Session.

#### hllapi_disconnect

Use the hllapi_disconnect extension to HLLAPI-disconnect your SSO script from an existing 3270 emulation session. For more information and examples, see Disconnect Session.

### hllapi_waitsys

In mainframe applications, the user's keyboard is locked while the host is responding to the last workstation activity. The hllapi_waitsys extension waits until this processing is finished or until the current timeout value is reached. Normally, you will not need to use the hllapi_waitsys extension because it is used internally by most of the other HLLAPI extensions, or because many 3270 emulations support the type-ahead mode, which buffers keystrokes even though the mainframe has locked the keyboard.

Nevertheless, you might use hllapi_waitsys when you want the user to type something in the 3720 screen and you want to make sure that the keyboard is ready for typing when the user is prompted to type. For example:

```
...
sso hllapi_type –text “{enter}”
sso hllapi_waitsys
sso msgbox -msg "Now you may start typing"
```

### hllapi_getscreen

Use the hllapi_getscreen extension when you want to assign all the contents of the 3270 screen to a Tcl variable. This extension could be used when you have to perform a data analysis of the 3270 screen (for which the regular hllapi_getfield extension is not suited). For example:

```
set screen [sso hllapi_getscreen]
set first_10 [string range "$screen" 0 9]
sso msgbox -msg "The first 10 characters of the \
3270 screen are: $first_10"
```

Another use of the hllapi_getscreen extension is to find the fields' textual content together with their attributes. In the 3270 presentation protocol, field attributes are the numeric values that determine the field type (input, output, etc.), field intensity, and other display factors.

You can obtain the field attributes by using the -attrb switch of the hllapi_getscreen extension. Refer to your 3270 emulation's HLLAPI documentation or to IBM's 3270 data stream documentation for a list of 3270 field attribute values and meanings.

## Field-Level HLLAPI Extensions

The term *field* in the 3270 environment refers to something fundamentally different from the same term in the Windows environment. In Windows, fields are graphical elements that can be placed in random locations on the two-dimensional area of a graphical window or message box. They are separated from one another by text or by empty spaces and there is no significance to the spaces between them. In the alphanumeric 3270 screen, fields and field attributes are the building blocks of a long one-dimensional string. This string is made up of all the screen's rows (typically 80 characters to a row), concatenated one after the other from the top down. This long string is divided into segments, called fields, which are connected to each other by invisible characters, called *field attributes*. These field attributes determine, among other things, if the following field is an input field, where the user is able to type, or an output field, where the user's keyboard is blocked.

Every single character in the 3270 screen is either part of a field or part of a field attribute. Many fields are invisible, because they are output fields without any text in them and without any visible indication where they begin and end. Fields may span more than one row.

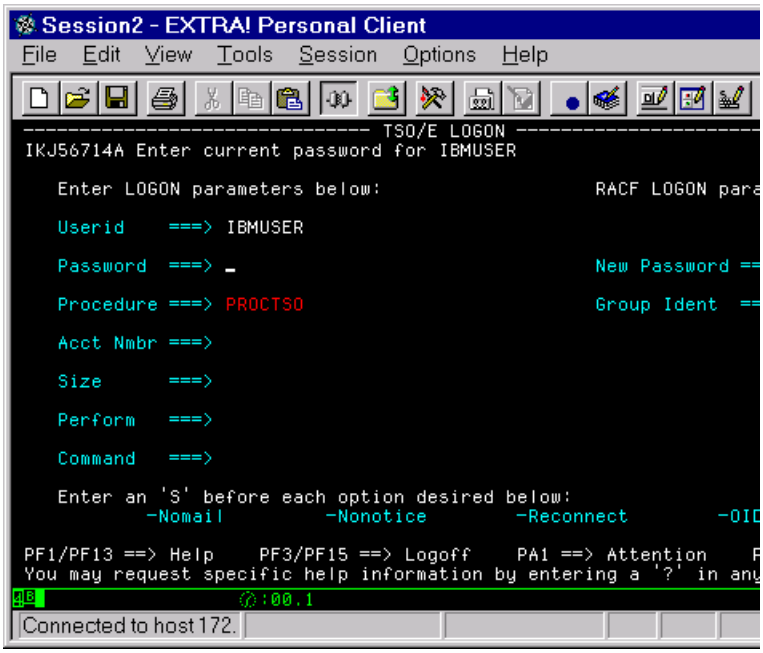
eTrust SSO provides two ways to refer to a field. One way is by specifying a text that is located in a previous field. A previous field is a field that is located either in a higher row than the wanted field's row or in the same row to the left of the wanted field.

Unlike Windows field notation, the text that is specified here to identify the field is itself always a part of another field (either an input or an output field). The text must be located on the 3270 screen (in the screen string) *before* the wanted field, so there is no meaning to the direction specifications available in the Windows field notation (-pos right|left|top|bottom). In the HLLAPI extensions, there is also no support for the text matching options that are supported by the Windows extensions.

The other way is by specifying an exact position (row and column notation) for one of the characters in the wanted field.

Unlike the Windows field notation, there is no way to specify the ordinal number of the field from the beginning of the screen. The reason for this is that, as mentioned before, many fields are invisible, and it is almost impossible to know the ordinal number of the wanted field.

There are some differences between the meanings of the field specifications in the hllapi_getfield and hllapi_setfield extensions, as explained in the following subsections. All the examples in the following two subsections relate to this 3270 screen (the TSO/E Logon panel):




## hllapi_getfield

Use the hllapi_getfield extension to get the textual content of a 3270 field. The target field can be either an input or an output field. As explained above, there are two alternative notations for specifying the wanted field.

The first method of notation is to specify a text string that is contained within a field in front of the wanted field (this text can be called the *caption* of the field). When this notation is used as a hllapi_getfield argument, the extension searches the entire 3270 screen until the specified text is found, or until the timeout period expires. If the specified text is found, the hllapi_getfield extension locates the next field and retrieves its contents. For example:

```
set userid [sso hllapi_getfield -label "Userid    ===>"]
```

assigns userid the value of the field following Userid ===>.

If -fromx ,-fromy switches are used, then hllapi_getfield searches for the specified text from the points designated by the -fromx, -fromy parameters to the end of the screen rather than searching the entire 3270 screen.

As shown in this example, you should specify not only the field caption itself (Userid), but also any other embedded text ( ===>) in order to provide a unique description of the wanted field. This helps prevent errors if there is other similar text on the screen.

The -posx, -posy switches can improve the accuracy with which hllapi_getfield identifies the wanted field. If -posx, -posy are specified, then hllapi_getfield searches for the specified text only at the specified position. If the specified text is present, then the hllapi_getfield extension locates the next field and retrieves its contents. The ability to specify both text and the -posx, -posy switches allows you to be sure that you are retrieving the contents of the correct field, and not that of another field that happens to be located next to a text similar to the text you were looking for. For example:

```
set userid [sso hllapi_getfield -label “Userid” -posx 6 -posy 5
```

In this example, if the string Userid is present in another position before the wanted field, hllapi_getfield ignores the other Userid and uses only the Userid that is located in row 6 column 5.

The second method of notation is to specify just the exact position (row and column) of the wanted field, or the position of a part of the wanted field (if the position is inside the field). When this notation is used as an hllapi_getfield argument, the hllapi_getfield extension retrieves the data from the specified position to the end of the field. This notation is useful when it is not possible to specify a fixed caption before the wanted field. For example:

```
set tso_msg [sso hllapi_getfield -label “” -posx 2 -posy 2]
```

In this example, since no text was specified (as indicated by the empty string argument), hllapi_getfield goes directly to the designated position (row 2 and column 2), which is the location of the TSO message, and retrieves the contents of the field from that designated position to the end of the field.

### hllapi_setfield

The hllapi_setfield extension sets the textual content of a 3270 input field. As above, there are two alternative notations for specifying the wanted field:

The first method of notation is to specify a text string that is contained within a field in front of the wanted field (this text can be called the *caption* of the field). When this notation is used as an hllapi_setfield argument, the extension searches the entire 3270 screen until the specified text is found, or until the timeout value is reached. If hllapi_setfield finds the text, hllapi_setfield locates the next input field (even if it is not the very next field), Finally, it inserts the specified string at the beginning of that input field. For example:

```
sso hllapi_setfield -label "Password  ==>" -value $_PASSWORD
```

puts the value of $_PASSWORD into the first input field following Password ===>.

If -fromx, -fromy switches are used, then the hllapi_setfield extension searches for the specified text from the points given by -fromx, -fromy to the end of the screen and rather than searching the entire 3270 screen.

As shown in this example, you should specify not only the field caption itself (Userid), but also any other embedded text ( ===>) in order to provide a unique description of the wanted field. This helps prevent errors if there is other similar text on the screen.

The -posx, -posy switches can improve the accuracy with which hllapi_setfield identifies the wanted field. If -posx, -posy are specified, then hllapi_setfield does not search for the specified text, but rather ensures that the specified text is located at the specified position. If the specified text is present, the hllapi_setfield extension locates the next input field and assigns it the designated value. This option (of specifying both text and the -posx, -posy switches) is useful when you want to make sure that you are setting the correct field, and not another input field that follows a text that is similar to the text you are looking for. For example:

```
sso hllapi_setfield -label “Password” -value $_PASSWORD -posx 8 -posy 5
```

In this example, if the string Password is present in another position before the wanted field, hllapi_setfield ignores the other Password and uses only the Password that is located in row 8 column 5.

The second method of notation is to specify just the exact position (row and column) of the wanted field, or the position of a part of the wanted field (if the position is inside the field). When this notation is used as a hllapi_setfield argument, the hllapi_setfield extension retrieves the data from the specified position to the end of the field. This notation is useful when it is not possible to specify a fixed caption before the wanted field or when you want to set only part of a field. For example:

```
sso hllapi_setfield -label "" -value "ispf" -posx 10 -posy 24
```

In this example, since no text was specified (as indicated by the empty string argument), hllapi_setfield goes directly to the designated position (row 10 and column 24) which is the location of the fifth character of the Procedure field, and inserts the specified string ("ispf") beginning at the specified position.

## Text-Level HLLAPI Extensions

### hllapi_waittext

The hllapi_waittext extension suspends script operation until a specific text appears in the 3270 screen or until the timeout value is reached (in which case it returns an error value).If the specified text already exists in the current screen, the hllapi_waittext will not perform a wait.

The syntax of the hllapi_waittext extension is similar to the syntax of the hllapi_getfield and hllapi_setfield extensions. However, remember that hllapi_getfield and hllapi_setfield work on fields, while hllapi_waittext works only on simple text strings.

Text that is an argument in the hllapi_waittext extension will be searched for in the whole 3270 screen until it is found, or until the timeout period expires. If the -fromx, -fromy switches are specified, only part of the screen is searched.

The following example waits until the appearance of ***, which is the TSO end-of-line-mode-output sign:

```
sso hllapi_waittext -text "***"
```

If the -pos switch is specified, hllapi_waittext checks for the specified text at the specified position and if it is not present, waits until it appears (or until the end of the timeout period).

This option is useful when you expect the specified text to be located at a known position and there is a possibility that the same text will also appear at some other location in the 3270 screen. For example:

```
sso hllapi_waittext –text “ISPF Primary Option Menu" –posx 3 –posy 29
```

This example waits for the ISPF primary menu to appear, by specifying the text and the expected position of this panel title.

## Cursor-Level HLLAPI Extensions

The cursor-level HLLAPI extensions set a new cursor position or retrieve the existing cursor location.

### hllapi_setcursor

Use the hllapi_setcursor extension to set the 3270 cursor location. The following example puts the cursor at row 12 and column 5:

```
sso hllapi_setcursor -offsx 12 -offsy 5
```

## hllapi_getcursor

Use the hllapi_getcursor extension to get the current location of the 3270 cursor. The cursor location is returned in two ways as described below.

One way is as the return value, a string formatted as *rr ccc*. You can later use this return value as an input to the hllapi_setcursor extension. The other way is in the Tcl variables _ROW and _COL. This option avoids having to extract the row and the column values from the return value string that was mentioned above.

Here is an example of invoking the hllapi_getcursor extension and using its outputs:

```
# get current cursor position
sso hllapi_getcursor

# save the current cursor position
set save_cursor_posx $_ROW
set save_cursor_posy $_COL

# set the cursor to a new relative position

sso hllapi_setcursor –offsx [expr $_ROW+1] –offsy [expr $_COL+2]
 # the script continues executing

# restore the original cursor position
sso hllapi_setcursor –offsx $save_cursor_posx –offsy $save_cursor_posy
```

## Simulating Operator Keystrokes

### hllapi_type

Use the hllapi_type extension to simulate a user typing in the 3270 environment. As opposed to the hllapi_setfield extension, where you can only write text in the input field, the hllapi_type extension lets you perform any 3270 keyboard operation, including pressing control keys such as Enter, Attn, Tab, Reset, and the PF keys.

HLLAPI uses a special representation for control keys, but this representation is not intuitive and not easy to use. For example, the HLLAPI representation for the Duplicate key is @S@X. To remedy this situation, eTrust SSO provides clearer representations, called *special mnemonics*, for these keys. For example, the eTrust SSO special mnemonic for the Duplicate key is {dup}.

Here is an example of a hllapi_type extension that uses eTrust SSO special mnemonics:

```
sso hllapi_type –text "{tab}123{enter}"
```

This example makes sure that the keyboard is not locked, using the Reset function; jumps to the next input field, using the Tab function; writes the text 123 in this field; and presses the Enter key.

When it is possible, you should use the hllapi_setfield extension rather than the hllapi_type extension. Consider, for example, the following rewrite of the previous example, where the caption of the wanted field is USER NAME:

```
sso hllapi_setfield –label “USER NAME” –value “123”
 sso hllapi_type –text “{enter}”
```

While this example is longer, it is much more stable and easier to maintain, because hllapi_setfield is independent of the cursor position at the time it starts executing. However, you can use hllapi_setfield only if you know the caption or the position of the wanted field. Otherwise, you will have to use the hllapi_type extension.

# The HLLAPI Environment for Scripting

Ensure that the HLLAPI environment has been set up according to the instructions in Setting Up the HLLAPI Environment in the appendix “Support for HLLAPI Environments” in the *eTrust SSO Administration Guide*.

The 3270 sessions with which your script will work must be associated with HLLAPI short names. Refer to your 3270 emulation documentation for an explanation of how to do this for the emulation you are running.

## 3270 Troubleshooting

A single SSO HLLAPI extension is usually built out of several HLLAPI function calls. If a HLLAPI function fails for some reason, it returns a bad return code, designating the problem that occurred. When eTrust SSO determines that such a problem exists, eTrust SSO sets the Tcl variable _SSOERR to 70, indicating a HLLAPI problem. The script execution stops at this stage, and a message appears, indicating the failed HLLAPI function number and the HLLAPI return code.

If the script is running with _ERRORMODE resume (or msg), script execution will continue, and SSO will set two special Tcl variables: _HLLAPI_FUNC_NO, which contains the HLLAPI *function number* of the function that failed; and _HLLAPI_RC, which contains the HLLAPI *return code* of the function that failed.

### Example

Suppose your script contains the hllapi_disconnect extension, without any previous hllapi_connect. If the default error mode setting is in effect, the script will stop execution, and display an error message.

This error message gives the HLLAPI function number 2 (representing the Disconnect Presentation Space function), and the HLLAPI return code 1. In this appendix, you will find that if the HLLAPI function number is 2, the HLLAPI return code 1 means "Not connected to the presentation space." To fix this problem, use this command in the script somewhere before the hllapi_disconnect:

```
sso hllapi_connect –session a
```

In some cases, an error message might still appear. A HLLAPI function number of 1 with a HLLAPI return code of 1 means "invalid short session ID parameter for the host presentation," that is, the specified session short name (a) was invalid. Generally, this would be due to one of the following reasons: there is currently no session that was emulation-connected (no connection between the workstation and the mainframe host), or one or more sessions that are emulation-connected to the host exist, but none of them has the short name of a. Either the sessions are associated with different short names, or they are not associated with any short name at all.

To solve this problem, you change your 3270 emulation settings, and associate an emulation-connected session with the short name. Most emulations provide you with the means to save these settings, so the next time you emulation-connect this session, the session will automatically be associated with the short name a.

Another common error gives an error code of 11 for the Connect Presentation Space function (HLLAPI function number 1). The error message indicates that the host presentation space is already being used by another user. This could be the result of a previous script failing before a disconnect command was sent, so that the Presentation Space remains connected. One way of handling the problem is to shut down the emulation program and then to restart it.

# Standard HLLAPI Return Codes

The codes in the table below are the standard HLLAPI return codes. Use them when you get an _SSOERR of 70 and you have to find the meaning of the _HLLAPI_RC value for a _HLLAPI_FUNCT_NO, and when the script stops because of a HLLAPI extension error and you get an error message giving the return code and the number of a failed HLLAPI function.

In this table, PS means Presentation Space, -the virtual 3270 screen-, and OIA means Operator Information Area, - the bottom line of the presentation space containing symbols indicating the 3270 session status-.

| Function Number | Function Name | Error Code | Error Code Description |
|---|---|---|---|
| 1 | Connect presentation space | 0 | The function was successful; host presentation space is connected and can accept input. |
| | | 1 | Invalid short session ID parameter for the host presentation. |
| | | 4 | Successful connection achieved, but the host presentation space is busy. |
| | | 5 | Successful connection achieved, but the host presentation space is input inhibited (locked). |
| | | 9 | System error occurred. |
| | | 11 | Resource unavailable. Host presentation space already being used by another user. One common cause of this error is that a previous session was not disconnected. |
| 2 | Disconnect presentation space | 0 | The function was successful. |
| | | 1 | Not connected to the presentation space. |
| | | 9 | System error occurred. |

| Function Number | Function Name | Error Code | Error Code Description |
|---|---|---|---|
| 3 | Send key | 0 | Keystrokes sent, status was normal. |
| | | 1 | Program not connected to host session. |
| | | 2 | Incorrect parameter passed to HLLAPI. |
| | | 4 | Host session busy and all the keystrokes could not be sent. |
| | | 5 | Input to target session inhibited; keystrokes were rejected or invalid keystroke mnemonics were sent. Not all the keystrokes could be sent. |
| | | 9 | System error occurred. |
| 4 | Wait | 0 | Keyboard unlocked and ready for input. |
| | | 1 | Application was not connected to a valid session. |
| | | 4 | Time-out while still busy for 3270 emulation. |
| | | 5 | Keyboard locked. |
| | | 9 | System error occurred. |
| 5 | Copy presentation space | 0 | Host presentation space copied to application program. Source presentation space was active and keyboard was unlocked. |
| | | 1 | Program not connected to host session. |
| | | 4 | Successful. Connect host presentation space was waiting for host response. |
| | | 5 | Successful. The keyboard was locked. |
| | | 9 | System error occurred. |
| 6 | Search presentation space | 0 | The function was successful. |
| | | 1 | Host presentation space not connected. |
| | | 2 | Invalid parameters. |
| | | 7 | Host presentation space invalid. |
| | | 9 | System error occurred. |
| | | 24 | Search string not found. |

| Function Number | Function Name | Error Code | Error Code Description |
|---|---|---|---|
| 7 | Query cursor location | 0 | The function was successful. |
| | | 1 | Program was not connected to the host session. |
| | | 9 | System error occurred. |
| 8 | Copy presentation space to string | 0 | Successful. Host presentation space contents copied to eTrust SSO. Your target presentation space was active and the keyboard was unlocked. |
| | | 1 | Program was not connected to the host session. |
| | | 2 | String length of 0 was specified. |
| | | 4 | Successful. Host presentation space was waiting for host response. |
| | | 5 | Successful. Keyboard was locked. |
| | | 7 | Host presentation space position was invalid. |
| | | 9 | System error occurred. |
| 9 | Set session parameters | 0 | Session parameters were set. |
| | | 2 | One or more parameters invalid. |
| | | 9 | System error occurred. |
| 10 | Query sessions | 0 | Function successful. |
| | | 1 | No session has been configured. |
| | | 2 | String length invalid. |
| | | 9 | System error occurred. |
| 11 | Reserve | 0 | Function successful. |
| | | 1 | Program was not connected to the host session. |
| | | 5 | Presentation space was inhibited. |
| | | 9 | System error occurred. |
| 12 | Release | 0 | Function successful. |
| | | 1 | Program was not connected to the host session. |

| Function Number | Function Name | Error Code | Error Code Description |
|---|---|---|---|
| | | 9 | System error occurred. |
| 13 | Copy OIA | 0 | Function successful. |
| | | 1 | Program was not connected to the host session. |
| | | 2 | Error in specifying string length. OIA data was not returned. |
| | | 4 | OIA data returned, but host presentation space is busy. |
| | | 5 | OIA data was returned, but the host presentation space is input inhibited (locked). |
| | | 9 | Internal system error occurred. |
| 14 | Query field attribute | 0 | Function successful. |
| | | 1 | Program was not connected to the host session. |
| | | 7 | Host presentation space position was invalid. |
| | | 9 | System error occurred. |
| | | 24 | Attribute byte was not found (presentation space was unformatted). |
| 15 | Copy string to presentation space | 0 | Function successful. |
| | | 1 | Program was not connected to the host session. |
| | | 2 | Parameter error. String length of 0 specified. |
| | | 5 | Target presentation space was protected or inhibited, or illegal data was sent to target presentation space (possibly a field attribute byte). |
| | | 6 | Copy was completed, but the data was truncated. |
| | | 7 | Invalid host presentation space. |
| | | 9 | System error occurred. |

| Function Number | Function Name | Error Code | Error Code Description |
|---|---|---|---|
| 18 | Pause | 0 | Wait duration has expired. |
| | | 2 | Parameter error occurred. |
| | | 9 | Internal system error occurred. Time results are unpredictable. |
| | | 26 | Host session presentation space or OIA was updated. |
| 20 | Query system | 0 | Function successful. A data string was returned. |
| | | 9 | System error occurred. |
| 21 | Reset system | 0 | Function successful. |
| | | 1 | EHLLAPI not loaded. |
| | | 9 | System error occurred. |
| 22 | Query session status | 0 | Function successful. |
| | | 1 | Invalid session requested. |
| | | 2 | Invalid string length. |
| | | 9 | System error occurred. |
| 23 | Start host notification | 0 | Function successful. |
| | | 1 | Invalid host presentation space requested. |
| | | 2 | Error in designating parameters. |
| | | 9 | System error occurred. |
| 24 | Query host update | 0 | No updates made since the last call. |
| | | 1 | Invalid host presentation space requested. |
| | | 8 | No prior start host notification (23) function was called for the host presentation space ID. |
| | | 9 | System error occurred. |
| | | 21 | OIA was updated. |
| | | 22 | The presentation space was updated. |
| | | 23 | The OIA and the host presentation space were updated. |

| Function Number | Function Name | Error Code | Error Code Description |
|---|---|---|---|
| 25 | Stop host notification | 0 | Function successful. |
| | | 1 | Invalid host presentation space specified. |
| | | 8 | No prior start host notification (23) function was issued. |
| | | 9 | System error occurred. |
| 30 | Search field | 0 | Function successful. |
| | | 1 | Program not connected to host session. |
| | | 2 | Error made in specifying parameters. |
| | | 7 | Invalid host presentation space position. |
| | | 9 | System error occurred. |
| | | 24 | Search string not found or host PS was unformatted. |
| 31 | Find field position | 0 | Function successful. |
| | | 1 | Program not connected to host session. |
| | | 2 | Error made in specifying parameters. |
| | | 7 | Invalid host presentation space position. |
| | | 9 | System error occurred. |
| | | 24 | No such field found, or presentation space is unformatted. |
| | | 28 | Field was 0. |
| 32 | Find field length | 0 | Function successful. |
| | | 1 | Program not connected to host session. |
| | | 2 | Error made in specifying parameters. |
| | | 7 | Invalid host presentation space position. |
| | | 9 | System error occurred. |
| | | 24 | No such field found, or presentation space is unformatted. |
| | | 28 | Field had zero bytes. |
| 33 | Copy string to field | 0 | Function successful. |

| Function Number | Function Name | Error Code | Error Code Description |
|---|---|---|---|
| | | 1 | Program not connected to host session. |
| | | 2 | Error made in specifying parameters. String length of 0 was specified. |
| | | 5 | Target field protected or inhibited, or illegal data sent to target field (for example, field attribute). |
| | | 6 | Copy was completed, but data was truncated. |
| | | 7 | Host presentation space position invalid. |
| | | 9 | System error occurred. |
| | | 24 | Host presentation space was unformatted. |
| 34 | Copy field to string | 0 | Function successful. |
| | | 1 | Program not connected to host session. |
| | | 2 | Error made in specifying parameters. |
| | | 6 | Data to be copied and source not the same size. Data is truncated if the string length was smaller than the field copied. |
| | | 7 | Invalid host presentation space position. |
| | | 9 | System error occurred. |
| | | 24 | Host presentation space was unformatted. |
| 40 | Set cursor | 0 | Cursor placed at specified position. |
| | | 1 | Program not connected to host session. |
| | | 4 | Session was busy. |
| | | 7 | Invalid parameter. Specified cursor location was less than one or greater than the maximum presentation space size. |
| | | 9 | System error occurred. |
| 41 | Start close intercept | 0 | Function was successful. |
| | | 1 | Incorrect host presentation space was specified. |
| | | 2 | Parameter error occurred. |
| | | 9 | System error occurred. |

| Function Number | Function Name | Error Code | Error Code Description |
|---|---|---|---|
| | | 10 | Function not supported by the emulation program. |
| 42 | Query close intercept | 0 | Close intercept event did not occur. |
| | | 1 | Presentation source was not valid. |
| | | 2 | Parameter error. |
| | | 8 | No prior Start Close Intercept function was called for this host presentation space. |
| | | 9 | System error occurred. |
| | | 12 | Session stopped. |
| | | 26 | A close intercept took place since the last query close intercept call. |
| 43 | Stop close intercept | 0 | Function successful. |
| | | 1 | Program not connected to host session. |
| | | 2 | Error in parameter specification. |
| | | 8 | No previous Start Close Intercept function was issued. |
| | | 9 | System error. |
| | | 12 | Session stopped. |
| 50 | Start keystroke intercept | 0 | Function successful. |
| | | 1 | Invalid presentation space. |
| | | 2 | Invalid option. |
| | | 4 | Resource unavailable. Requested presentation space was being used by another API application. |
| | | 9 | System error occurred. |
| 51 | Get key | 0 | Function successful. |
| | | 1 | Invalid presentation space. |

| Function Number | Function Name | Error Code | Error Code Description |
|---|---|---|---|
| | | 5 | You specified D option for AID keystrokes only in start keystroke intercept (50). Non-AID keys are inhibited by this session when attempt is made to write invalid keys in presentation space. |
| | | 8 | No prior start keystroke intercept (50) function was called for the presentation space. |
| | | 9 | System error occurred. |
| | | 20 | Invalid combination of keys for this presentation space session. |
| | | 25 | Requested keystrokes not available on the input queue. |
| | | 31 | Keystroke queue overflowed and keystrokes were lost. |
| 52 | Post intercept status | 0 | Function successful. |
| | | 1 | Invalid presentation space. |
| | | 2 | Invalid option specified. |
| | | 8 | No prior start keystroke intercept (50) function was called for the presentation space short session ID. |
| | | 9 | System error occurred. |
| 53 | Stop keystroke intercept | 0 | Function successful. |
| | | 1 | Invalid host presentation space. |
| | | 8 | No prior start keystroke intercept (50) function was called for the presentation space. |
| | | 9 | System error occurred |
| 99 | Convert position or convert rowcol | 0 | Incorrect column, row, or position input was provided. |
| | | >0 | The PS position or column |
| | | 9998 | Invalid host presentation space or system error. |

| Function Number | Function Name | Error Code | Error Code Description |
|---|---|---|---|
| | | 9999 | Character 2 in the data string is not P or R. |

# Appendix D: Advanced Tcl Language

This section contains the following topics:



## Lists

Tcl enables you to group strings into lists. Each string is an element of the list, but when necessary, a string can be extracted from the list and processed individually.

A Tcl list consists of zero or more elements, with spaces as separators. You can override a space or spaces between elements by using braces or backslashes. To illustrate:

| List | Notes |
|---|---|
| red green blue | The list consists of three items. |
| a b {c d e} f | The list consists of four items. The braces make one item from what would otherwise be three. |
| one\ word two three | The list consists of three items one\ word, two, and three. The backslash causes the space that follows it to be treated like any alphanumeric character. |

Like variables, lists do not need to be declared before you can use them. But as a rule, a list that appears literally as a command argument needs to be enclosed in braces, to keep it from seeming like more than one argument.

## Commands for Lists

Tcl includes several commands for handling lists:

## concat

The concat command receives one or more lists as arguments and combines them into a single list. For example:

```
concat {red blue} {black white gray} clear
```

The above command returns a single six-element list.

## foreach

The foreach command is a loop command that accesses each element in a list in the list's order. Its arguments are a variable name (to represent the current list element), a list, and a script to be executed for each list element. For example, the following command uses the string length command to record the length of each list element:

```
set colors {red green blue clear bronze}
foreach thiscolor $colors {
  sso msgbox –msg [string length $thiscolor]
 }
```

## lappend

The lappend command adds list elements to the end of a new or existing list variable. Its arguments are the list variable name and the new elements. For example:

```
set colors {red green blue}
lappend colors black white
```

After lappend is executed, the list colors contains the following elements:

```
red green blue black white
```

## lindex

The lindex command returns one of the elements of a list. The command has two arguments: the list and the index number of the requested element. The first element of the list is element 0. For example:

```
set letters {a b c d e f g}
lindex $letters 2
```

returns c.

## linsert

The linsert command produces a new list from a list by inserting all of the element arguments just before the indexed element of list. Each element argument will become a separate element of the new list.

The linsert command has three arguments; the list, the index number to be assigned to the first inserted element, and the elements to be inserted. For example:

- set colors {red green blue}
- sso msgbox -msg $colors
- set new_colors [linsert $colors 1 black white]
- sso msgbox -msg $new_colors

After linsert is executed, the new list contains the following elements: red black white green blue.

## list

The list command returns a list that consists of all the arguments that follow the command. For example:

```
list one two "two and a half"
```

The above command returns a three-element list. Unlike the concat command, the list command makes each argument a single element of the new list, even if the argument itself is a list.

## llength

The llength command returns the number of elements in the specified list. For example:

```
set sizes {one two "two and a half"}
llength $sizes
```

returns the value 3.

## lrange

The lrange command returns an extract of a list. Its arguments are a list, the index of the first element to be returned, and either the index of the last element to be returned or the value end (to return all the remaining elements of the list). For example:

```
set days {su mo tu we th fr sa}
lrange $days 1 5
```

returns the list:

```
mo tu we th fr
```

## lreplace

The lreplace command returns a list in which a range of elements has been replaced. Its arguments are the list, the indices of the first and last elements to be replaced, and the new elements to replace them. The number of new elements need not equal the number of replaced elements. You can even omit the last argument in order to delete the specified range of elements.

For example:

```
set times {0800 0900 1100 1300 1600}
lreplace $times 1 3 1015 1315
```

returns the list:

```
0800 1015 1315 1600
```

```
set times {0800 0900 1100 1300 1600}
lreplace $times 2 2
```

returns the list:

```
0800 0900 1300 1600
```

## lsearch

The lsearch command searches a list for the first element that matches a specified pattern, and returns the index of that element. Its arguments are a pattern-matching method (optional switch), the list, and the pattern. The three pattern matching methods are:

- -exact (Exact matching, character for character)
- -glob (Glob-style matching; this is the default)

Together with the characters that are to be matched literally, glob matching can use the following special characters in the pattern:

| Special characters | Definitions |
|---|---|
| * (asterisk) | The asterisk is used to match any string of characters, any one character, or the absence of a character. |
| ? (question mark) | The question mark is used to match any one character. |
| [ ] (square brackets) | The square brackets are used to match any one character in a series delimited by the brackets. The series may be abbreviated as a range, using a hyphen. For example, [abcd] matches a, b, c, or d, and so does [a-d]. If you use brackets, use braces around your pattern. |
| \ (backslash) | The backslash is used to match the character that follows the backslash, without treating that character as a special character. For example, * matches only the asterisk, rather than matching other characters as the asterisk normally does in glob-style matching. |

- -regexp (Regular-expression matching)

  **Note:** Although -regexp is not used in eTrust SSO extensions, it can be used in native Tcl commands.

Together with the characters to be matched literally, regular-expression matching can use the following special characters in the pattern:

| Special characters | Definitions |
|---|---|
| . (period) | The period is used to match any one character. |
| ^ (caret) | The caret means that the next character is the first character; for example ^b matches the b in bcd but not in abcd. |
| $ (dollar sign) | The dollar sign means that the preceding character is the final character; for example c$ matches the c in abc but not in abcd. If you use the $ sign, use braces around your pattern. |

| Special characters | Definitions |
|---|---|
| \ (backslash) | The backslash is used to match the character that follows the backslash, rather than treating that character as a special character. For example, \. matches only the period, rather than matching other characters as the period normally does in regular-expression matching. |
| [ ] (square brackets) | The square brackets are used to match any one character in a series delimited by the brackets. The series may be abbreviated as a range, using a hyphen. For example, [abcd] matches a, b, c, or d, and so does [a-d]. If you use brackets, use braces around your pattern. |
| [^ ] (caret in square brackets) | The caret placed inside of square brackets are used to match any one character that is *not* in a series following the ^ sign. The series may be abbreviated as a range, using a hyphen. For example, [^abcd] matches anything except a, b, c, or d, and so does [^a-d]. If you use brackets, use braces around your pattern. |
| ( ) (parentheses) | Parentheses are used to match any regular expression delimited by the parentheses. They may be convenient, or even necessary, for separating parts of a complex regular expression. |
| \| (pipe symbol) | The pip symbol is used to match either the regular expression to the left of the \| sign or the regular expression to its right. For example, ^success \| ^error matches any string that begins with either success or error. |

The ^ (circumflex), $ (dollar), and . (period), used as special characters in regular expressions, are called *atoms*. The \ (backslash) and the character that follows it, together, are also an atom. Any range and any regular expression with delimiting parentheses are also atoms. The following special characters are used with atoms:

| Special Characters | Definitions |
|---|---|
| * (asterisk) | The asterisk is used to match one or more consecutive repetitions of the preceding atom, or no repetitions. |
| + (plus sign) | The plus sign is used to match one or more consecutive repetitions of the preceding atom. |

| Special Characters | Definitions |
|---|---|
| ? (question mark) | The question mark is used to match one repetition of the preceding atom, or no repetitions. |

A repetition needs only to match the regular expression, not the previous matching string. For example, the string 121 matches the regular expression [0-9] three times, even though 2 does not match 1.

If a given string matches a regular expression, any longer string containing the given string also matches the regular expression. For example, the string abcde matches the regular expression b.d. Note, though, that a match no longer exists if characters are added where ^ or $ must match the start or end of the string. For example:

```
set times {0809 0845 0915 0930 0945 1015}
lsearch -regexp $times ^09
```

The script above returns the value 2 (the index of 0915).

## lsort

The lsort command arranges a list in upward ASCII sequence, or in another sequence as specified by one or more optional arguments. Some of the optional arguments are:

- -integer
- -real
- -decreasing
- -command mysort (where mysort is a sorting function of your own)

Here is an example of decreasing ASCII sequence:

```
lsort -decreasing {Lauren Charles Nora Jon}
```

The above command returns the list Nora Lauren Jon Charles.

# Handling List Variables

Tcl treats list variables as either lists or strings. The way the variable will be handled depends on the context and on the command used. To illustrate this, observe that:

```
set letters {a b          c d}
lrange $letters 1 2
```

returns b c with all the intermediate spaces of the original string.

However, lindex $letters 1 and lindex $letters 2 return b and c respectively, without any trailing or leading spaces. This can be proved by running the following:

```
set test "[index $letters 1] [windex $letters 2]"
```

which returns b c with no extra intervening spaces.

# Advanced Tcl commands

The following are Tcl commands that can be used in complex scripts.

## break

The break command stops a loop immediately. It jumps to the end of the loop and then continues as if the loop's termination condition were satisfied. If the loop is inside a larger loop, the larger loop is unaffected.

For example:

```
foreach i $oldlist {
   if {$i == "STOP"} break
   append newlist $i
}
```

The example copies the elements of oldlist to newlist until the value of an oldlist element is STOP.

## continue

The continue command stops the current loop iteration but continues the looping. In a while command, it jumps to the next evaluation of the loop's condition. In a for command, it jumps to the reinitialization. Here is a continue example:

```
foreach i $oldlist {
   if {$i == "NONE"} continue
   append newlist $i
}
```

The example copies the each element of oldlist to newlist unless the value of the oldlist element is NONE.

## eval

The eval command treats a string as a Tcl command and returns the command's result. For example, eval "set a 122" returns the value 122. (Double quotes are used here to delimit the string because the string contains spaces.)

Although the eval command is not essential in this example (set a 122 is a simpler way to return the same value), in a more complex script the eval command can give the necessary control over parsing.

## uplevel

The uplevel command is like an eval command that works within the scope of the calling script. For example:

```
uplevel {set timer stop}
```

Inside a procedure, the above command gives the value stop to the variable timer in the script that called the procedure.

# Advanced String Manipulations

### format

The format command works like its C-language counterpart.

## join

The join command is the opposite of the split command. It unites a list to form a string, inserting the specified delimiter between the list elements, and returns the string. For example:

```
join "c:\\win c:\\sys c:\\apps" "\;"
```

The command above returns the string c:win;c:sys;c:\apps.

If you do not use a second argument, join simply concatenates the elements of the specified list to produce the resulting string.

## regexp

The regexp command tells you which parts of a string match which parts of a regular expression.

The first argument is the regular expression with parts of it parenthesized. For example:

```
\$([0-9])*\.([0-9][0-9])
```

represents a sum in dollars.

The second argument is the string that may match the regular expression.

If there is a match, then the third argument is set to the substring in the second argument that matched the regular expression.

The fourth and following arguments are the substrings that matched the parenthesized parts of the regular expression.

For example:

```
set price {"only $349.95"}
regexp ^\$([0-9])*\.([0-9][0-9])  \
        $price complete dollars cents.
```

The above script would return $349.95 as “complete,” 349 as “dollars,” and 95 as “cents.”

## regsub

The regsub command substitutes a specified string for a substring that matches a regular expression. As arguments it takes the regular expression, the string that might match the regular expression, the string to replace the substring that matches the regular expression, and a variable to receive the resulting string.

The regsub command returns 0 if no match was found, and 1 if a match was found and a substitution was made.

For example:

```
set reminder "Check our latest update"
regsub "latest" $reminder "January" janreminder
```

The example returns 1 and gives janreminder the value Check our January update.

### scan

The scan command works like its C-language counterpart.

### split

The split command divides a string into list elements, using a specified delimiter as a dividing point. The arguments are the string and the delimiter. The return value is the list. For example:

```
split "c:\\win\;c:\\sys\;c:\\apps" "\;"
```

The command above returns the list {c:\win} {c:\sys} {c:\apps}.

If you do not use a second argument, split divides the specified string into characters.

## Procedures with an Indeterminate Number of Arguments

The proc command can be modified so that the procedure will accept an indeterminate number of arguments. To do so, use the special value args as the last of the arguments before the script. Do not specify a default for args; the default for args is always an empty string.

Here is an example of a procedure with an indeterminate number of arguments. It returns the sum of as many numbers as are designated.

```
proc sum args {
    set s 0
    foreach i $args {
        incr s $i
    }
    return $s
}
```

Once the procedure above is specified, sum 1 2 3 4 5 returns 15 and sum without any arguments returns 0.

# Arrays

Tcl can organize a set of individual strings as an array and not only as a list. In an array, each *element* (that is, each individual string) can be referred to by a name that appears in parentheses after the array name. The element's name may be numeric, but it does not need to be.

You define an array by simply setting one of its elements. Here is an example in which the array element's *name* is not numeric but its *content* happens to be numeric:

```
set x(fred) 44
```

Assuming that no array named x existed previously, the above command creates the array and gives it one element. The element's name is fred and the element's value is the string 44.

Tcl arrays are one-dimensional, but you can simulate further dimensions by carefully structuring your element names. For example:

```
set y(fred,age) 44
set y(fred,shoesize) 9
set y(fred,eyes) blue
set y(joan,age) 46
set y(joan,shoesize) 7
set y(joan,eyes) brown
```

## array

The array command is a family of commands.

The command:

```
array names name-of-array
```

returns the names (not the values) of the specified array's elements.

The command:

```
array size name-of-array
```

returns the number of elements in the array.

For example:

```
set x(fred) 44
set x(5) [expr $x(fred) + 6]
 set howmany [array size x]
 set calledwhat [array names x]
```

The above script creates an array with two elements, The element x(fred) receives the value 44, and the element x(5) receives the value 50. It sets the variable howmany to the number of elements in the array: 2.

It sets the variable calledwhat to the names of the elements: fred 5.

## Errors

Unless you intervene, any Tcl error (such as an attempt to use a nonexistent command or an unacceptable argument) aborts the entire Tcl script and returns control to the non-Tcl program. You can expect to receive an error message. Here is an example:

```
if {[catch {expr {2+}} msg]} {
set msg {syntax error in expression "2+"}
}
```

## errorInfo

To retrieve more information about the error, you can consult the global variable errorInfo, which receives a fresh stack trace as its value each time an error occurs. The errorInfo variable tells you what the error is, and where it is located in your script. After the above example, set errorInfo (the set command with a single argument, to return the value of that argument) would return this:

```
syntax error in expression "$n + i"
    while executing
"expr {$n + i}"
    invoked from within
"set n [expr {$n + i}]..."
    ("foreach" body line 2)
    ...
```

In the preceding example, the error is a syntax error in the expression "$n + i."

If you know that your script may generate an error but you do not want the error to abort the script, write the error-prone part of the script as an argument for the catch command.

## catch

As its first argument, the catch command takes the script that you want to execute. The second argument, a variable to log the result, is optional.

If the script finishes without incident, then the catch command returns the value 0 and writes the script's result to the second argument (if any).

If an error occurs in the course of the script, then the catch command returns the value 1 and writes the error message to the second argument (if any). Here is an example:

```
if {[catch {expr {2+}} msg]} {
set msg {syntax error in expression "2+"}
}
```

In the above example, the expression {expr {2 +}} is incorrect. The catch command returns the value 1, indicating an error has occurred. The catch command has three more hard-coded return values: 2 if the return command occurred, 3 if the break command occurred, and 4 if the continue command occurred.

You can cause the catch command to return other values of your own in order to signal other incidents.

# Tcl Style Recommendations

The following are style recommendations for Tcl scripts:

- Keep substitutions simple. For complex arguments, use commands like format.
- When you need another level of expansion, use eval.
- To create commands safely, use list commands.
- Lists parse cleanly as commands; each list element becomes one word of the command.

# Index

## F



## G



## H



## I



## J

## T



## U



## V



## W