



WinRunner®
TSL Reference Guide
Version 7.0



MERCURY INTERACTIVE

TSL Reference Guide, Version 7.0

© Copyright 1994 - 2000 by Mercury Interactive Corporation

All rights reserved. All text and figures included in this publication are the exclusive property of Mercury Interactive Corporation, and may not be copied, reproduced, or used in any way without the express permission in writing of Mercury Interactive. Information in this document is subject to change without notice and does not represent a commitment on the part of Mercury Interactive.

Mercury Interactive may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents except as expressly provided in any written license agreement from Mercury Interactive.

WinRunner, XRunner, LoadRunner, TestDirector, TestSuite, WebTest, and Astra are registered trademarks of Mercury Interactive Corporation in the United States and/or other countries. Astra SiteManager, Astra SiteTest, Astra QuickTest, Astra LoadTest, Topaz, RapidTest, QuickTest, Visual Testing, Action Tracker, Link Doctor, Change Viewer, Dynamic Scan, Fast Scan, and Visual Web Display are trademarks of Mercury Interactive Corporation in the United States and/or other countries.

This document also contains registered trademarks, trademarks and service marks that are owned by their respective companies or organizations. Mercury Interactive Corporation disclaims any responsibility for specifying which marks are owned by which companies or organizations.

If you have any comments or suggestions regarding this document, please send them via e-mail to documentation@mercury.co.il.

Mercury Interactive Corporation
1325 Borregas Avenue
Sunnyvale, CA 94089 USA

Table of Contents

Welcome to TSL	v
Using This Guide	v
WinRunner Documentation Set.....	vi
Online Resources	vi
Typographical Conventions.....	viii
Chapter 1: Introduction	1
Function Types	2
Analog Functions.....	3
Context Sensitive Functions	3
Customization Functions	5
Standard Functions.....	5
Chapter 2: Language	7
Variables and Constants	7
Operators and Expressions	12
Statements	17
Control Flow.....	18
Arrays.....	22
Input-Output	28
Comments	29
Built-in Functions.....	29
User-Defined Functions.....	30
External Function Declarations.....	32
Chapter 3: Guidelines for Working with TSL	35
Test Scripts.....	36
Flow Control.....	40
Return Values	41
Path Names.....	42
tl_step Function.....	43
GUI Map	44
Libraries and Functions	45

Chapter 4: Reserved Words	51
Chapter 5: Functions by Category	55
Analog Functions	56
Context Sensitive Functions	58
Customization Functions	97
Standard Functions	100
Chapter 6: Return Values	117
General Return Values.....	118
Return Values for Database Functions	122
Return Values for PowerBuilder and Table Functions	123
Return Values for Terminal Emulator Functions	124
Chapter 7: Alphabetical Reference	125
Index	487

Welcome to TSL

Welcome to TSL, Mercury Interactive's Test Script Language (TSL).

Using This Guide

This book is a comprehensive guide to Mercury Interactive's Test Script Language (TSL). It provides a detailed description of TSL and how to use it in your test scripts. It lists all TSL functions alphabetically and by category, and describes the parameters, return values, and availability for each function. This book assumes that you are already familiar with WinRunner. For information on using WinRunner, see the *WinRunner User's Guide*.

This book contains the following chapters:

Introduction

Provides an overview of TSL and the different types of TSL functions. Read this section to learn which groups of TSL functions are relevant for your product.

Language

Describes the basic elements of the TSL programming language, such as: constants and variables, operators and expressions, statements, control-flow, arrays, input/output.

Guidelines for Working with TSL

Provides guidelines to assist you in creating intuitive and readable test scripts and libraries.

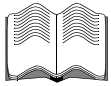
Functions by Category

Provides a list of TSL functions grouped according to the type of tasks they perform. Functions are arranged alphabetically within each category, and a brief description of each function is included.

Alphabetical Reference

Lists all TSL functions alphabetically. The name of each function is listed along with the type and the category to which it belongs. A description and complete syntax are provided. The definition of the function's parameters and its return values and availability are also described.

WinRunner Documentation Set



In addition to this guide, WinRunner comes with a complete set of documentation:

WinRunner Installation Guide describes how to install WinRunner on a single computer or a network.

WinRunner Tutorial teaches you basic WinRunner skills and shows you how to start testing your application.

WinRunner User's Guide explains how to use WinRunner to meet the special testing requirements of your application.

WinRunner Customization Guide explains how to customize WinRunner to meet the special testing requirements of your application.

Online Resources

WinRunner includes the following online resources:

Read Me First provides last-minute news and information about WinRunner.

What's New in WinRunner describes the newest features in the latest versions of WinRunner.

Books Online displays the complete documentation set in PDF format. Online books can be read and printed using Adobe Acrobat Reader 4.0, which is included in the installation package. Check Mercury Interactive's Customer Support web site for updates to WinRunner online books.

WinRunner Context-Sensitive Help provides immediate answers to questions that arise as you work with WinRunner. It describes menu commands and dialog boxes, and shows you how to perform WinRunner tasks. Check Mercury Interactive's Customer Support Web site for updates to WinRunner help files.

TSL Online Reference provides additional information on each function and examples of usage. You can open the *TSL Online Reference* from the WinRunner group in the Start menu or from WinRunner's Help menu. To open the online reference to a specific function, click the context-sensitive Help button and then click a TSL statement in your test script, or place your cursor on a TSL statement in your test script and then press the F1 key. Check Mercury Interactive's Customer Support Web site for updates to the *TSL Online Reference*.

WinRunner Sample Tests includes utilities and sample tests with accompanying explanations. Check Mercury Interactive's Customer Support Web site for updates to WinRunner sample tests.

Technical Support Online uses your default Web browser to open Mercury Interactive's Customer Support Web site.

Support Information presents Mercury Interactive's home page, its Customer Support web site, and a list of Mercury Interactive's offices around the world.

Mercury Interactive on the Web uses your default web browser to open Mercury Interactive's home page. This site provides you with the most up-to-date information on Mercury Interactive, its products and services. This includes new software releases, seminars and trade shows, customer support, training, and more.

Typographical Conventions

This book uses the following typographical conventions:

Bold	Bold text indicates function names and the elements of the functions that are to be typed in literally.
<i>Italics</i>	<i>Italic</i> text indicates variable and parameter names.
Helvetica	The Helvetica font is used for examples and statements that are to be typed in literally.
[]	Square brackets enclose optional parameters.
{ }	Curly brackets indicate that one of the enclosed values must be assigned to the current parameter.
...	In a line of syntax, three dots indicate that more items of the same format may be included. In a program example, three dots are used to indicate lines of a program that have been intentionally omitted.
	A vertical bar indicates that either of the two options separated by the bar should be selected.

1

Introduction

The scripts you create with Mercury Interactive systems are written in Test Script Language (TSL). TSL is an enhanced, C-like programming language designed for testing. At the heart of Mercury Interactive's integrated testing environment, TSL is high-level and easy to use. It combines the power and flexibility of conventional programming languages with functions specifically developed for use with Mercury Interactive's products. This enables you to modify recorded material or to program sophisticated test suites.

This reference manual is intended to help you read, edit, and write TSL scripts. It contains a description of the programming language capabilities of TSL and a list of TSL functions.

This chapter provides overviews about:

- ▶ Function Types
- ▶ Analog Functions
- ▶ Context Sensitive Functions
- ▶ Customization Functions
- ▶ Customization Functions

Function Types

There are four types of TSL functions. Each type of function addresses a different requirement.

Function Type	Requirement
Analog	perform mouse and keyboard input
Context Sensitive	perform operations on GUI objects
Standard	perform basic programming-language operations
Customization	configure the testing tool according to your requirements

The functions that are available depend on which testing product you are using.

WinRunner: If you work with WinRunner, you can use functions from all of the categories. Some functions are supported only when working with applications developed in a specific environment such as PowerBuilder or Visual Basic. To check the availability of a specific function, click the Availability button at the top of the Help screen for that function.

LoadRunner GUI Vusers on PC platforms: This type of GUI Vuser uses WinRunner to create system load. For this reason, you can use functions from any of the categories. You can also use the LoadRunner functions described in the “GUI Vuser Scripts” section of the *LoadRunner Creating Virtual User Scripts User’s Guide for Windows and UNIX Platforms*.

LoadRunner Scenarios: In LoadRunner scenario scripts (UNIX only), you can use standard functions in addition to the LoadRunner functions described in the *LoadRunner Controller User’s Guide*.

Analog Functions

Analog functions record and execute operations at specified screen coordinates. When you record in Analog mode, these functions are used to depict mouse clicks, keyboard input, and the exact coordinates traveled by the mouse. When you run a test, Analog functions retrace the mouse tracks and exactly resubmit the input you recorded. Analog functions also support different test operations such as synchronization, verification, and text manipulation.

Analog functions are available for:

- ▶ WinRunner
- ▶ LoadRunner GUI Vusers on PC Platforms

Coordinate and Numbering Conventions

Many of the Analog functions refer to screen coordinates. In the system of coordinates used by Mercury Interactive's products, the origin (0,0 coordinate) is located in the upper left corner of the screen. The maximum value of x is the width of the screen, in pixels, minus one. The maximum value of y is the height of the screen, in pixels, minus one.

Context Sensitive Functions

Context Sensitive functions depict actions on the application under test in terms of GUI objects (such as windows, lists, and buttons), ignoring the physical location of an object on the screen. In Context Sensitive mode, each time you record an operation on the application under test (AUT), a TSL statement is generated in the test script which describes the object selected and the action performed.

Context Sensitive functions are available for:

- ▶ WinRunner
- ▶ LoadRunner GUI Vusers on PC Platforms

Context Sensitive Object Naming Conventions

Most Context Sensitive functions include parameters which refer to an object's logical name.

Note that you can replace the logical name of the object with the physical description. During recording, the logical name is automatically used by the system. However, the function will also work with the physical description of the object.

For example, the syntax of **button_press** function is:

```
button_press ( button [, mouse_button ] );
```

The *button* parameter may be the logical name of the button—for example:

```
button_press("OK");
```

But it can also be the physical description—for instance:

```
button_press("{class:push_button, label:\\"OK\\"}");
```

Numbering Conventions

Numbering for most Context Sensitive functions starts from 0. For example, the function **list_get_item** returns 0 for the first item of the given list.

Position coordinates for the “edit” Context Sensitive functions, such as **edit_get_info**, are denoted by row and column. The first row is numbered “0.” Columns are denoted by insertion position, not by character index. In other words, the position before the first character in any line is “0”, the position between the first and second characters is “1”, and so on.

Customization Functions

Customization functions allow you to enhance your testing tool so that it better supports your specific needs. For example, you can add functions to the Function Generator, or create custom GUI checkpoints.

Customization functions are available for WinRunner.

Standard Functions

Standard functions include the general elements of a programming language, such as basic input and output, control-flow, mathematical, and array functions. By combining these elements with Analog and Context Sensitive functions, you can transform a simple test into an advanced testing program.

Standard functions are available for all Mercury Interactive products.

2

Language

This chapter describes the basic elements of the TSL programming language, including:

- ▶ Variables and Constants
- ▶ Operators and Expressions
- ▶ Statements
- ▶ Control Flow
- ▶ Arrays
- ▶ Input-Output
- ▶ Comments
- ▶ Built-in Functions
- ▶ User-Defined Functions
- ▶ External Function Declarations

Variables and Constants

Variables and constants may be either strings or numbers. Declaration is optional; if variables are not declared, their type is determined at run time according to their context.

Variable names can include English-language letters (a-z and A-Z), digits, and underscores (_). The first character must be a letter or an underscore. TSL is case-sensitive; *y* and *Y* are therefore two different characters. Note that names of built-in functions and keywords (such as *if*, *while*, *switch*) cannot be used as variable names.

Types of Variables and Constants

TSL supports two types of constants and variables: *numbers* and *strings*. Numbers may be either integer or floating point, and exponential notation is also acceptable. For example, -17, .05, -3e2, and 3E-2 are all legal values.

Strings consist of a sequence of zero or more characters enclosed within double quotes. When a backslash (\) or double-quote (") character appears within a string, it must be preceded by a backslash. Special characters can be incorporated in a string using the appropriate representation:

backspace	\b	vertical tab	\v
carriage return	\r	newline	\n
formfeed	\f	octal number	\ooo
horizontal	\t		

In the case of octal numbers, the zeroes represent the ASCII code of a character. For example, "\126" is equivalent to the letter "v."

For example, to represent the string "The values are: 12 14 16", type:

```
"\The values are:\t12\t14\t16\"
```

At a given moment, the value of a constant or variable can be either a string or a number. The TSL interpreter determines the type according to the operation performed. For example:

```
x = 123;
s = x & "Hello";
y = x + 1;
```

Variable *x* is assigned the value *123*. In the second statement, because the operation is concatenation (&), *x* is treated as a string. The interpreted value of *s* is therefore *123Hello*. In the third line, because the operation is addition, *x* is treated as a number. Variable *y* is therefore assigned the value *124*.

In the case of an expression where a mathematical operation is performed on a string, such as

```
"6RED87" + 0
```

the numeric value of the string is the first part of the string that can be evaluated to a number. Here, the numeric value of the expression is 6.

Since relational operators are valid for both strings and numbers, a numeric comparison is always performed if both operands can be evaluated to a number. For instance, in the relational expression below,

```
"0.01" == "1e-2"
```

although both constants are written like strings (enclosed within quotation marks), both expressions are also valid numbers so a numeric comparison is performed. But in the next expression,

```
"0.01" == "1f-2"
```

the second expression is not a number, so a string comparison is performed.

Undeclared Variables

If a variable is not declared, it is created implicitly when it is assigned or used in an expression. If a variable is not initialized, it is given the string value "" (null) at run time.

All undeclared variables are global, unless they are on the formal Parameter List of a called test. For more information on parameters, see the *WinRunner User's Guide*.

Variable Declarations

Note that while constant and variable declarations are optional in tests, they are required in user-defined functions. Variable declarations have the following syntax:

```
class variable [ = init_expression ];
```

The *init_expression* assigned to a declared variable can be any valid expression. If an *init_expression* is not set, the variable is assigned an empty string. The variable *class* can be any one of the following:

auto: An auto variable can only be declared within a function and is local to that function. It exists only while the function is running. A new copy of the variable is created each time the function is called.

static: A static variable is local to the function, test, or compiled module in which it is declared. The variable retains its value until the test is terminated by a Stop command.

public: A public variable can only be declared within a test or module, and is available for all functions, tests, and compiled modules.

extern: An extern declaration indicates a reference to a public variable declared outside of the current test or module.

With the exception of the auto variable, all variables continue to exist until the Stop command is executed. For example, the statement

```
static a=175, b=get_time( ), c = 2.235;
```

defines three variables (a, b, and c), and assigns each an initial value. This value is retained between invocations of the test. The following script segment demonstrates how a static variable can be used so that a message is printed only the first time that the test, T_2, is called.

```
static first = 1;
  pause ("first = " & first);
  if (first == 1) {
    first = 0;
    report_msg ("Test T_2 was called.");
  }
```

The following table summarizes the scope, lifetime, and location of the variable declarations for each class:

Declaration	Scope	Lifetime	Declare the variable in...
auto	local	end of function	function
static	local	until stop	function, test, or module
public	global	until stop	test or module
extern	global	until stop	function, test, or module

Constant Declarations

The **const** specifier indicates that the declared value cannot be modified. The syntax of this declaration is:

```
[ class ] const name [ = expression ];
```

The *class* of a constant may be either public or static. (If no class is explicitly declared, the constant is assigned the default class public.) Once a constant is defined, it remains in existence until the Stop command is executed.

For example, defining the constant TMP_DIR using the declaration:

```
const TMP_DIR = "/tmp";
```

means that the assigned value /tmp cannot be modified. (This value can be changed only by explicitly making a new constant declaration for TMP_DIR.)

Operators and Expressions

TSL supports six types of operators: arithmetical, concatenation, relational, logical, conditional, and assignment. Operators are used to create expressions by combining basic elements. In TSL, expressions can consist of constants, variables, function calls, and other expressions.

Arithmetical Operators

TSL supports the following arithmetical operators:

+	addition
-	subtraction (unary)
-	subtraction (binary)
*	multiplication
/	division
%	modulus
^ or **	exponent
++	increment (adds 1 to its operand - unary operator)
--	decrement (subtracts 1 from its operand - unary operator)

The result of the modulus operation is assigned the sign of the dividend. For example:

```
7 % -4 = 3
-4.5 % 4 = -0.5
```

The increment and decrement operators may be placed before the variable ($++n$), or after ($n++$). As a result, the variable is incremented either before or after the value is used. For example:

```
i = 5;
j = i++;
k = ++i;
print(i & j & k);
```

prints the values 7, 5, 7. Note that the increment and decrement operators may be applied only to variables, and not to expressions, such as (a + b).

Concatenation Operator

The ampersand (&) character is used to concatenate strings. For example, the statement

```
x = "ab" & "cd";
```

assigns the string value *abcd* to variable *x*.

Relational Operators

The relational operators used in TSL are:

>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
!=	not equal to

Relational expressions are evaluated to the value 1 if true, and 0 if false. When the value of an expression is null or zero, it is considered false. All other values are considered true.

Strings are compared character by character according to their ASCII value. Letter strings are evaluated in terms of alphabetical order; the string which comes first alphabetically is considered smaller. For instance, "galactic" < "galaxy".

Logical Operators

Logical operators are used to create logical expressions by combining two or more basic expressions. TSL supports the following logical operators:

&&	and
	or
!	not (unary)

Logical expressions are assigned the value 1 if true, and 0 if false. When the value of an expression is null or zero, it is considered false. All other values are considered true. Logical expressions are evaluated from left to right, and as soon as the value of an expression is determined, interpretation stops. For example, in the expression

```
(g != 0) && (d/g > 17)
```

if the first expression is false, then the second expression is not evaluated.

Conditional Operator

The conditional operator is the ? (question mark) character. Conditional expressions have the format:

```
expression1 ? expression2 : expression3
```

expression1 is evaluated first; if it is true, *expression2* is evaluated and becomes the value of the expression. If *expression1* is false (zero or null), then *expression3* is executed and becomes the value of the expression. In the following statement,

```
(g != 0) ? 17 : 18;
```

if the first expression is true (*g* is not equal to zero), then the value of the conditional expression is 17. If the first expression is false, then the value of the conditional expression is 18.

For more information, see “Control Flow” on page 18.

Assignment Operators

Assignment operators are used to assign values to variables and arrays. All of the binary arithmetical operators have corresponding assignment operators:

Operator	Example	Meaning
=	$a = b$	assign the value of b to a
+ =	$a += b$	assign the value of a plus b to a
- =	$a -= b$	assign the value of a minus b to a
* =	$a *= b$	assign the value of a times b to a
/ =	$a /= b$	assign the value of a divided by b to a
% =	$a \% = b$	assign the value of a modulo b to a
^= or **=	$a \wedge = b$	assign the value of a to the power of b to a

For example, in the following segment of a test script,

```
for (i=0; i<200; i+=20)
    move_locator_abs(i,i);
```

the value of i is incremented by 20 after each repetition of the loop. The mouse pointer is then moved to the new position defined by i . For more information about for loops see “Control Flow” on page 18.

Precedence and Associativity of Operators

The rules of precedence and associativity determine the order in which operations are performed when more than one operator appears in an expression. Operators with higher precedence are interpreted before operators with lower precedence. For example, multiplication is performed before addition.

When more than one operator of the same level of precedence appears in an expression, the associativity indicates the order in which they are interpreted. For example, in

$$x / 2 + i - q$$

division is performed first. Addition is performed before subtraction because the associativity of these operators, which have the same level of precedence, is left to right.

The following table lists the precedence, in descending order, and the associativity of operators:

Operator (in order of precedence)	Associativity
() (parentheses)	none
++ --	none
^ **	right to left
! - + (unary)	none
* / %	left to right
+ - (binary)	left to right
&	left to right
< <= > >= == !=	none
in (array operator)	none
&&	left to right
	left to right
?	right to left
= += -= *= /= %= ^= **=	right to left

Statements

Any expression followed by a semicolon is a statement. A statement can continue beyond one line.

In a control-flow structure, a single statement can be replaced by a group of statements, or block. Statements are grouped by enclosing them within curly brackets { }. Each individual statement within brackets is followed by a semicolon, but the brackets themselves are not. This is illustrated below:

```
for (i = 0; i < 10; i++) {  
    st = "Iteration number " & i;  
    type (st);  
}
```

Control Flow

TSL control-flow statements include:

- ▶ *if/else* and *switch* for decision-making
- ▶ *while*, *for*, and *do* for looping
- ▶ *break* and *continue* for loop modification

If/Else Statement

TSL provides an *if/else* statement for decision-making. The *else* clause is optional. The syntax of this statement is:

```
if ( expression )  
    statement1  
[ else  
    statement2 ]
```

The *expression* is evaluated; if the value of the *expression* is true (nonzero or non-null), *statement1* is executed; if the value is false (zero or null), and the [else *statement2*] clause is included, *statement2* is executed.

When if statements are nested, the TSL interpreter associates each *else* with the if that appears closest to it. For example, a statement such as:

```
if (b1) if (b2) s1; else s2;
```

is interpreted as follows:

```
if (b1) {  
    if (b2)  
        s1;  
    else  
        s2;  
}
```

Switch Statement

The *switch* statement provides the mechanism for a multi-way decision. The syntax of this structure is:

```
switch ( expression )
{
    case case_expr1:
        statement(s)
    case case_expr2:
        statement(s)
    case case_exprn:
        statement(s)
    [ default: statement(s) ]
}
```

The *switch* statement consecutively evaluates each of the enumerated case expressions (*case_expr1*, *case_expr2*, ..., *case_exprn*), until one is found that equals the initial *expression*. If no case expression is equal to the specified *expression*, then the optional default statements are executed.

Note that the first time a case expression is found to be equal to the specified initial *expression*, no further case expressions are evaluated. However, all subsequent statements enumerated by these cases are executed, unless you use a *break* statement within a case to end the loop. For example:

```
switch (a) {
case"xyz":
    b = a & "tw";
    break;
case"uv":
    pause ("hello");
    x = a;
    break;
default:
    x = a;
}
```

Note that while the initial expression can be any regular expression, case expressions can only be constants or variables.

Looping Statements

TSL provides several statements that enable looping.

```
while ( expression )
    statement
```

While the *expression* is true, the *statement* is repeatedly executed. At the start of each repetition of the loop, the *expression* is evaluated; if it is true (nonzero or non-null), the *statement* is executed, and the *expression* is re-evaluated. The loop ends when the value of the *expression* is false. For example,

```
i = 1;
while (i < 21)
    type (i++);
```

types the value of *i* 20 times.

```
for ( [ expression1 ]; [ expression2 ]; [ expression3 ]; )
    statement
```

First, *expression1* is implemented as the starting condition. While *expression2* is true, the *statement* is executed, and *expression3* is evaluated. The loop repeats until *expression2* is found to be false. This statement is equivalent to:

```
expression1           # state initial condition
while (expression2) { # while this is true
    statement          # perform this statement and
    expression3       # evaluate this expression
}
```

For example, the *for* loop below performs the same function as the *while* loop above.

```
for (i=1; i<21; i++)
    type (i);
```

Note that if *expression2* is missing, it is always considered true, so that

```
for (i=1;i++)
    type (i);
```

is an infinite loop.

```
do
    statement
while ( expression );
```

The *statement* is executed and then the *expression* is evaluated. If the *expression* is true, then the cycle is repeated. This statement differs from the *while* and *for* statements in that the *expression* is evaluated at the end. Therefore, the loop is always executed at least once. For example, in the following statement,

```
i = 20;
do
    type (i++);
while (i < 17);
```

the structure of the loop ensures that the value of *i* is typed at least once.

Loop Modification

The following statements can be used to exit a loop or to jump to the next iteration.

break;

The *break* statement causes an exit from within a loop. If loops are nested, *break* affects the innermost *for*, *while*, or *do* loop that encloses it.

For example, a *for* loop where *expression2* is undefined can be terminated using *break*:

```
for (i = 1;; i++) {
    type (i);
    if (i > 29)
        break;
}
continue;
```

The *continue* statement causes the next cycle of the loop to begin. In a *do/while* loop, execution resumes with the test expression. In a *for* loop, execution resumes with *expression3*. For example:

```
for (i = 1; i<=300; i++) {
    if (i % 3 != 0) {
        continue; # to next number
    }
    ...           # long processing
    type(i & "<kReturn>");
}
```

Here, a certain process should only be performed on every third number. Therefore, if *i* cannot be divided equally by three, execution continues with the next iteration of the loop.

Arrays

TSL supports associative arrays. Arrays in TSL are unique in that:

- ▶ Array declaration and initialization are optional.
- ▶ Each element has a user-defined string subscript.

Rather than arrays of fixed length with numeric subscripts, TSL arrays contain an undefined number of elements, each with a user-defined string subscript.

For example, the statement

```
capitals["Ohio"] = "Columbus";
```

assigns the value "Columbus" to the element with subscript "Ohio" in the array *capitals*. If array elements are not declared, they are created the first time they are mentioned and the order of the elements in the array is not defined. Any uninitialized array element has the numeric value zero and the string value null ("").

Arrays can be used to store both numbers and strings. In the following test script, an array is used to store a series of dates and times:

```
for (i=0; i<5; i++) {
    date = time_str();
    date_array[i] = date;
    wait(5);
}
```

Here, each array element includes the date and time of the call to the **time_str** function. The subscript of the array element is the value of *i*.

Array Declaration

Array declaration is optional within a test but required within user-defined functions (initialization is optional). Using the following syntax, you can define the class and/or the initial expression of an array. Array size need not be defined in TSL.

```
class array_name [ ] [ =init_expression ]
```

The array *class* may be any of the classes listed under Variable Declarations. The *init* expression can take one of two formats: C language syntax, or a string subscript for each element.

An array can be initialized using the C language syntax. For example:

```
public hosts [ ] = {"lithium", "silver", "bronze"};
```

This statement creates an array with the following elements:

```
hosts[0]="lithium"  
hosts[1]="silver"  
hosts[2]="bronze"
```

Note that, as in C, arrays with the class *auto* cannot be initialized.

In addition, an array can be initialized using a string subscript for each element. The string subscript may be any legal TSL expression. Its value is evaluated during interpretation or compilation. For example:

```
static gui_item [ ]={  
    "class"="push_button",  
    "label"="OK",  
    "X_class"="XmPushButtonGadget",  
    "X"=10,  
    "Y"=60  
};
```

creates the following array elements:

```
gui_item ["class"]="push_button"  
gui_item ["label"]="OK"  
gui_item ["X_class"]="XmPushButtonGadget"  
gui_item ["X"]=10  
gui_item ["Y"]=60
```


Array Initialization

Arrays are initialized once during a test run. The TSL interpreter maintains the original initialization values throughout the test run. If you edit an array's initialization values, the new values will not be reflected during test execution. To reset the array with new initialization values, perform one of the following:

- ▶ stop/abort the test run
- ▶ define the array elements explicitly

When you stop the test run, all of the script's variables are destroyed. The next time you execute the script, the array is initialized with the new values.

Alternatively, you can explicitly define an array's elements. When you assign a value to each array element, you ensure that the array is updated with the new values for each test run. In the following example, the regular array initialization is replaced with explicit definitions:

Regular Initialization	Explicit Definitions
<code>public array[] = {1,2,3};</code>	<code>array[0] = 1;</code>
	<code>array[1] = 2;</code>
	<code>array[2] = 3;</code>

Multidimensional Arrays

TSL supports multidimensional arrays such as `a[i,j,k]`. Multidimensional arrays can be used like records or structures in other languages. For example, the following script uses a multidimensional array to store the date and time:

```
for (i = 0; i < 10; i++) {
    date=time_str();
    split(date,array, " ");
    multi_array[i, "day"] = array[1];
    multi_array[i, "time"] = array[4];
    wait(5);
}
```

TSL simulates multidimensional arrays using one-dimensional arrays. The element `multi_array[i1, i2,...in]` is stored in the one-dimensional array called `multi_array`, in the element `[i1 & SUBSEP & i2 & SUBSEP... & in]`. (The variable `SUBSEP` has the initial value “\034,” but this value may be changed.)

Multidimensional arrays can also be declared and initialized, as described above. For example, a multidimensional array could be initialized as follows:

```
static rectangles [ ] = {
    {153, 212, 214, 437},
    {72, 112, 88, 126},
    {351, 312, 399, 356}
}
```

The *in* Operator

The *in* operator is used to determine if a subscript exists in an array.

```
subscript in array;
```

returns the value 1 if the subscript exists, and 0 if it does not. It can be used in a conditional statement, like the one below which checks whether the element with the subscript *new* exists in the array *menu_array*:

```
if ("new" in menu_array)
```

The operator *in* should be used rather than the following statement:

```
if (menu_array["new"] != "")...
```

because this statement causes the element to be created, if it does not already exist. (Recall that array elements are created the first time they are mentioned.)

The *in* operator can also be used for multidimensional arrays. The subscript of the element is enclosed in parentheses, as in the following statement:

```
if (("new.doc", 12) in multi_array)...
for ( element in array ) statement
```

causes the *element* to be set to the subscript of each element in the *array*. The statement is executed once for each element of the array, and the loop is terminated when all elements have been considered. The order in which the subscripts are read is undefined. The sample script below reads an array for which each element is a date and time string. A *for* loop is used to print to the screen each of the elements of the array.

```
for (i in date_array)
    print ("the date was " & date_array[i]);
```

Specifying a Starting Subscript

TSL allows you to assign values to array elements starting from a specific subscript number. You specify the starting subscript in the array initialization. Remember that the array subscripts are zero-based—the first subscript number is 0.

```
abc[ ] = {starting subscript = value1, value2, value3... }
```

For example, if the array size is ten, you can assign values to the last five elements of the array:

```
public abc[ ] = {5 = 100,101,102,103,104}
```

As a result, the abc array receives the following values:

```
abc[5]=100
abc[6]=101
abc[7]=102
abc[8]=103
abc[9]=104
```

Array Functions

TSL provides two array functions: **delete** and **split**. The **delete** function removes an element from an array; **split** splits a string into fields and stores the fields in an array. Note that since TSL arrays are associative, deleting one element does not affect any other element. For instance, if you delete the element `a[2]` from an array with three elements, `a[1]` and `a[3]` will not be affected. For details, see the alphabetical reference.

Input-Output

TSL provides a number of built-in functions that allow you to read and write to files or to the screen.

For UNIX products, the **sprintf** function returns a formatted string to a variable.

For WinRunner and other PC products, use the **file_open** function to open a file for reading and writing. The **file_printf** function writes to a file, and **file_getline** reads from a file. The **file_close** function closes a file that you opened with **file_open**.

There are two functions that generate output within the testing environment. The **report_msg** function prints a user-defined string expression to the test run report. The **pause** function stops the test run and displays a string expression in a message box on the screen.

For more information on any of the TSL built-in functions, refer to the *TSL Online Reference*.

Comments

A number sign (#) indicates that all text from this point to the end of the line is a comment. Comments can appear within statements that extend beyond one line, or can stand alone on a line of test script. The TSL interpreter does not process comments. For example,

```
# Type the date
i=1
while (i<=31)# number of days in month
    type ("The date is January " & i++ & ", 1994");
```

Note that a number sign (#) that appears within a string constant is not considered a comment; for instance, `a="#3"`.

Built-in Functions

TSL provides numerous built-in functions that perform a range of tasks. To call a built-in function from within a test script, use the following syntax:

```
function ( [ parameters ] );
```

Most built-in functions return a value. This value can be assigned to a variable. For example,

```
x = int(12.42);
```

The **int** function returns the integer portion of a positive, real number. Here, `x` is equal to 12.

The return value of a built-in function can also become part of an expression. When a function returns the value 0, the value of the expression is considered false. When it returns any other value, it is considered true. For example,

```
while (getline address < "clients.doc")
    type (address "<kReturn>");
```

The **getline** function returns the value 1 if it succeeds, and 0 at the end of the file. Therefore, the *while* loop above continues until the end of the file is reached (the function returns the value 0).

For detailed information on each of the TSL functions, refer to the *TSL Online Reference*.

User-Defined Functions

In addition to the built-in functions it offers, TSL allows you to design and implement your own functions in test scripts. A user-defined function has the following structure:

```
[class] function name ( [mode] parameter... )
{
  declarations;
  statements;
}
```

Class

The class of a function may be either public or static. If no class is explicitly declared, the function is assigned the default class public. A public function is available to all tests; a static function is available only to the test or compiled module within which the function was defined.

Parameters

Function parameters can be of mode *in*, *out*, or *inout*. For all non-array parameters, the default mode is *in*. The significance of each parameter type is as follows:

in: A parameter which is assigned a value from outside the function.

out: A parameter which passes a value from inside the function.

inout: A parameter which can be assigned a value from outside the function as well as pass on a value to the outside.

A parameter designated as *out* or *inout* must be a variable name, not an expression. Only a variable can be assigned a value in a function call, not an expression. For example, consider a function defined in the following manner:

```
function my_func (out p) {... }
```

Proper usage of the function call is: `my_func (var_1)`; Illegal usage of the function call is: `my_func (arr[i])`; `my_func (a+b)`; because `arr[i]` and `a+b` are expressions.

Array parameters are designated by square brackets. For example, the following parameter list indicates that parameter *a* is an array:

```
function my_func (a[], b, c){
...
}
```

Array parameters can be either *out* or *inout*. If no class is specified, the default *inout* is assumed.

While variables used within a function must be explicitly declared, this is not the case for parameters.

Declarations

Variables used by a function must be declared. The declaration for such a variable can be within the function itself, or anywhere else within the test or module. For syntax, see “Variable Declarations” on page 9 in this chapter.

Return Statement

Any valid statement used within a TSL test script can be used within a function. In addition, the *return* statement is used exclusively in functions.

```
return [ expression ];
```

This statement halts execution of the called function and passes control back to the calling function or test. It also returns the value of the evaluated expression to the calling function or test. (If no expression is attached to the return statement, an empty string is returned.) For additional information on functions, refer to the *TSL Online Reference*.

External Function Declarations

The extern function declaration is used to declare functions that are not part of TSL, but reside in external C libraries. For more information on using C functions stored in external dlls, refer to your *User's Guide*.

The extern declaration must appear before the function can be called. The syntax of the extern function declaration is:

```
extern type function_name ( param1, param2,...);
```

The *type* refers to the return value of the function. Type can be one of the following:

- ▶ *char* (signed and unsigned)*float*
- ▶ *short* (signed and unsigned)*double*
- ▶ *int* (signed and unsigned)*string* (equivalent to C char*)
- ▶ *long* (signed and unsigned)

Each parameter must include the following information:

[*mode*] *type* [*name*] [< *size* >]

<i>mode</i>	The <i>mode</i> can be <i>in</i> , <i>out</i> , or <i>inout</i> . The default is <i>in</i> . Note that these values must appear in lower case.
<i>type</i>	The <i>type</i> can be any of the values listed above.
<i>name</i>	An optional <i>name</i> can be assigned to the parameter to improve readability.
<i>size</i>	This information is required only for an <i>out</i> or <i>inout</i> parameter of type <i>string</i> . (See below.)

For example, to declare a function named `set_clock` that sets the time in a clock application, you write the following:

```
extern int set_clock ( string name, int time );
```

The `set_clock` function accepts two parameters. Since they are both input parameters, no mode is specified. The first parameter, a string, is the name of the clock window. The second parameter specifies the time to be set on the clock. The function returns an integer that indicates whether the operation was successful.

Once the extern declaration is interpreted, you can call the `set_clock` function the same way you call a TSL built-in function:

```
result = set_clock ( "clock v. 3.0", 3 );
```

If an extern declaration includes an *out* or *inout* parameter of type *string*, you must budget the maximum possible string size by specifying an integer *size* after the parameter *type* or (optional) *name*. For example, the statement below declares the function `get_clock_string`. It returns the time displayed in a clock application as a string value in the format “The time is...”

```
extern int get_clock_string ( string clock, out string time <20> );
```

The *size* should be large enough to avoid an overflow. If no value is specified for *size*, the default is 127. There is no maximum size.

TSL identifies the function in your C code by its name only. You must pass the correct parameter information from TSL to the C function. TSL does not check parameters: if the information is incorrect, the operation fails.

In addition, your C function must adhere to the following conventions:

- ▶ Any parameter designated as a *string* in TSL must be associated with a parameter of type *char** in C.
- ▶ Any parameter of mode *out* or *inout* in TSL must be associated with a pointer in C. For instance, a parameter *out int* in TSL must be associated with a parameter *int** in the C function.
- ▶ For WinRunner the external function must observe the standard Pascal calling convention *export far Pascal*.

For example, the following declaration in TSL:

```
extern int set_clock (string name, inout int time);
```

must appear as follows in C:

```
int _far _pascal _export [_loads] set_clock (  
    char* name,  
    int* time  
)
```

3

Guidelines for Working with TSL

This chapter provides guidelines to assist you in creating intuitive and readable test scripts and libraries. There are several advantages to using these guidelines:

- ▶ Uniformity - Shorter learning curve for new test engineers.
- ▶ Clarity - Scripts and functions are easier to read, maintain, and debug.
- ▶ Customer Support - Mercury CSO engineers can easily understand scripts, which results in faster support.

The following guidelines are offered as suggestions. There is an infinite number of styles for creating a test. If you are partial to another style, use the style with which you are most comfortable.

This chapter provides guidelines for working with TSL in the following areas:

- ▶ Test Scripts
- ▶ Flow Control
- ▶ Return Values
- ▶ Path Names
- ▶ tl_step Function
- ▶ GUI Map
- ▶ Libraries and Functions

Test Scripts

Test Header

The test header is inserted at the top of the test script, enclosed with the # symbol. It contains necessary information about the test:

- ▶ Test Name
- ▶ Subject
- ▶ Test Creator
- ▶ Date of creation/Date of revision
- ▶ Purpose of the test
- ▶ Vital information (for example, initial conditions, variable information, state of AUT, and so on.)

The following is an example of a test header:

```
#####  
# TEST NAME: Open Order  
# DATE: 12/12/95  
# DATE OF LAST REVISION: 2/14/96  
# CREATED BY: John Smith  
#  
# PURPOSE: Verify that the correct orders are retrieved from the database  
# INITIAL STATE: Application's main window is open, all others are closed.  
# PARAMETERS: None  
#####
```

Constant Declaration

Constants (const) should be defined at the top of the test. When defining a constant in a particular test, the syntax is as follows:

```
static const <CONST_NAME> = <const_value>;
```

Constant name should be in capital letters and underscores; spaces are not allowed. For example:

```
static const NUMBER_OF_FILES = 3;
static const PATH_OF_FILES = "C:\\TESTS\\FILES";
```

You should not define a constant as public in a test, since a constant defined in one test might subsequently be used in another test as a different value. A constant declared as public should be defined in a library or an initialization test, where it can be used by all tests within a testing session or batch run.

Variable Declaration

Variables used in a test should be declared below the constant declarations and test header. Because TSL is an interpretive language, variables are automatically defined when they are assigned. Therefore, variable declaration should be used for the purpose of holding information that the tester might have to change in order to ensure a successful test run.

When defining a variable, the syntax is as follows:

```
[static/public] <variable_name> = [<variable_value>];
```

Variable names can include letters, underscores, and digits. For example:

```
public my_first_variable = 7;
public MyFirstVariable;
static myFirstVariable = "Hello World!";
```

You should not mix underscores and upper case letters.

There are two ways to initialize a variable:

- [static/public] x = 1;
- [static/public] x;
x = 1;

Functionally, the two choices are the same. The difference is that the variable *x* cannot be reinitialized by the technique in example 1 (all on one line). To ensure that a variable can be reinitialized, use the technique in example 2. For example:

Test A:

```
public x = 1;  
x = 5 + y;
```

...

Test B:

```
call A();  
call A();
```

When you run test B, the second call to test A will not reinitialize x. Use the technique in example 2.

Note the way that the test initializes variables. In a batch run, separate tests might have the same variable names. It is important to ensure that they are reinitialized for each test; otherwise a test might not replay correctly.

Array Declaration

Array declarations should occur with variable declarations. Because TSL is an interpretive language, array declaration is optional. Arrays should be declared when they store information that the tester might change from one test run to another.

When declaring a standard array whose indices are: 0,1,2...,n; the syntax is as follows:

```
[static/public] <array_name> [0]= <value_0>;  
<array_name>[1] = <value_1>;  
...  
<array_name>[n] = <value_n>;
```

For example:

```
public capital[0] = "Sacramento";  
capital[1] = "Austin";  
capital[2] = "Albany";
```

Declaring associative arrays follows the same syntax:

```
[static/public] <array_name>["string_1"] = <value_1>;
...
<array_name>["string_n"] = <value_2>;
```

For example:

```
public capital["California"] = "Sacramento";
capital["Texas"] = "Austin";
capital["New York"] = "Albany";
```

User-Defined Functions

User-defined functions should be defined after the variable declarations. Functions should be declared as static. They can be accessed only by the test in which they reside. Functions declared as public should be placed in a function library. For further information, see “Libraries and Functions” on page 45.

Comments

Comments are essential for clear and intuitive test scripts. A number sign (#) indicates that the text from this point to the end of the line is a comment. Comments can appear within statements that extend beyond one line, or they can stand alone on a line of a test script. They should always begin in the same column as the lines of the script on which they are commenting. When you run a test, the TSL interpreter does not process comments. For example:

```
# This is a comment
set_window ("Window_Name");
button_press ("Button_Name");

# This is also a comment. Checks if window exists
if (win_exists ("Window_Name") == E_OK) {
activate_window ("WinName");
```

Flow Control

Flow control statements should be indented one tab length for easier readability.

If / Else

TSL provides an *if/else* statement for decision-making. The *else* clause is optional. The syntax is as follows:

```
if (<condition>) {  
    statement_1;  
    ...  
    statement_n;  
}  
else {  
    statement_1;  
    ...  
    statement_n;  
}
```

For Loops

For loop syntax is as follows:

```
for (<initial condition>; <end condition>, <index increment/decrement>) {  
    statement_1;  
    statement_n;  
}
```

While Loops

While loop syntax is as follows:

```
while (<condition>) {  
    statement_1;  
    ...  
    statement_n;  
}
```


Do Loops

Do loop is executed at least once. Syntax is as follows:

```
do {
    statement_1;
    ...
    statement_n;
}
while (<condition>)
```

Return Values

Error Codes

Every TSL statement generates a *return value*. Statements within a test script can be checked for specific error codes to indicate whether the statements were executed successfully. You can branch your test according to the return value.

When checking return values, you should use the name instead of the numeric value.

The following bits of script all have the same functionality:

- a) `if (win_exists ("Window_Name") == 0) {
 set_window ("Window_Name");
 ...
}`
- b) `if (!win_exists ("Window_Name")) {
 set_window ("Window_Name");
 ...
}`
- c) `if (win_exists ("Window_Name") == E_OK) {
 set_window ("Window_Name");
 ...
}`

The `win_exists()` statement returns the value 0 when executed successfully. For readability purposes, example `c` is recommended. The return value checked is the constant `E_OK`, whose value is equal to 0.

There is a complete list of generated return values in Chapter 6, “Return Values.” In addition, TSL enables you to create your own error codes. Use the following conventions:

- ▶ Error codes should be in capital letters.
- ▶ Error codes should begin with the letter “E” followed by an underscore (for example, `E_MY_ERROR`).
- ▶ Error code numbers should include a dash “-” followed by a five digit value (for example, `-31001`).
- ▶ Error codes should be defined as public in a library or initialization test (for example, `public const E_MY_ERROR = -31001`).

Return Codes

The variable `rc` is used for checking return codes from a TSL statement. For example:

```
rc = activate_window ("Window Name");  
if (rc!= E_OK)  
report_message ("Could not activate Window Name");
```

The above example verifies that the `activate_window()` function is successful by checking the return code. The return value is `E_OK`.

Path Names

The rule regarding path names is simple: do not use absolute (hardcoded) path names. Because pathnames are so dynamic, you should always to use variables that hold the name of the path in a test script. For example, the line:

```
GUI_load ("c:\\files\\my_file.gui");
```

should be replaced with:

```
path = "c:\\files\\";
GUI_load (path & "my_file.gui");
```

In the case where path names are not parameters, substituting a variable involves a bit more work. For example:

```
call "c:\\tests\\my_test" ();
```

contains a path name that is not a parameter. To replace a hardcoded path name with variables, an *eval* statement must be used. For example:

```
pathname = "\\c:\\\\tmp\\";
eval ("call " & pathname & "my_test" ());
```

tl_step Function

The **tl_step** is an extremely useful function for two reasons:

- ▶ It enables you to enhance a test report by naming a step, giving it a *pass* or *fail* status. It provides additional information as to why a step passed or failed.
- ▶ It can give the entire test a *fail* status without the use of *check_gui* or *check_window*.

You should use the **tl_step** function after every verification point in a test script. In addition, a test that contains a **tl_step** can be imported into the TestDirector test set immediately.

The recommended construction of a **tl_step** statement is as follows:

```
rc = check_gui (5, "Open Order", "list1.ckl", "gui_1");
#verification point
if (rc != E_OK) {
    tl_step ("Init state", 1, "Initial state of Open Order window was incorrect");
}
else {
    tl_step ("Init state", 0, "Initial state of Open Order window was correct");
}
```

In the above example, the **tl_step** statement is used twice: once for failure, and once for success. You should use this construction for readable and informative test reports.

GUI Map

A script generated by WinRunner in Context Sensitive mode is relatively intuitive. However, you can make the test even more intuitive using your GUI map.

You can modify the logical names for objects, as they appear in a test script, for further clarity. For instance, when recording a script in WinRunner, a statement such as the following might be generated:

```
button_press ("ThunderSSCommand_0");
```

You can modify the statement as follows:

```
button_press ("NewOrder");
```

Now you can see what button was pushed after that statement was executed. This new logical name is much more readable and intuitive. To ensure that a readable and logical name is recorded in your script, remember to create the GUI map before recording. Modify logical names as you proceed, wherever necessary.

Note that creating and editing the GUI map before any script has been created will save you having to modify an existing script.

Libraries and Functions

A library is a test consisting of constant declarations and user-defined function declarations. Once the test is completed, it is converted into a module where it can be compiled and loaded into memory, allowing all tests public access to the declarations and functions inside.

Library Header

The format for the header is much like the header for a test script. It is enclosed by the "#" symbol and contains the following information:

- Library Name
- List of functions

For example:

```
#####
# COMPILED MODULE: flt_lib
#
# FUNCTION:
# -STATIC
# - get_flight_from_table()
# - set_table_fields ()
# - set_working_dir ()
#
# -PUBLIC
# - open_order ()
# - delete_order ()
# - insert_order ()
#####
```

Constants

Constants declarations should follow the Library Header. Constants should always be declared as public when defined in a library. For example:

public const <CONST_NAME> = <const_value>;

Constants declared as public can be used by any test.

Function Header

The function header is placed above a user defined function. Like the Test Header, the function header is enclosed by the "#" symbol and stores information about the function:

- Function Name
- Description or purpose of the function
- Input parameters
- Output parameters
- Return Values

For example:

```
#####
FUNCTION: get_flight_from_table
# PURPOSE: This function is static only to this file. It selects a flight from the
# flight table using a given flight number. Also uses split() function to access the
# flight number from the table.
#
# INPUT PARAMS: flight_num  The flight number to be selected.
# OUTPUT PARAMS: none
# RETURN VALUES: Standard return values.
#####

static function get_flight_from_table (in flight_num){...
```

User-Defined Functions

The user-defined function follows immediately after the function header. When declaring a function, the function starts with the function heading. The function heading has the following format:

```
[class] function <function_name> ([mode] <parameter_list>)
```

A function can be one of two classes:

- *Static* - Available only to the current module; not accessible outside the module. A function should be declared as static if it is used only by other functions *within* the library.

- *Public* (default) - Available to all tests and functions *outside* the library. Most functions in a library are declared as public.

The class of the function is followed by the reserved word *function* followed by the function name.

The name of the function should be intuitively meaningful, such as "insert_order". The first character of a function name can be a letter or an underscore.

A parameter can be one of three modes:

- *In* (default) - Assigned a value from *outside* the function
- *Out* - Assigned a value from *inside* the function
- *Inout* - Can be assigned a value from outside the function *and* pass a value to the outside.

Array parameters are designated by square brackets and can be declared only as out or inout (the default).

The function body follows the function heading as follows:

```
[class] function <function_name> ([mode] <parameter_list>
  {
  declarations;
  statement_1;
  statement_n;
  }
```

The function body is enclosed by curly brackets. The open curly bracket ({) is aligned with the first column of the heading. The close curly bracket (}) is aligned in the same column as the open curly bracket.

In test scripts, variable declaration is optional (see "Variable Declaration" on page 37). In functions, however, variables, constants, and arrays all must be declared. A variable can be one of two types:

- *Static* - Limited in scope to the function, test, or module within which it is running.

- ▶ *Auto* (default) - Short for "automatic" (a C language convention). When in doubt, declare the variable as *auto*. Once a variable is declared as *auto*, it is local in scope and exists only for the duration of the function's execution.

For example:

```
public function issue_report_line (in line_to_print)
{
static internal_line_count;
auto tmp_line;
tmp_line = internal_line_count & ":" line_to_print;
report_msg (line_to_print);
internal_line_count++;
}
```

Note that the variable *internal_line_count* retains its value even after control is passed from the function body. It holds the value representing the number of lines reported throughout the test run. It will retain its value as long as the function remains in memory. However, the value of *tmp_line* will be redefined every time *issue_report_line* is called, losing its value from the last call.

The statements in a user-defined function follow the declarations in the function body. A statement can be any valid TSL statement. Statements should be indented one tab length for better readability.

All functions should return a standard return value such as `E_OK` or `E_GENERAL_ERROR`. To return error codes, use the `return()` statement. It returns a value and passes control back to the calling test or function. For example:

```
public function open_order (in OrderNum)
{
set_window ("Open Order");
button_set ("Order Num:", ON);
edit_set ("Order Num:", OrderNum);
button_press ("OK");
if (win_exists ("Flight Reservation System") == E_OK)
{
```



```
set_window ("Flight Reservation System");
button_press ("OK");
return (E_COULD_NOT_OPEN);
    }
# end if
else
return (E_OK);
# Function executed successfully
    }
```

Note that the function *open_order* returns `E_COULD_NOT_OPEN` when the order does not exist and `E_OK` when the function is executed successfully. A function should return an error code, rather than the error code's value.

4

Reserved Words

WinRunner contains reserved words. In addition to the words listed below, all TSL functions and statements are reserved words in WinRunner.

Note that you can change the color and appearance of reserved words in WinRunner's script editor. For more information, refer to the "Customizing the Test Script Editor" chapter in the *WinRunner User's Guide*.

auto	button_check_enabled
button_get_value	case
char	check_file
check_wid	const
continue	default
display_date_result	display_euro_result
double	edit_check_content
edit_check_format	else
endif	exception_on_print
exit	extern
float	function
get_lang	get_obj_record_method
get_runner_str	getline
grab	gsub
GUI_buf_get_data	GUI_buf_get_data_attr

GUI_buf_set_data_attr	GUI_data_get_attr
GUI_data_set_attr	GUI_list_data_attrs
GUI_mark	GUI_point_to
GUI_replay_wizard	if
in	inout
input_to_description_int	list_check_multi_selection
list_check_row_num	list_check_selection
list_get_items_count	list_get_multi_selected
long	menu_get_items_count
menu_verify	move_mouse_abs
move_mouse_rel	move_window
next	obj_check_attr
obj_check_enabled	obj_check_focused
obj_check_label	obj_check_pos
obj_check_size	obj_check_style
obj_set_focus	obj_verify
out	pause_test
printf	process_return_value
prvars	public
quad_click	report_event
report_param_msg	reset_filter
reset_internals	return
save_report_info	scroll_get_value
set_filter	set_obj_record_method
short	signed

static	string
sub	tab_get_page
tab_get_selected_page	tab_select_page
tbl_get_cell_coords	tbl_synchronize
tech	tl_get_status
tl_set_status	tl_setvar
toolbar_get_info	toolbar_wait_info
treturn	trpl_click
tsl_set_module_mark	tsl_test_is_module
ungrab	unsigned
vendor	vuser_status_message
wait_stable_window	win_check_attr
win_check_label	win_check_pos
win_check_size	win_press_cancel
win_press_ok	win_press_return
win_set_focus	win_verify

5

Functions by Category

This section lists all TSL functions according to the type of tasks they perform. Functions are arranged alphabetically within each category, and a very brief description of each function is included. Where appropriate, functions appear in more than one category.

There are four types of functions:

- ▶ Analog Functions
- ▶ Context Sensitive Functions
- ▶ Customization Functions
- ▶ Standard Functions

Analog Functions

Analog functions record and run operations at specified screen coordinates. When you record in Analog mode, these functions are used to depict mouse clicks, keyboard input, and the exact coordinates traveled by the mouse. When you run a test, Analog functions retrace the mouse tracks and exactly resubmit the input you recorded. Analog functions also support test operations such as synchronization, verification, and text manipulation.

Analog functions are divided into the following categories:

- ▶ Bitmap Checkpoint Function
- ▶ Input Device Functions
- ▶ Synchronization Function
- ▶ Table Functions
- ▶ Text Checkpoint Functions

Bitmap Checkpoint Function

Function	Description	See Page
<code>check_window</code>	compares a bitmap of an AUT window to an expected bitmap	149

Input Device Functions

Function	Description	See Page
<code>click</code>	clicks a mouse button	151
<code>click_on_text</code>	clicks a mouse button on a string	151
<code>dbl_click</code>	double-clicks a mouse button	165
<code>get_x</code>	returns the current x-coordinate of the mouse pointer	229
<code>get_y</code>	returns the current y-coordinate of the mouse pointer	229
<code>move_locator_abs</code>	moves the mouse to a new absolute position	282
<code>move_locator_rel</code>	moves the mouse to a new relative position	282
<code>move_locator_text</code>	moves the mouse to a string	283
<code>move_locator_track</code>	moves the mouse along a prerecorded track	283
<code>mtype</code>	clicks one or more mouse buttons	284
<code>type</code>	specifies keyboard input	427

Synchronization Function

Function	Description	See Page
<code>wait_window</code>	waits for a window bitmap to appear in order to synchronize test execution	432

Table Functions

Function	Description	See Page
<code>tbl_click_cell</code>	clicks in a cell in a JFC JTable object	357
<code>tbl_dbl_click_cell</code>	double-clicks in a cell in a JFC JTable object	358
<code>tbl_drag</code>	drags a cell to a different location within a JFC JTable object	362

Text Checkpoint Functions

Function	Description	See Page
<code>click_on_text</code>	clicks on a string	151
<code>find_text</code>	searches for a string	220
<code>get_text</code>	reads text from the screen	228
<code>move_locator_text</code>	moves the mouse to a string	283

Context Sensitive Functions

Context Sensitive functions depict actions on the application under test in terms of GUI objects, ignoring the physical location of an object on the screen. When you record in Context Sensitive mode, a TSL statement, which describes the object selected and the action performed, is generated in the test script.

Context Sensitive functions are divided into the following categories:

- ▶ ActiveBar Functions
- ▶ ActiveX/Visual Basic Functions
- ▶ Bitmap Checkpoint Functions
- ▶ Button Object Functions
- ▶ Calendar Functions
- ▶ Database Functions
- ▶ Data-Driven Test Functions
- ▶ Delphi Functions
- ▶ Edit Object Functions
- ▶ EURO Functions
- ▶ GUI Checkpoint Functions
- ▶ GUI Map Configuration Functions

- ▶ GUI Map Editor Functions
- ▶ Icon Object Functions
- ▶ Java Functions
- ▶ List Object Functions
- ▶ Menu Object Functions
- ▶ Object Functions
- ▶ Oracle Functions
- ▶ PowerBuilder Functions
- ▶ Scroll Object Functions
- ▶ Siebel Functions
- ▶ Spin Object Functions
- ▶ Static Text Object Functions
- ▶ Statusbar Functions
- ▶ Synchronization Functions
- ▶ Tab Object Functions
- ▶ Table Functions
- ▶ Terminal Emulator Functions
- ▶ Text Checkpoint Functions
- ▶ Toolbar Object Functions
- ▶ WAP Functions
- ▶ Web Functions
- ▶ Table Functions for WebTest
- ▶ Window Object Functions
- ▶ Year 2000 Functions

ActiveBar Functions

Function	Description	See Page
ActiveBar_combo_select_item	Selects an item in a ComboBox tool.	126
ActiveBar_dump	Stores information about ActiveBar bands and tools. This information includes captions, names, types and IDs.	127
ActiveBar_select_menu	Selects a menu item in a toolbar.	128
ActiveBar_select_tool	Selects a tool in the toolbar.	129

ActiveX/Visual Basic Functions

The following functions are available only when the ActiveX or the Visual Basic Add-in is installed and loaded.

Function	Description	See Page
ActiveX_activate_method	invokes an ActiveX method of an ActiveX control	130
ActiveX_get_info	returns the value of an ActiveX/Visual Basic control property	131
ActiveX_set_info	sets the value of a property in an ActiveX/Visual Basic control	132
vb_get_label_names	retrieves the names of all label controls in the given form window. The names are stored as subscripts of an array	431

Bitmap Checkpoint Functions

Function	Description	See Page
<code>obj_check_bitmap</code>	compares a current object bitmap to an expected bitmap	285
<code>win_check_bitmap</code>	compares a current window bitmap to an expected bitmap	454

Button Object Functions

Function	Description	See Page
<code>button_check_info</code>	checks the value of a button property	137
<code>button_check_state</code>	checks the state of a radio or check button	138
<code>button_get_info</code>	returns the value of a button property	138
<code>button_get_state</code>	returns the state of a radio or check button	139
<code>button_press</code>	clicks a push button	139
<code>button_set</code>	sets the state of a radio or check button	140
<code>button_wait_info</code>	waits for the value of a button property	140

Calendar Functions

The following functions are available for calendars included in Visual Studio Version 6 and higher and in Internet Explorer Active Desktop Version 4 and higher.

Function	Description	See Page
<code>calendar_activate_date</code>	double clicks the specified date in the calendar	141
<code>calendar_get_selected</code>	retrieves and counts the selected dates in a calendar	142
<code>calendar_get_status</code>	returns the status validity of the date	142
<code>calendar_get_valid_range</code>	returns the date range	143
<code>calendar_select_date</code>	clicks the specified date in a calendar	144
<code>calendar_select_range</code>	clicks the specified date in a calendar	144
<code>calendar_select_time</code>	selects a time in the HH:MM:SS format	145
<code>calendar_set_status</code>	sets the selection status to valid or invalid	145

Database Functions

Function	Description	See Page
db_check	compares current database data to expected database data	158
db_connect	creates a new database session and establishes a connection to an ODBC database	159
db_disconnect	disconnects from the database and ends the database session	160
db_execute_query	executes the query based on the SQL statement and creates a record set	161
db_get_field_value	returns the value of a single field in the database	161
db_get_headers	returns the number of column headers in a query and the content of the column headers, concatenated and delimited by tabs	162
db_get_last_error	returns the last error message of the last ODBC or Data Junction operation	162
db_get_row	returns the content of the row, concatenated and delimited by tabs	163
db_record_check	compares information that appears in the application under test during a test run with the current values in the corresponding record(s) in your database	164
db_write_records	writes the record set into a text file delimited by tabs	165

Database Function for Working with Data Junction

Function	Description	See Page
db_dj_convert	runs a Data Junction export file (.djs file)	160

Data-Driven Test Functions

Function	Description	See Page
ddt_close	closes a data table file	166
ddt_export	exports the information of one table file into a different table file	167
ddt_get_current_row	retrieves the active row in a data table	168
ddt_get_parameters	returns a list of all the parameters in a data table	168
ddt_get_row_count	retrieves the number of rows in a data table	169
ddt_is_parameter	returns whether a parameter in a data table is valid	170
ddt_next_row	changes the active row in a data table to the next row	170
ddt_open	creates or opens a data table file so that WinRunner can access it	171
ddt_report_row	reports the active row in a data table to the test results	171
ddt_save	saves the information in a data table	172
ddt_set_row	sets the active row in a data table	172
ddt_set_val	sets a value in the current row of the data table	173
ddt_set_val_by_row	sets a value in the specified row of the data table	174
ddt_show	shows or hides the table editor of a specified data table	175
ddt_update_from_db	imports data from a database into a data table	175

Function	Description	See Page
ddt_val	returns the value of a parameter in the active row in a data table	176
ddt_val_by_row	returns the value of a parameter in the specified row in a data table	177

Delphi Functions

The following functions are available only when WinRunner support for Delphi is installed and loaded.

Function	Description	See Page
add_dlp_obj	adds a Delphi object	135
dlph_edit_set	replaces the entire content of a Delphi edit object	182
dlph_list_select_item	selects a Delphi list item	182
dlph_obj_get_info	retrieves the value of a Delphi object	183
dlph_obj_set_info	sets the value of a Delphi object	183
dlph_panel_button_press	clicks a button within a Delphi panel	184

Edit Object Functions

Function	Description	See Page
edit_check_info	checks the value of an edit object property	186
edit_check_selection	checks that a string is selected	187
edit_check_text	checks the contents of an edit object	187
edit_delete	deletes the contents of an edit object	188
edit_delete_block	deletes a text block from an edit object	188
edit_get_block	returns a block of text from an edit object	189

Function	Description	See Page
edit_get_info	returns the value of an edit object property	190
edit_get_row_length	returns the length of a row in an edit object	190
edit_get_rows_count	returns the number of rows written in an edit object	191
edit_get_selection	returns the selected string in an edit object	191
edit_get_selection_pos	returns the position at which the selected block starts and ends	192
edit_get_text	returns the text in an edit object	193
edit_insert	inserts text in an edit object	193
edit_insert_block	inserts text in a multi-line edit object	194
edit_replace	replaces part of the contents of an edit object	194
edit_replace_block	replaces a block of text in a multi-line edit object	195
edit_set	replaces the entire contents of an edit object	195
edit_set_insert_pos	places the cursor at the specified point in an edit object	196
edit_set_selection	selects text in an edit object	197
edit_type	types a string in an edit object	197
edit_wait_info	waits for the value of an edit object property	198

EURO Functions

The following functions are available for WinRunner EURO users only:

Function	Description	See Page
EURO_check_currency	captures and compares the currencies in a window	199
EURO_compare_columns	compares two currency columns (dual display) and returns the number of mismatches	200
EURO_compare_fields	compares two fields while converting	201
EURO_compare_numbers	compares two numbers while converting	202
EURO_convert_currency	returns the converted currency value between two currencies	204
EURO_override_field	overrides the original currency in a field to a new currency	205
EURO_set_auto_currency_verify	activates/deactivates automatic EURO verification	206
EURO_set_capture_mode	determines how WinRunner EURO captures currency in terminal emulator applications	207
EURO_set_conversion_mode	sets the EURO conversion run mode in the test script	207
EURO_set_conversion_rate	sets the conversion rate between the EURO currency and a national currency	208
EURO_set_cross_rate	sets the cross rate method between two currencies	209

Function	Description	See Page
EURO_set_currency_threshold	sets the minimum value of an integer which will be considered a currency	210
EURO_set_decimals_precision	sets the number of decimals in the conversion results	210
EURO_set_original_new_currencies	sets the original and new currencies of the application	211
EURO_set_regional_symbols	sets the character used as decimal separator and the character used to separate groups of digits to the left of the decimal	212
EURO_set_triangulation_decimals	sets the default decimals precision for the EURO triangulation	212
EURO_type_mode	disables/enables overriding of automatic currency recognition for all integer objects in a GUI application	213

GUI Checkpoint Functions

Function	Description	See Page
<code>obj_check_gui</code>	compares current GUI data to expected GUI data for any class of object	285
<code>win_check_gui</code>	compares current GUI data to expected GUI data for a window	455

GUI Map Configuration Functions

Function	Description	See Page
<code>get_class_map</code>	returns the standard class associated with a custom class	225
<code>get_record_attr</code>	returns the properties recorded for an object class	226
<code>get_record_method</code>	returns the recording method used for an object class	227
<code>set_class_map</code>	associates a custom class with a standard class	319
<code>set_record_attr</code>	sets the properties to learn for an object class	320
<code>set_record_method</code>	specifies the record method for a class	321
<code>unset_class_map</code>	unbinds a custom class from a standard class	430

GUI Map Editor Functions

Function	Description	See Page
GUI_add	adds an object to a GUI map file	231
GUI_buf_get_desc	returns the physical description of an object in a GUI map file	231
GUI_buf_get_desc_attr	returns the value of an object property in a GUI map file	232
GUI_buf_get_logical_name	returns the logical name of an object in a GUI map file	233
GUI_buf_new	creates a new GUI map file	233
GUI_buf_set_desc_attr	sets the value of a property in a GUI map file	234
GUI_close	closes a GUI map file	234
GUI_close_all	closes all GUI map files	235
GUI_delete	deletes an object from a GUI map file	235
GUI_desc_compare	compares two physical descriptions	236
GUI_desc_get_attr	gets the value of a property from a physical description	236
GUI_desc_set_attr	sets the value of a property	237
GUI_get_name	returns the type of GUI for the application under test	237
GUI_get_window	returns the active window in the GUI map	238
GUI_list_buf_windows	lists all windows in a GUI map file	239
GUI_list_buffers	lists all open GUI map files	239
GUI_list_desc_attrs	returns a list of all property values for an object	240
GUI_list_map_buffers	lists all loaded GUI map files	240

Function	Description	See Page
GUI_list_win_objects	lists all objects in a window	241
GUI_load	loads a GUI map file	242
GUI_map_get_desc	returns the description of an object in the GUI map	243
GUI_map_get_logical_name	returns the logical name of an object in the GUI map	243
GUI_open	opens a GUI map file	244
GUI_save	saves a GUI map file	244
GUI_save_as	saves a GUI map file under a new name	245
GUI_set_window	sets the scope for identifying objects in the GUI map	245
GUI_unload	unloads a GUI map file	246
GUI_unload_all	unloads all loaded GUI map files	246

Icon Object Functions

Function	Description	See Page
<code>icon_move</code>	moves an icon to a new location	249
<code>icon_select</code>	clicks an icon	249

Java Functions

The following functions are available only when WinRunner support for Java is installed and loaded

Function	Description	See Page
<code>java_activate_method</code>	invokes the requested Java method for the given object	252
<code>jco_create</code>	creates a Java object within your application or applet, or within the context of an existing object in your application or applet.	253
<code>jco_free</code>	frees the specified jco object from memory.	254
<code>jco_free_all</code>	frees all jco objects from memory.	254
<code>java_fire_event</code>	simulates an event on a Java object	252
<code>jdc_aut_connect</code>	establishes a connection between WinRunner and Java applications	255
<code>method_wizard</code>	Launches the Java Method wizard, which enables you to view the methods associated with any jco object in your application or applet and to generate the appropriate <code>java_activate_method</code> statement for one of the displayed methods.	281

List Object Functions

Function	Description	See Page
<code>list_activate_item</code>	activates an item	256
<code>list_check_info</code>	checks the value of a list property	257
<code>list_check_item</code>	checks the content of an item in a list	257
<code>list_check_selected</code>	checks that the specified item is selected	258
<code>list_collapse_item</code>	hides items in a tree view object	258
<code>list_deselect_item</code>	deselects an item	259
<code>list_deselect_range</code>	deselects all items between two specified items	259
<code>list_drag_item</code>	drags an item from a source list	260
<code>list_drop_on item</code>	drops an object onto a target list item	260
<code>list_expand_item</code>	displays hidden items in a tree view object	261
<code>list_extend_item</code>	adds an item to the items already selected	262
<code>list_extend_multi_items</code>	adds multiple items to the items already selected	262
<code>list_extend_range</code>	selects a range of items and adds them to the items currently selected	263
<code>list_get_checked_items</code>	returns the value of items marked as checked	264
<code>list_get_column_header</code>	returns the value of a ListView column header	264
<code>list_get_info</code>	returns the value of a list property	265
<code>list_get_item</code>	returns the contents of an item	265
<code>list_get_item_coord</code>	returns the dimensions and coordinates of the list item	266

Function	Description	See Page
list_get_item_info	returns the state of a list item	267
list_get_item_num	returns the position of an item	267
list_get_selected	returns the currently selected item	268
list_get_subitem	returns the value of the ListView subitem	269
list_rename_item	activates an item's edit mode in order to rename it	269
list_select_item	selects an item in a list	270
list_select_multi_items	selects items in a multiple-selection container object	271
list_select_range	selects all items between two specified items	271
list_set_item_state	sets the state of an icon of the specified ListView or TreeView	272
list_wait_info	waits for the value of a list property	273

Menu Object Functions

Function	Description	See Page
menu_get_desc	returns the physical description of a menu	278
menu_get_info	returns the value of a menu property	278
menu_get_item	returns the contents of an item	279
menu_get_item_num	returns the position of an item	280
menu_select_item	selects an item	280
menu_wait_info	waits for the value of a menu property	281

Object Functions

Function	Description	See Page
obj_check_bitmap	compares a current object bitmap to an expected bitmap	285
obj_check_gui	compares current GUI data to expected GUI data	285
obj_check_info	checks the value of an object property	286
obj_click_on_text	clicks on text in an object	287
obj_drag	begins dragging an object	288
obj_drop	ends dragging an object	288
obj_exists	checks if an object is displayed	289
obj_find_text	returns the location of a string within an object	290
obj_get_desc	returns an object's physical description	291
obj_get_info	returns the value of an object property	291
obj_get_text	reads text from an object	292
obj_highlight	highlights an object	293
obj_mouse_click	clicks on an object	293
obj_mouse_dbl_click	double-clicks on an object	294
obj_mouse_drag	drags the mouse within an object	295
obj_mouse_move	moves the mouse within an object	296
obj_move_locator_text	moves the mouse to a string in an object	297
obj_set_info	sets the value of an object property	298
obj_type	sends keyboard input to an object	299
obj_wait_bitmap	waits for an object bitmap	299
obj_wait_info	waits for the value of an object property	300

Oracle Functions

The following functions are available only when WinRunner support for Oracle is installed and loaded

Function	Description	See Page
<code>edit_activate</code>	double-clicks an object in an Oracle application	186
<code>edit_set_focus</code>	focuses on an object in an Oracle application	196
<code>lov_get_item</code>	retrieves an item from a list of values in an Oracle application	276
<code>lov_select_item</code>	selects an item from a list of values in an Oracle application	276

PowerBuilder Functions

The following functions are available only when WinRunner support for PowerBuilder is installed and loaded

Function	Description	See Page
<code>datawindow_get_info</code>	retrieves the value of a DataWindow object property	157
<code>datawindow_text_click</code>	clicks a DataWindow text object	157
<code>datawindow_text_dbl_click</code>	double-clicks a DataWindow text object	158

Scroll Object Functions

Function	Description	See Page
<code>scroll_check_info</code>	checks the value of a scroll property	312
<code>scroll_check_pos</code>	checks the current position of a scroll	312
<code>scroll_drag</code>	drags a scroll to the specified location	313
<code>scroll_drag_from_min</code>	scrolls the specified distance from the minimum position	313
<code>scroll_get_info</code>	returns the value of a scroll property	314
<code>scroll_get_max</code>	returns the value of a scroll at its maximum (end) position	314
<code>scroll_get_min</code>	returns the value of the scroll at its minimum (start) position	315
<code>scroll_get_pos</code>	returns the current scroll position	315
<code>scroll_get_selected</code>	returns the minimum and maximum values of the selected range on a slider	316
<code>scroll_line</code>	scrolls the specified number of lines	316
<code>scroll_max</code>	sets a scroll to the maximum (end) position	317
<code>scroll_min</code>	sets a scroll to the minimum (start) position	317
<code>scroll_page</code>	moves a scroll the specified number of pages	318
<code>scroll_wait_info</code>	waits for the value of a scroll property	318

Siebel Functions

The following functions are available only when WinRunner support for Siebel is installed and loaded

Function	Description	See Page
siebel_click_history	clicks the history button	323
siebel_connect_repository	connects to the Siebel repository database	324
siebel_get_active_applet	returns the active applet name	324
siebel_get_active_buscomp	returns the active business component name	325
siebel_get_active_busobj	returns the active business object name	326
siebel_get_active_control	returns the active control name	326
siebel_get_active_view	returns the active view name	327
siebel_get_chart_data	returns the legend data and chart values from the specified chart	328
siebel_get_control_value	returns the active control value	329
siebel_goto_record	navigates to the specified record	329
siebel_navigate_view	navigates to the specified view	330
siebel_obj_get_info	returns the value of a single Siebel object property from the Siebel repository database	330
siebel_obj_get_properties	returns all properties of a Specified siebel object in the Siebel repository database.	331
siebel_select_alpha	selects a letter button from the alpha tab bar	332
siebel_set_active_applet	sets the specified applet as the active applet.	333

Function	Description	See Page
siebel_set_active_control	sets the specified control as the active control	333
siebel_set_control_value	sets a new value for the active control	334
siebel_terminate	closes the Siebel application	334

Spin Object Functions

Function	Description	See Page
<code>spin_down</code>	scrolls a spin control down a specified number of times	335
<code>spin_get_info</code>	returns the value of a spin property	336
<code>spin_get_pos</code>	returns the position of a spin object	336
<code>spin_get_range</code>	returns the minimum and maximum positions of a spin	337
<code>spin_max</code>	sets a spin to its maximum value	337
<code>spin_min</code>	sets a spin to its minimum value	338
<code>spin_next</code>	sets a spin to its next value	338
<code>spin_prev</code>	sets a spin to its previous value	339
<code>spin_set</code>	sets a spin to the specified value	339
<code>spin_up</code>	scrolls a spin control up the specified number of times	340
<code>spin_wait_info</code>	waits for the value of a spin property	340

Static Text Object Functions

Function	Description	See Page
<code>static_check_info</code>	checks the value of a static text object property	343
<code>static_check_text</code>	checks the contents of a static text object	344
<code>static_get_info</code>	returns the value of a static text property	344
<code>static_get_text</code>	returns the contents of a static text object	345
<code>static_wait_info</code>	waits for the value of a static text property	345

Statusbar Functions

Function	Description	See Page
<code>statusbar_get_field_num</code>	returns the numeric index of a field on a status bar	346
<code>statusbar_get_info</code>	returns the value of a status bar property	346
<code>statusbar_get_text</code>	reads text from a field on a status bar	347
<code>statusbar_wait_info</code>	waits for the value of a status bar property	347

Synchronization Functions

Function	Description	See Page
<code>button_wait_info</code>	waits for the value of a button property	140
<code>edit_wait_info</code>	waits for the value of an edit property	198
<code>list_wait_info</code>	waits for the value of a list property	273
<code>menu_wait_info</code>	waits for the value of a menu property	281
<code>obj_wait_info</code>	waits for the value of an object property	300
<code>scroll_wait_info</code>	waits for the value of a scroll property	318
<code>spin_wait_info</code>	waits for the value of a spin property	340
<code>static_wait_info</code>	waits for a the value of a static text property	345
<code>statusbar_wait_info</code>	waits for the value of a status bar property	347
<code>tab_wait_info</code>	waits for the value of a tab property	352
<code>win_wait_info</code>	waits for the value of a window property	472

Tab Object Functions

Function	Description	See Page
tab_get_info	returns the value of a tab property	350
tab_get_item	returns the name of a tab item	350
tab_get_selected	returns the name of the selected tab item	351
tab_select_item	selects a tab item	351
tab_wait_info	waits for the value of a tab property	352

Table Functions

Function	Description	See Page
tbl_activate_cell	double-clicks the specified cell in a table	353
tbl_activate_col	double-clicks the specified column	355
tbl_activate_header	double-clicks the specified column header in a table	356
tbl_activate_row	double-clicks the specified row	357
tbl_deselect_col	deselects the specified column	359
tbl_deselect_cols_range	deselects the specified range of columns	360
tbl_deselect_row	deselects the specified row	361
tbl_deselect_rows_range	deselects the specified range of rows	361
tbl_extend_col	adds a column to the currently selected columns	363
tbl_extend_cols_range	adds columns to the currently selected columns	364
tbl_extend_row	adds a row to the currently selected rows	365

Function	Description	See Page
tbl_extend_rows_range	adds rows to the currently selected rows	366
tbl_get_cell_data	retrieves the contents of the specified cell from a table	367
tbl_get_cols_count	retrieves the number of columns in a table	369
tbl_get_column_name	retrieves the column header name of the specified column in a table	370
tbl_get_column_names	returns the names and number of columns in a table for PowerBuilder applications	371
tbl_get_rows_count	retrieves the number of rows in the specified table	372
tbl_get_selected_cell	returns the cell currently in focus in a table	373
tbl_get_selected_row	returns the row currently selected in a table	375
tbl_select_cells_range	selects the specified range of cells	376
tbl_select_col_header	clicks the specified column header of a table	377
tbl_select_cols_range	selects the specified range of columns	379
tbl_select_rows_range	selects the specified range of rows	380
tbl_set_cell_data	sets the contents of a cell to the specified text in a table	381
tbl_set_cell_focus	sets the focus to the specified cell in a table	383
tbl_set_selected_cell	selects the specified cell in a table	385
tbl_set_selected_col	selects the specified column in a table	387
tbl_set_selected_row	selects the specified row in a table	388

Terminal Emulator Functions

The following functions are available for the Year 2000 add-in and WinRunner EURO users only:

Function	Description	See Page
TE_add_screen_name_location	instructs WinRunner where to look for the logical name of a screen	391
TE_bms2gui	teaches WinRunner the user interface from a BMS file	392
TE_check_text	captures and compares the text in a terminal emulator window	392
TE_create_filter	creates a filter in the test database	393
TE_define_sync_keys	sets keys that enable automatic synchronization in type , win_type and obj_type commands	394
TE_delete_filter	deletes a specified filter from the test database	395
TE_edit_field	inserts text into an unprotected field	395
TE_edit_hidden_field	inserts text into a hidden field	396
TE_edit_screen	types a string in the specified location in a screen	396
TE_find_text	returns the location of a specified string	397
TE_force_send_key	defines a key causing a screen to change	398
TE_get_active_filter	returns the coordinates of a specified active filter.	398

Function	Description	See Page
TE_get_auto_reset_filters	indicates whether or not filters are automatically deactivated at the end of a test run	399
TE_get_auto_verify	indicates whether automatic text verification is on or off	400
TE_get_cursor_position	returns the position of the cursor	400
TE_get_field_content	returns the contents of a field to a variable	401
TE_get_filter	returns the properties of a specified filter	401
TE_get_merge_rule	returns the rule for merging fields	402
TE_get_refresh_time	returns the time WinRunner waits for the screen to refresh	403
TE_get_screen_name_location	returns the screen name location	403
TE_get_sync_time	returns the system synchronization time	404
TE_get_text	reads text from screen and stores it in a string	404
TE_get_timeout	returns the current synchronization time	405
TE_merge_fields	sets the rule for merging fields	405
TE_reset_all_filters	deactivates all filters in a test	406
TE_reset_all_force_send_key	deactivates the execution of TE_force_send_key functions	406
TE_reset_all_merged_fields	deactivates the merging of fields	406
TE_reset_filter	deactivates a specified filter	407

Function	Description	See Page
TE_reset_screen_name_location	resets the screen name location to 0	407
TE_send_key	sends to the mainframe the specified F-key function	408
TE_set_auto_reset_filters	deactivates the automatic reset of filters when a test run is completed	408
TE_set_auto_transaction	defines a recorded TE_wait_sync statement as a transaction	409
TE_set_auto_verify	activates/deactivates automatic text	409
TE_set_BMS_name_tag	changes a name tag that appears in your BMS file	410
TE_set_cursor_position	defines the position of the cursor	410
TE_set_field	specifies the field that will receive subsequent input	411
TE_set_filter	creates and activates a filter	411
TE_set_filter_mode	specifies whether to assign filters to all screens or to the current screen	412
TE_set_record_method	specifies the recording method for operations on terminal emulator objects	413
TE_set_refresh_time	sets the interval that WinRunner waits for the screen to refresh	413
TE_set_screen_name_location	resets the screen name location to 0 and instructs WinRunner where to look for the logical name of a screen	414

Function	Description	See Page
TE_set_sync_time	defines the system synchronization time	414
TE_set_timeout	sets the maximum time WinRunner waits for a response from the server	415
TE_set_trailing	determines whether WinRunner types spaces and tabs in fields during test execution	415
TE_user_attr_comment	enables a user to add a user-defined comment property to the physical description of fields in the GUI map	416
TE_user_reset_all_attr_comment	resets all user-defined comment properties	416
TE_wait_field	waits for a specified string in a specified field to appear on screen	417
TE_wait_string	waits for a string to appear on screen	417
TE_wait_sync	instructs WinRunner to wait for the terminal emulator screen to be redrawn	418

Text Checkpoint Functions

Function	Description	See Page
<code>obj_click_on_text</code>	clicks on text in an object	287
<code>obj_find_text</code>	returns the location of a string in an object	290
<code>obj_get_text</code>	reads text from an object	292
<code>obj_move_locator_text</code>	moves the mouse to a string in an object	297
<code>win_find_text</code>	returns the location of a string in a window	460
<code>win_click_on_text</code>	clicks on text in a window	457
<code>win_get_text</code>	reads text from a window	462
<code>win_move_locator_text</code>	moves the mouse to a string in a window	468

Toolbar Object Functions

Function	Description	See Page
<code>toolbar_button_press</code>	clicks on a toolbar button	422
<code>toolbar_get_button</code>	returns the name of a toolbar button	422
<code>toolbar_get_button_info</code>	returns the value of a toolbar button property	423
<code>toolbar_get_button_num</code>	returns the position of a toolbar button	424
<code>toolbar_get_buttons_count</code>	returns the number of buttons on a toolbar	425
<code>toolbar_select_item</code>	selects an item from a menu-like toolbar, as in Microsoft Internet Explorer 4.0 or the Start menu in Windows 98	425

WAP Functions

The following functions are available only when WinRunner support for WAP applications is installed and loaded:

Function	Description	See Page
phone_append_text	appends the specified text string to the current contents of the phone editor	303
phone_edit_set	replaces the contents of the phone editor with the specified text string	303
phone_get_name	returns the model name of the phone	304
phone_GUI_load	loads the GUI map for the specified Phone.com phone	304
phone_key_click	clicks a phone key	305
phone_navigate	directs the phone to connect to the specified site	305
phone_sync	recorded after any phone navigation on the Nokia emulator and instructs WinRunner to wait until the phone is ready to handle the next operation	306

Web Functions

The following functions are available only when the WebTest add-in is installed and loaded:

Function	Description	See Page
web_browser_invoke	invokes the browser and opens a specified site	433
web_cursor_to_image	moves the cursor to an image on a page.	433
web_cursor_to_label	moves the cursor to a label on a page	434
web_cursor_to_link	moves the cursor to a link on a page	434
web_cursor_to_obj	moves the cursor to an object on a page	435
web_event	runs an event on the specified object	435
web_file_browse	clicks a browse button	436
web_file_set	sets the text value in a file-type object	437
web_find_text	returns the location of text within a page	437
web_frame_get_text	retrieves the text content of a page	438
web_frame_get_text_count	returns the number of occurrences of a regular expression in a page	439
web_frame_text_exists	returns a text value if it is found in a frame	439
web_get_run_event_mode	returns the current run mode	440
web_get_timeout	returns the maximum time that WinRunner waits for response from the web	440

Function	Description	See Page
web_image_click	clicks a hypergraphic link or an image	441
web_label_click	clicks the specified label	441
web_link_click	clicks a hypertext link	442
web_link_valid	checks whether a URL name of a link is valid (not broken)	442
web_obj_click	clicks an object in a frame	443
web_obj_get_child_item	returns the description of the children in an object	443
web_obj_get_child_item_count	returns the count of the children in an object	444
web_obj_get_info	returns the value of an object property	445
web_obj_get_text	returns a text string from an object	445
web_obj_get_text_count	returns the number of occurrences of a regular expression string in an object	446
web_obj_text_exists	returns a text value if it is found in an object	447
web_password_encrypt	encrypts a password on a Web page.	447
web_refresh	resets all events to their default settings.	448
web_restore_event_default	resets all events to their default settings	448
web_set_event	sets the event status	449
web_set_run_event_mode	sets the event run mode	450

Function	Description	See Page
web_set_timeout	sets the maximum time WinRunner waits for a response from the Web	451
web_set_tooltip_color	sets the colors for the WebTest ToolTip	451
web_sync	waits for the navigation of a frame to be completed	452
web_tbl_get_cell_data	retrieves the contents of the specified cell from a Web table, starting from the specified character	452
web_url_valid	checks whether a URL is valid	453

Table Functions for WebTest

Function	Description	See Page
<code>tbl_get_cell_data</code>	retrieves the contents of the specified cell from a table	367
<code>tbl_get_cols_count</code>	retrieves the number of columns in a table	369
<code>tbl_get_column_name</code>	retrieves the column header name of the specified column	370
<code>tbl_get_rows_count</code>	retrieves the number of rows in the specified table	372

Window Object Functions

Function	Description	See Page
<code>set_window</code>	specifies the window to receive input, according to the window's logical name	322
<code>_set_window</code>	specifies a window to receive input, according to the window's physical description	322
<code>win_activate</code>	activates a window	454
<code>win_check_bitmap</code>	compares a current window bitmap to an expected bitmap	454
<code>win_check_gui</code>	compares current GUI data to expected GUI data	455
<code>win_check_info</code>	checks the requested window property	456
<code>win_click_help</code>	clicks the help button in a window title bar	456
<code>win_click_on_text</code>	clicks on text in a window	457
<code>win_close</code>	closes a window	458
<code>win_drag</code>	drags an object from a source window	458
<code>win_drop</code>	drops an object on a target window	459

Function	Description	See Page
win_exists	checks whether a window is displayed	459
win_find_text	returns the location of a string in a window	460
win_get_desc	returns the physical description of a window	461
win_get_info	returns the value of a window property	461
win_get_text	reads text from a window	462
win_highlight	highlights a window	463
win_max	maximizes a window	463
win_min	minimizes a window to an icon	464
win_mouse_click	clicks in a window	464
win_mouse_dbl_click	double-clicks in a window	465
win_mouse_drag	drags the mouse in a window	466
win_mouse_move	moves the mouse in a window	467
win_move	moves a window to a new absolute location	467
win_move_locator_text	moves the mouse to a string in a window	468
win_open	opens a window	469
win_resize	resizes a window	469
win_restore	restores a window from a minimized or maximized state to its previous size	470
win_type	sends keyboard input to a window	470
win_wait_bitmap	waits for a window bitmap	471
win_wait_info	waits for the value of a window property	472

Year 2000 Functions

The following functions are available for WinRunner 2000 users only:

Function	Description	See Page
Y2K_age_string	ages date string and returns the aged date	472
Y2K_align_day	ages dates to a business day or to the same day of the week	473
Y2K_calc_days_in_field	calculates the number of days between two dates	474
Y2K_calc_days_in_string	calculates the number of days between two numeric strings	475
Y2K_change_field_aging	overrides aging on a specified date object	475
Y2K_change_original_new_formats	overrides automatic date recognition for a specified object	476
Y2K_check_date	checks all dates in the current screen of a terminal emulator application	477
Y2K_disable_format	disables a date format	477
Y2K_enable_format	enables a date format	478
Y2K_field_to_Julian	translates a date field to a Julian number	478
Y2K_is_date_field	determines whether a field contains a valid date	479
Y2K_is_date_string	determines whether a numeric string contains a valid date	479

Function	Description	See Page
Y2K_is_leap_year	determines whether a year is a leap year	480
Y2K_leading_zero	determines whether to add a zero before single-digit numbers when aging and translating dates	480
Y2K_month_language	sets the language used for month names	481
Y2K_set_aging	sets aging in a test script	481
Y2K_set_attr	sets the record configuration mode for a field	482
Y2K_set_auto_date_verify	automatically generates a date checkpoint for the current screen in a terminal emulator application.	482
Y2K_set_capture_mode	determines how WinRunner 2000 captures dates in terminal emulator applications	483
Y2K_set_replay_mode	changes the Year 2000 run mode in the test script	484
Y2K_set_system_date	changes the system date and time	484
Y2K_set_year_limits	sets the minimum and maximum years valid for date verification and aging	485
Y2K_set_year_threshold	sets the year threshold	485

Function	Description	See Page
<code>Y2K_string_to_Julian</code>	translates a numeric string to a Julian number	486
<code>Y2K_type_mode</code>	disables overriding of automatic date recognition for all date objects in a GUI application	486

Customization Functions

Customization functions let you enhance your testing tool for your own needs. For example, you can add functions to the Function Generator or create custom GUI checkpoints.

Customization functions are divided into the following categories:

- Custom Record Functions
- Custom User Interface Functions
- Function Generator Functions
- GUI Checkpoint Functions

Custom Record Functions

Function	Description	See Page
<code>add_cust_record_class</code>	registers a custom record function and/or logical name function	134
<code>add_record_attr</code>	registers a custom property	136
<code>add_record_message</code>	adds a message to the list of Windows messages that WinRunner processes	136
<code>delete_record_attr</code>	removes a custom property	181

Custom User Interface Functions

Function	Description	See Page
<code>create_browse_file_dialog</code>	displays a browse dialog box from which the user selects a file	153
<code>create_custom_dialog</code>	creates a custom dialog box.	154
<code>create_input_dialog</code>	creates a dialog box with an edit field for use in interactive test execution	155
<code>create_list_dialog</code>	creates a dialog box with a list of items for use in interactive test execution	155
<code>create_password_dialog</code>	creates a password dialog box	156

Function Generator Functions

Function	Description	See Page
<code>generator_add_category</code>	adds a category to the Function Generator	221
<code>generator_add_function</code>	adds a function to the Function Generator	221
<code>generator_add_function_to_category</code>	adds a function defined in the Function Generator to a category	223
<code>generator_add_subcategory</code>	adds a subcategory to a category in the Function Generator	223
<code>generator_set_default_function</code>	sets a default function for a Function Generator category	224

GUI Checkpoint Functions

Function	Description	See Page
<code>gui_ver_add_check</code>	registers a new check for a GUI checkpoint	247
<code>gui_ver_add_check_to_class</code>	adds a check to an object class, which can be viewed in the GUI Checkpoint dialog boxes	247
<code>gui_ver_add_class</code>	adds a checkpoint for a new object class	248
<code>gui_ver_set_default_checks</code>	sets default checks for a GUI object class	248

Standard Functions

Standard functions include all the general elements of a programming language, such as basic input and output, control-flow, mathematical, and array functions.

Standard functions are divided into the following categories:

- ▶ Arithmetic Functions
- ▶ Array Functions
- ▶ Call Statements
- ▶ Compiled Module Functions
- ▶ Exception Handling Functions
- ▶ I/O Functions
- ▶ Load Testing Functions
- ▶ Miscellaneous Functions
- ▶ Operating System Functions
- ▶ Password Functions
- ▶ QuickTest 2000 Functions
- ▶ String Functions
- ▶ TDAPI Functions
- ▶ Testing Option Functions
- ▶ TestDirector Functions
- ▶ Time-Related Functions

Arithmetic Functions

Function	Description	See Page
<code>atan2</code>	returns the arctangent of y/x , in radians	137
<code>cos</code>	returns the cosine of an angle, in radians	153
<code>exp</code>	calculates the exponential function of ex	215
<code>int</code>	returns the integer part of a real number	250
<code>log</code>	returns a natural logarithm	275
<code>rand</code>	returns a pseudo-random real number	308
<code>sin</code>	calculates the sine of an angle	335
<code>sqrt</code>	returns the square root of its argument	342
<code>srand</code>	defines a seed parameter for the <code>rand</code> function	342

Array Functions

Function	Description	See Page
<code>delete</code>	removes an element from an array	180
<code>split</code>	divides an input string into fields, stores them in an array, and indicates the number of fields generated	341

Call Statements

Function	Description	See Page
call	invokes a test from within another test script	146
call_chain_get_attr	obtains information about a test or function in the current call chain	147
call_chain_get_depth	returns the number of items in the current call chain	147
call_close	invokes a test from within a script and closes the test when the test is completed	148
call_ex	invokes an Astra QuickTest test from within a WinRunner test script	149
return	returns a value to the calling function or test	310
texit	stops execution of a called test	419
treturn	stops a called test and returns control to the calling test	426

Compiled Module Functions

Function	Description	See Page
load	loads a compiled module into memory	273
reload	removes a compiled module from memory and loads it again	308
unload	removes a compiled module or selected functions from memory	428

Exception Handling Functions

Function	Description	See Page
<code>define_object_exception</code>	defines a GUI object exception	178
<code>define_popup_exception</code>	defines a popup window exception	179
<code>define_tsl_exception</code>	defines a TSL exception	180
<code>exception_off</code>	deactivates handling for an exception	214
<code>exception_off_all</code>	deactivates handling of all exceptions	214
<code>exception_on</code>	enables detection and handling of a previously defined exception	215

I/O Functions

Function	Description	See Page
<code>file_close</code>	closes a file opened with <code>file_open</code>	216
<code>file_compare</code>	compares the contents of two files	216
<code>file_getline</code>	reads a line from a file	217
<code>file_open</code>	opens a file for reading or printing, or creates a new file	217
<code>file_printf</code>	prints formatted output to a file	218
<code>pause</code>	pauses a test and displays a message	302
<code>report_msg</code>	inserts a message in a test report	310
<code>sprintf</code>	returns a formatted string to a variable	341
<code>str_map_logical_to_visual</code>	converts a logical string to a visual string or vice-versa	348

Load Testing Functions

The following functions are available for LoadRunner GUI Vusers only.

Function	Description	See Page
declare_rendezvous	declares a rendezvous	177
declare_transaction	declares a transaction	178
end_transaction	marks the end of a transaction for performance analysis	198
error_message	sends an error message to the controller	199
get_host_name	returns the name of a host	225
get_master_host_name	returns the name of the controller's host	226
lr_whoami	returns information about the Vuser executing the script	277
output_message	sends a message to the controller	301
rendezvous	sets a rendezvous point in a Vuser script	309
start_transaction	marks the beginning of a transaction for performance analysis	343
user_data_point	records a user-defined data sample	430

Miscellaneous Functions

Function	Description	See Page
eval	evaluates and executes the enclosed TSL statements	213
getenv	returns the value of any environment variable, as defined in the [WrCfg] section of <i>wrun.ini</i> in the WinRunner runtime environment	230
load_16_dll	performs a runtime load of a 16-bit Dynamic Link Library	274
load_dll	performs a runtime load of a Dynamic Link Library	275
nargs	returns the number of arguments passed to the function or test	284
tl_step	divides a test script into sections and inserts a status message in the test results for the previous section. When WinRunner is connected to a TestDirector project, the message is inserted in the TestDirector “step” table for each statement.	420
tl_step_once	divides a test script into sections and inserts a status message in the test results for the previous section. When WinRunner is connected to a TestDirector project, the message is inserted in the TestDirector “step” table once for each step name.	420
unload_16_dll	unloads a 16-bit DLL from memory	429
unload_dll	unloads a DLL from memory	429

Operating System Functions

Function	Description	See Page
dos_system	executes a DOS command	185
invoke_application	invokes a Windows application from within a test script	250

Password Functions

Function	Description	See Page
<code>password_edit_set</code>	sets the value of a password edit field to a given value	301
<code>password_encrypt</code>	encrypts a plain password	302

QuickTest 2000 Functions

The following functions are available for QuickTest 2000 users only:

Function	Description	See Page
<code>qt_force_send_key</code>	instructs WinRunner 2000 to recognize an edit field which prompts a screen change when information is inserted	307
<code>qt_reset_all_force_send_key</code>	negates screen change configurations previously made using the <code>qt_force_send_key</code> function	307

String Functions

Function	Description	See Page
ascii	returns the ASCII code of the first character in a string	137
compare_text	compares two strings	152
index	indicates the position of one string within another	250
length	counts characters in a string	256
match	finds a regular expression in a string	277
split	divides an input string into fields and stores them in an array	341
sprintf	returns a formatted string to a variable	341
substr	extracts a substring from a given string	349
tolower	converts uppercase characters to lowercase	421
toupper	converts lowercase characters to uppercase	426

TDAPI Functions

To add the TDAPI functions to WinRunner's Function Generator, run the *tdapi* test in the *lib* folder of your WinRunner installation directory.

For explanations and examples of all TDAPI functions, refer to the *TestDirector Open Test Architecture Guide*.

Project Connection Functions

Project connection functions let you select the TestDirector remote agent and project to which you want to connect. The TDAPI includes the following project connection functions:

Function	Description
TDServerInitInstance	creates a connection to the TestDirector remote agent
TDServerRelease	closes the connection to the TestDirector remote agent
TDAPI_Connect	connects to the specified project
TDAPI_Disconnect	disconnects from the currently connected project
TDAPI_CreateTDDatabasesList	creates a list of projects.
TDAPI_GetDatabaseNameFromList	retrieves the name of a project from a project list

Test Functions

Test functions let you retrieve information relating to the tests stored in TestDirector's test repository. The TDAPI contains the following test functions:

Function	Description
TDAPI_CreateTest	creates a new test
TDAPI_CreateTestList	creates a list of all tests in the project
TDAPI_DeleteTest	deletes a test

Function	Description
TDAPI_FindTestByPath	locates a test by its file system path
TDAPI_FindTestBySubject Path	locates a test by its subject path
TDAPI_GetTestFieldSize	returns the size of a field in a test.
TDAPI_GetTestFullPath	retrieves the full path of a test
TDAPI_GetTestSubjectPath	retrieves a test's subject path.
TDAPI_GetTestValue	retrieves the value of a field in a test
TDAPI_SetTestValue	updates a field in a test
TDAPI_TestExists	locates a test
TDAPI_TestListMove	steps through a list of tests

Design Steps Functions

TestDirector tests are divided into design steps. These are detailed step-by-step instructions that describe the actions the tester (manual tests) or testing tool (automated tests) should perform as the test is executed. The TDAPI contains the following design steps functions:

Function	Description
TDAPI_CreateDesStep	creates a design step in a test
TDAPI_CreateDesStepList	creates a list of design steps
TDAPI_DeleteDesStep	deletes a design step in a test
TDAPI_DesStepListMove	steps through a list of design steps
TDAPI_GetDesStepFieldSize	returns the size of a design step field
TDAPI_GetDesStepValue	retrieves the value of a field in a design step
TDAPI_SetDesStepValue	updates a field in a design step record

Defect Tracking Functions

Defect records contain errors discovered during test execution. Defect tracking functions let you add, locate, update defect information in your project. The TDAPI contains the following defect tracking functions:

Function	Description
TDAPI_BugListMove	steps through a list of defects
TDAPI_CreateBug	creates a new defect
TDAPI_CreateBugList	creates a list of defects in the project
TDAPI_DeleteBug	deletes a defect from the TestDirector project
TDAPI_GetBugFieldSize	returns the size of a defect field
TDAPI_GetBugValue	retrieves the value of a field in a defect
TDAPI_SetBugValue	updates a field in a defect

Test Set Functions

A test set is a group of tests designed to meet a specific testing goal. For example, to verify that the application under test is functional and stable, you create a sanity test set that checks the application's basic features. The TDAPI contains the following functions to help you build and maintain test sets:

Function	Description
TDAPI_AddTestToCycle	adds a test to a test set
TDAPI_CreateCycle	creates a new test set
TDAPI_CreateCycleList	creates a list of test sets in the project
TDAPI_CreateTestinCycleList	creates a list of test sets in the project
TDAPI_CycleExists	checks a test set exists
TDAPI_CycleListMove	steps through a list of test sets
TDAPI_DeleteCycle	deletes a test set
TDAPI_DeleteTestFromCycle	removes a test from a test set

Function	Description
TDAPI_GetCyclesForTest	retrieves names of the test sets to which the test belongs
TDAPI_GetCycleValue	retrieves value of a field in a test set record
TDAPI_GetCycleFieldSize	returns the size of a field in a test set
TDAPI_GetTestInCycleFieldSize	returns the size (in bytes) of a field of a test in a test set.
TDAPI_GetTestInCycleValue	retrieves the value of a field in a test in a test set record
TDAPI_SetCycleValue	updates a field of a test set record to new value
TDAPI_SetTestInCycleValue	updates the specified field of a test set record to new value
TDAPI_TestInCycleExists	looks for a test in a test set
TDAPI_TestInCycleListMove	steps through a list of tests in a test set

Test Run Functions

A test run stores information about how each test performs during test execution. The TDAPI includes the following functions to let you create and manage test runs:

Function	Description
TDAPI_CreateRun	creates a test run for a test
TDAPI_CreateRunList	creates a list of test runs
TDAPI_DeleteRun	deletes a test run
TDAPI_GetRunFieldSize	returns the size of a field in a test run
TDAPI_GetRunValue	retrieves value of a field in a test run
TDAPI_RunListMove	steps through a list of test runs
TDAPI_SetRunValue	updates a field in a test run record

Test Step Functions

Test steps record the performance of each test step during a test run. Each test step contains detailed information on what actions were performed during each test run. These include the IDs of the test and test run, the name of the step, the status of the step, and the line number of where the step will appear within the test script. The TDAPI contains the following functions to help you create and manage test runs:

Function	Description
TDAPI_AddStepToRun	creates a step in a test run
TDAPI_CreateStepList	creates a list of steps
TDAPI_DeleteStep	deletes a step in a test run
TDAPI_GetStepFieldSize	retrieves size of a field in a step
TDAPI_GetStepValue	returns the value of a field in a step
TDAPI_SetStepValue	updates a step to a new value
TDAPI_StepListMove	steps through a list of defects

Test Plan Tree Functions

The test plan tree is a representation of how information is stored within your project. When you access the project, you use the tree to locate information in the project. The TDAPI contains the following functions to help you create and manage test plan trees:

Function	Description
TDAPI_GetCategoryTreeRoot	returns the ID of a the test plan tree's subject folder
TDAPI_TreeAddNode	adds a folder to the test plan tree
TDAPI_TreeChanged	indicates if changes were made to the test plan tree
TDAPI_TreeCreateRoot	sets a parent folder in the test plan tree
TDAPI_TreeGetChild	returns the ID of a subfolder in a test plan tree folder

Function	Description
TDAPI_TreeGetNodeAttribute	returns the ID of a subfolder in the test plan tree
TDAPI_TreeGetNumberOfChildren	returns the number of subfolders contained in a folder
TDAPI_TreeGetRoot	returns the ID of the current parent folder
TDAPI_TreeGetSubjectIDFromPath	returns the ID of a test plan tree folder

Project Administration Functions

Project administration functions let you create and manage project users, return internal project error information, and view project statistics. The TDAPI includes the following project administration functions:

Function	Description
TDAPI_CreateUser	creates a new user
TDAPI_CreateUserList	creates a list of TestDirector users
TDAPI_DeleteUser	deletes a user
TDAPI_GetFieldProperty	returns information from the System_fields table
TDAPI_GetFunctionStatistics	returns performance statistics of TDAPI functions
TDAPI_GetLastErrorString	returns a description of an error
TDAPI_GetStackErrorString	returns all the errors in the error stack
TDAPI_GetUserFieldSize	returns the size of the field in a user record.
TDAPI_GetUserValue	returns value of a field in a user record
TDAPI_SetUserValue	updates a field in a user record
TDAPI_UserExists	checks whether a user record exists
TDAPI_UserListMove	returns the current user name.

Testing Option Functions

Function	Description	See Page
<code>get_aut_var</code>	returns the value of a variable that determines how WinRunner learns descriptions of objects, records tests, and runs tests on Java applets or applications.	224
<code>getvar</code>	returns the value of a testing option	230
<code>set_aut_var</code>	sets how WinRunner learns descriptions of objects, records tests, and runs tests on Java applets or applications	319
<code>setvar</code>	sets the value of a testing option	323

TestDirector Functions

The following functions are only available when working with TestDirector:

Function	Description	See Page
<code>tddb_get_step_value</code>	returns the value of a field in the "dessteps" table in a TestDirector database	389
<code>tddb_get_test_value</code>	returns the value of a field in the "test" table in a TestDirector database	390
<code>tddb_get_testset_value</code>	returns the value of a field in the "testcycl" table in a TestDirector database.	390
<code>tddb_load_attachment</code>	loads a test's file attachment and returns the file system path of the location where it was loaded.	391
<code>tl_step</code>	divides a test script into sections	420
<code>tl_step_once</code>	divides a test script into sections and inserts a status message in the test results for the previous section	420

Time-Related Functions

Function	Description	See Page
end_transaction	marks the end of a transaction for performance analysis	198
get_time	returns the current system time	229
pause	pauses test execution and displays a message	302
start_transaction	marks the beginning of a transaction for performance analysis	343
time_str	converts the integer returned by get_time to a string	419
wait	causes test execution to pause for the specified amount of time	431

6

Return Values

Unless otherwise specified, functions may return one of the general return values listed below. This function returns one of the return values listed in “General Return Values,” on page 118.

In addition, some functions may return specialized return values.

- ▶ For database functions (**db_**), see also “Return Values for PowerBuilder and Table Functions,” on page 123.
- ▶ For table and PowerBuilder functions (**tbl_** and **datawindow_**), see also “Return Values for Database Functions,” on page 122.
- ▶ For Terminal Emulator functions (**TE_**), see also “Return Values for Terminal Emulator Functions,” on page 124.

General Return Values

Unless otherwise specified, all functions may return one of the general return values listed below.

Error Code	Number	Description
E_OK	0	Operation successful.
E_FILE_OK	0	Operation successful.
E_GENERAL_ERROR	-10001	General error occurred.
E_NOT_FOUND	-10002	Window or object not found.
E_NOT_UNIQUE	-10003	More than one window or object responds to the physical description.
E_ILLEGAL_OPERATION	-10004	Operation invalid for object. For more information, see the note on page 122.
E_OUT_OF_RANGE	-10005	Parameter is out of range.
E_ILLEGAL_PARAMETER	-10006	Specified value for one or more parameters is invalid.
E_FILE_OPEN	-10007	Cannot open file. File may already be open.
E_ILLEGAL_ARGLIST	-10009	Illegal argument list.
E_NOT_IN_MAPPING	-10011	Cannot find window or object in the GUI map.
E_EXIST	-10012	Object already exists.
E_OPERATION_NOT_PERFORMED	-10018	Cannot perform requested operation.
E_FUNCTION_NOT_LOADED	-10019	Specified function is not currently loaded. In the case of a handler function, the exception is undefined.
E_NO_FONT	-10024	No font was loaded.

Error Code	Number	Description
E_SYNTAX	-10025	Syntax error in TSL statement.
E_NO_SVC	-10026	Called function does not exist.
E_FUNCTION_NOT_IMPLEMENTED	-10028	Called function could not be implemented.
E_ATTR_IN_DESC	-10029	Specified property is used in the object's physical description in the GUI map.
E_NO_LABEL	-10030	Label property is not used in the window's physical description in the GUI map.
E_USING_WIN_TITLE	-10031	Error using window title.
E_FILE_NOT_OPEN	-10032	File is not open.
E_FILE_NOT_FOUND	-10033	File is not found.
E_FILE_LINE_TRUNC	-10034	File line is truncated.
E_FILE_EOF	-10035	End of file.
E_FILE_NOT_READ_MODE	-10036	Cannot read file because file is not in read mode.
E_FILE_READ_MODE	-10037	Cannot write to file because file is in read mode.
E_BAD_PATH	-10038	Incorrect path.
E_ACCESS_DENIED	-10039	Access is denied.
E_DISK_FULL	-10040	Disk is full.
E_SHARING_VIOLATION	-10041	Sharing violation.
E_FILE_ERROR	-10042	General file error.
E_NOT_PARAMETER	-10044	Parameter is invalid.

Error Code	Number	Description
E_MAX_COLUMNS_EXCEEDED	-10045	Column cannot be added to the data table because the data table already contains the maximum allowable number of columns (255).
E_NOT_DISPLAYED	-10101	Window, object or data table is not displayed.
E_DISABLED	-10102	Window or object is disabled.
E_IMPROPER_CLASS	-10103	Operation cannot be performed on this object class.
E_ILLEGAL_KEY	-10104	Key or mouse button name is illegal.
E_ITEM_NOT_FOUND	-10105	Item in list or menu not found.
E_NOT_RESPONDING	-10106	Application did not respond within the specified timeout.
E_OBJECT_SYNTAX	-10107	Illegal syntax used.
E_ILLEGAL_NUM_OF_PARAMS	-10112	Number of parameters does not match those for the command.
E_AUT_DISCONNECTED	-10114	The application under test was disconnected.
E_ATTR_NOT_SUPPORTED	-10115	Property in function is not supported.
E_MISMATCH	-10116	Verification mismatch found.
E_ITEM_NOT_UNIQUE	-10117	More than one item in list or menu has this name.
E_TEXT_TOO_LONG	-10118	Text to be inserted exceeds maximum number of characters. The string will be truncated to the appropriate length.
E_DIFF	-10119	GUI checkpoint mismatch found.

Error Code	Number	Description
E_CMP_FAILED	-10120	Comparison failed.
E_CAPT_FAILED	-10121	Capture failed.
E_SET_WIN	-10123	Window setting parameters missing.
E_BITMAP_TIMEOUT	-10124	The wait_bitmap operation exceeded specified wait time.
E_BAD_CHECK_NAME	-10125	Syntax error in requested check.
E_OBJ_CAPT_FAILED	-10126	Capture failed for specified object.
E_UNEXP_WIN	-10127	Window in checklist is not the window in the command.
E_CAPT_FUNC_NOT_FOUND	-10128	Capture function not defined.
E_CMP_FUNC_NOT_FOUND	-10129	Compare function not defined.
E_TSL_ERR	-10130	Syntax error detected.
E_TOOLKIT_MISMATCH	-10131	Incorrect toolkit detected.
E_RECT_COVERED	-10132	Desired rectangle is hidden.
E_RECT_OUT	-10133	Desired rectangle does not appear on screen.
E_AREA_COVERED	-10134	Desired area is hidden.
E_AREA_OUT	-10135	Desired area does not appear on screen.
E_STR_NOT_FOUND	-10136	Text string not located.
E_WAIT_INFO_TIMEOUT	-10137	The wait_info operation exceeded specified wait time.
E_DIFF_SIZE	-10139	Expected and actual bitmaps are different sizes.
E_DROP_WITHOUT_DRAG	-10141	Drop operation is performed without a drag operation preceding it.

Error Code	Number	Description
E_VIR_OBJ	-10142	Function not supported for virtual objects.
E_MISSING_ATTR	-10143	Lack of x-, y-, height, or width coordinates in the description of the virtual object.
E_EDIT_SET_FAILED	-10144	The edit_set operation failed.

Note about E_ILLEGAL_OPERATION: A function may fail if the method does not exist, the parameter number is wrong, the parameter types are wrong, etc. For more information regarding a failure, insert the following statement and then rerun the function. This will provide you with more details.

```
set_aut_var("DEBUG_GCALL", ON);
```

Return Values for Database Functions

Unless otherwise specified in the function description, database functions (**db_**) may return one of the following return values in addition to the regular return values.

Error Code	Number	Description
E_SESSION_NOT_STARTED	-10160	The database session was not started.
E_CONNECTION_FAILED	-10161	The connection to the database failed.
E_SQL_SYNTAX_ERROR	-10162	Syntax error in the SQL statement.
E_PASSED_LAST_ROW	-10163	The row number exceeded the row number of the last row in the table.
E_QUERY_CAPTURE_FAILED	-10164	General error while capturing data.

Return Values for PowerBuilder and Table Functions

Unless otherwise specified, table and PowerBuilder functions (**tbl_** and **datawindow_**) may return one of the following return values in addition to the regular return values.

Error Code	Number	Description
PB_E_NO_PBTAPI	-10145	Internal error.
PB_E_ROW_COL_INVALID	-10146	Parameter is out of range.
PB_E_ROW_INVALID	-10147	Parameter is out of range.
PB_E_DESC_OVERFLOW	-10149	Internal error.
PB_E_DW_LIST_ITEM_NOT_FOUND	-10150	Item not found.
PB_E_DESC_NOT_FOUND	-10151	Internal error.
PB_E_CELL_NOT_VISIBLE	-10152	Cell not visible.
PB_E_PARSE_ERROR	-10153	Internal error.
PB_E_TAPI_ERROR	-10154	Internal error.
PB_E_BUF_NOT_INIT	-10155	Internal error.
PB_E_CELL_NOT_FOUND	-10156	Cell not found.
PB_E_API_ERROR	-10157	General error.
PB_E_INVALID_COL_TYPE	-10158	Unknown column type.
PB_E_ILLEGAL_COORDS	-10159	Illegal coordinates.

Return Values for Terminal Emulator Functions

Unless otherwise specified in the function description, terminal emulator functions (TE_) may return one of the following return values in addition to the regular return values.

WinRunner/TE Error Code	Number	Description
E_PROT_FIELD	-10400	Field is protected and cannot accept input.
E_TERM_DISCONNECTED	-10401	Terminal is probably disconnected.
E_TERM_LOCKED	-10402	Terminal is locked. In an interactive run, the user can continue, pause, or unlock the terminal. In a batch run, WinRunner unlocks the terminal and sends a report message.
E_TERM_BUSY	-10403	Terminal is synchronizing. In an interactive run, user can continue, pause, or perform wait_sync . In a batch run, WinRunner synchronizes and sends a report message.
E_RULE_NOT_FOUND	-10405	Cannot write to a merged field after all merged fields were reset.

7

Alphabetical Reference

This chapter contains an alphabetical reference of all TSL functions in WinRunner. The name of each function appears, along with the type and the category to which the function belongs. The following additional information is provided for each function:

- description
- complete syntax
- parameter definitions
- return values
- availability

For additional information and examples of usage, refer to the *TSL Online Reference*. You can open the *TSL Online Reference* from the WinRunner group in the Start menu or from WinRunner's Help menu. To open the online reference to a specific function, click the context-sensitive Help button and then click a TSL statement in your test script, or place your cursor on a TSL statement in your test script and then press the F1 key. Check Mercury Interactive's Customer Support Web site for updates to the *TSL Online Reference*.

ActiveBar_combo_select_item

Context Sensitive • Active Bar

selects an item in a ComboBox tool.

ActiveBar_combo_select_item (*band_tool* , *item_name*);

band_tool A string containing the band identifier (Name or Caption) and tool identifier (Name, Caption or ToolID), separated by semicolon (;).

The *band identifier* can be specified either by Name or Caption

The *tool identifier* can be specified either by Name, Caption, or ToolID. The ampersand character (&) in Caption is ignored.

item_name Either item text or item number in the "#" format.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is supported for DataDynamics ActiveBar 1.0.

Note: This function is not recordable.

ActiveBar_dump

Context Sensitive • Active Bar

stores information about ActiveBar bands and tools. This information includes captions, names, types and IDs.

ActiveBar_dump (*file_name*);

file_name The file pathname in which the ActiveBar information will be dumped.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for DataDynamics ActiveBar 1.0 and Sheridan ActiveToolbars 1.01.

Note: This function is not recordable.

ActiveBar_select_menu

Context Sensitive • Active Bar

selects a menu item in a toolbar.

ActiveBar_select_menu (*band_tool* [, *events_only*]);

band_tool

A string containing the band identifier (Name or Caption) and tool identifier (Name, Caption or ToolID), separated by semicolon (;).

The *band identifier* can be specified either by Name or Caption

The *tool identifier* can be specified either by Name, Caption, or ToolID. The ampersand character (&) in Caption is ignored.

events_only

TRUE or FALSE.

If this parameter set to TRUE, then executing this function during a test run uses events.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for DataDynamics ActiveBar 1.0 and Sheridan ActiveToolbars 1.01.

Note: The *events_only* parameter is supported only for the DataDynamics ActiveBar.

ActiveBar_select_tool

Context Sensitive • Active Bar

selects a tool in the toolbar.

ActiveBar_select_tool (*band_tool* [, *events_only*]);

band_tool

A string containing the band identifier (Name or Caption) and tool identifier (Name, Caption or ToolID), separated by semicolon (;).

The *band identifier* can be specified either by Name or Caption

The *tool identifier* can be specified either by Name, Caption, or ToolID. The ampersand character (&) in Caption is ignored.

events_only

TRUE or FALSE.

If this parameter set to TRUE, then executing this function during a test run uses events.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for DataDynamics ActiveBar 1.0 and Sheridan ActiveToolbars 1.01.

Note: The *events_only* parameter is supported only for the DataDynamics ActiveBar.

ActiveX_activate_method

Context Sensitive • ActiveX/Visual Basic

invokes an ActiveX method of an ActiveX control.

ActiveX_activate_method (*object*, *ActiveX_method*, *return_value*
[*param₄*, ..., *param₈*]);

object The name of the object.
ActiveX_method The ActiveX control method to be invoked.

Tip: You can use the ActiveX tab in the GUI Spy to view the methods of an ActiveX control.

return_value Return value of the method.
param₄, ..., *param₈* The parameters of the method (optional). These parameters may only be call variables and not constants.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available only for the following add-ins: ActiveX, PowerBuilder, Visual Basic, or WebTest.

ActiveX_get_info

Context Sensitive • ActiveX/Visual Basic

returns the value of an ActiveX/Visual Basic control property. The property can have no parameters or a one or two-dimensional array. Properties can also be nested.

For an ActiveX property without parameters, the syntax is as follows:

```
ActiveX_get_info ( ObjectName, PropertyName, OutValue [ , IsWindow ] );
```

For an ActiveX property that is a one-dimensional array, the syntax is as follows:

```
ActiveX_get_info ( ObjectName, PropertyName ( X ) , OutValue [ , IsWindow ] );
```

For an ActiveX property that is a two-dimensional array, the syntax is as follows:

```
ActiveX_get_info ( ObjectName, PropertyName ( X, Y ) , OutValue [ , IsWindow ] );
```

ObjectName The name of the ActiveX/Visual Basic control.

PropertyName Any ActiveX/Visual Basic control property.

Tip: You can use the ActiveX tab in the GUI Spy to view the properties of an ActiveX control.

OutValue The output variable that stores the property value.

IsWindow An indication of whether the operation is performed on a window. If it is, set this parameter to TRUE.

Note: The *IsWindow* parameter should be used only when this function is applied to a Visual Basic form to get its property or a property of its sub-object. In order to get a property of a label control you should set this parameter to TRUE.

Note: To get the value of nested properties, you can use any combination of indexed or non-indexed properties separated by a dot. For example:

```
ActiveX_get_info("Grid", "Cell(10,14).Text", Text);
```

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available only for the following add-ins: ActiveX, PowerBuilder, Visual Basic, or WebTest.

ActiveX_set_info

Context Sensitive • ActiveX/Visual Basic

sets the value of an ActiveX/Visual Basic control property. The property can have no parameters or a one or two-dimensional array. Properties can also be nested.

For an ActiveX property without parameters, the syntax is as follows:

```
ActiveX_set_info ( ObjectName, PropertyName, Value [ , Type [ , IsWindow ] ] );
```

For an ActiveX property that is a one-dimensional array, the syntax is as follows:

```
ActiveX_set_info ( ObjectName, PropertyName ( X ) , Value [ , Type [ , IsWindow ] ] );
```

For an ActiveX property that is a two-dimensional array, the syntax is as follows:

```
ActiveX_set_info ( ObjectName, PropertyName ( X , Y ) , Value [ , Type [ , IsWindow ] ] );
```

ObjectName The name of the ActiveX/Visual Basic control.

PropertyName Any ActiveX/Visual Basic control property.

Tip: You can use the ActiveX tab in the GUI Spy to view the properties of an ActiveX control.

<i>Value</i>	The value to be applied to the property.	
<i>Type</i>	The value type to be applied to the property. The following types are available:	
VT_I2 (short)	VT_I4 (long)	VT_R4 (float)
VT_R8 (float double)	VT_DATE (date)	VT_BSTR (string)
VT_ERROR (S code)	VT_BOOL (boolean)	VT_UI1 (unsigned char)
<i>IsWindow</i>	An indication of whether the operation is performed on a window. If it is, set this parameter to TRUE.	

Note: The *IsWindow* parameter should be used only when this function is applied to a Visual Basic form to set its property or a property of its sub-object. In order to get a property of a label control you should set this parameter to TRUE.

Note: To set the value of nested properties, you can use any combination of indexed or non-indexed properties separated by a dot. For example:

```
ActiveX_set_info("Book", "Chapter(7).Page(2).Caption", "SomeText");
```

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available only for the following add-ins: ActiveX, PowerBuilder, Visual Basic, or WebTest.

add_cust_record_class**Customization • Custom Record**

associates a custom record function or a logical name function with a custom class.

```
add_cust_record_class ( MSW_class, dll_name [ , rec_func [ , log_name_func ] ] );
```

<i>MSW_class</i>	The custom class with which the function is associated.
<i>dll_name</i>	The full path of the DLL containing the function.
<i>rec_func</i>	The name of the custom record function defined in the DLL. This custom record function returns the statement recorded in the test script.
<i>log_name_func</i>	The name of the logical name function defined in the DLL. This logical name function supplies custom logical names for GUI objects in the custom class, <i>MSW_class</i> .

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

add_dlph_obj

Context Sensitive • Delphi

adds a Delphi object.

add_dlph_obj (*MSW_class*, *class*, *oblig_attr*, *optional_attr*, *default_check_prop*, *item*);

<i>MSW_class</i>	The custom class with which the function is associated.
<i>class</i>	The name of the Mercury class, <i>MSW_class</i> , or <i>X_class</i> .
<i>oblig_attr</i>	A list of obligatory properties (separated by blank spaces).
<i>optional_attr</i>	A list of optional properties (separated by blank spaces), in descending order, to add to the description until the object is uniquely identified.
<i>default_check_prop</i>	The default status of the object.
<i>item</i>	Indicates whether the item is an object or a grid.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available only for WinRunner with Delphi support.

add_record_attr

Customization • Custom Record

registers a custom property.

add_record_attr (*attr_name*, *dll_name*, *query_func_name*, *verify_func_name*);

<i>attr_name</i>	The name of the custom property to register. This cannot be a standard WinRunner property name.
<i>dll_name</i>	The full path of the DLL in which the query and verify functions are defined.
<i>query_func_name</i>	The name of the query function included in the DLL.
<i>verify_func_name</i>	A WinRunner standard property verification function (see below) or a custom property verification function included in the DLL.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

add_record_message

Customization • Custom Record

adds a message to the list of Windows messages.

add_record_message (*message_number*);

<i>message_number</i>	The number or identifier of the Windows message.
-----------------------	--

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

ascii

Standard • String

returns the ASCII code of the first character in a string.

ascii (*string*);

string A string expression.

Return Values

This function returns the ASCII code of the first character in the string.

Availability

This function is always available.

atan2

Standard • Arithmetic

returns the arctangent of y/x.

atan2 (*y*, *x*);

Return Values

This function returns a real number.

Availability

This function is always available.

button_check_info

Context Sensitive • Button Object

checks the value of a button property.

button_check_info (*button*, *property*, *property_value*);

button The logical name of the button.

property The property to check.

property_value The property value.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

button_check_state

Context Sensitive • Button Object

checks the state of a radio or check button.

button_check_state (*button*, *state*);

<i>button</i>	The logical name of the button.
<i>state</i>	The state of the button. The value can be 1 (ON) or 0 (OFF). A value of 2 indicates that the button is DIMMED.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

button_get_info

Context Sensitive • Button Object

returns the value of a button property.

button_get_info (*button*, *property*, *out_value*);

<i>button</i>	The logical name of the button.
<i>property</i>	Any of the properties listed in the <i>User's Guide</i> .
<i>out_value</i>	The output variable that stores the value of the specified property.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

button_get_state**Context Sensitive • Button Object**

returns the state of a radio or check button.

button_get_state (*button*, *out_state*);

<i>button</i>	The logical name of the button.
<i>out_state</i>	The output variable that stores the state of the button. For check and radio buttons, the value can be 1 (ON) or 0 (OFF). A value of 2 indicates that the button is DIMMED. For push buttons, the value is 0.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

button_press**Context Sensitive • Button Object**

clicks on a push button.

button_press (*button*);

<i>button</i>	The logical name of the button.
---------------	---------------------------------

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

button_set

Context Sensitive • Button Object

sets the state of a radio or check button.

button_set (*button*, *state*);

<i>button</i>	The logical name of the button.
<i>state</i>	For a radio button, one of the following states can be specified: DIMMED, ON, OFF, or TOGGLE. The TOGGLE option reverses the current state between ON and OFF. For a check button, the state can be ON or OFF.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

button_wait_info

Context Sensitive • Button Object

waits for the value of a button property.

button_wait_info (*button*, *property*, *value*, *time*);

<i>button</i>	The logical name of the button.
<i>property</i>	Any of the properties listed in the <i>WinRunner User's Guide</i> .
<i>value</i>	The property value.
<i>time</i>	Indicates the maximum interval, in seconds, before the next statement is executed.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

calendar_activate_date**Context Sensitive • Calendar**

double-clicks the specified date in a calendar.

calendar_activate_date (*calendar*, *date*);

calendar The logical name of the calendar.

date The date in the calendar.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for ActiveX controls.

This function is available for calendars included in Visual Studio version 6 and higher and in Internet Explorer Active Desktop version 4 and higher.

calendar_get_selected

Context Sensitive • Calendar

retrieves and counts the selected dates in a calendar.

```
calendar_get_selected ( calendar, selected_dates, selected_dates_count  
    [ , selected_time ] );
```

<i>calendar</i>	The logical name of the calendar.
<i>selected_dates</i>	The output variable that stores the dates selected in the calendar.
<i>selected_dates_count</i>	The output variable that stores the total number of selected dates in the calendar.
<i>selected_time</i>	The output variable that stores the time selected. This parameter is valid for the Date Time control only.

Return Values

This function returns a string representing the date and an integer representing the number of dates chosen.

Availability

This function is supported for ActiveX controls.

This function is available for calendars included in Visual Studio version 6 and higher and in Internet Explorer Active Desktop version 4 and higher.

calendar_get_status

Context Sensitive • Calendar

retrieves the selection status.

```
calendar_get_status ( calendar, selection_status );
```

<i>calendar</i>	The logical name of the calendar.
<i>selection_status</i>	The status of the date; it may either be valid or invalid.

Based on the validity of the date, **calendar_get_status** retrieves the integer 1 (valid) or 0 (invalid).

Return Values

This function returns an integer, 1 or 0, based on whether or not the status is valid or invalid.

Availability

This function is supported for the Date Time control only.

This function is available for calendars included in Visual Studio version 6 and higher and in Internet Explorer Active Desktop version 4 and higher.

calendar_get_valid_range**Context Sensitive • Calendar**

retrieves the range of allowed values for a calendar control.

calendar_get_valid_range (*calendar*, *in_range_type*, *allowed_min_time*, *allowed_max_time*);

<i>calendar</i>	The logical name of the calendar.
<i>in_range_type</i>	DATE_TYPE (1) minimum and maximum allowed date values for the control. TIME_TYPE (0) minimum and maximum allowed time values for the control.
<i>allowed_min_time</i>	The minimum allowed date or time of the control, according to the <i>in_range_type</i> parameter.
<i>allowed_max_time</i>	The maximum allowed date or time of the control, according to the <i>in_range_type</i> parameter.

Return Values

This function returns two strings representing the minimum and maximum dates allowed.

Availability

This function is available for the Date Time and Month Calendar controls only.

This function is available for calendars included in Visual Studio version 6 and higher and in Internet Explorer Active Desktop version 4 and higher.

calendar_select_date

Context Sensitive • Calendar

clicks the specified date in a calendar.

calendar_select_date (*calendar*, *date*);

<i>calendar</i>	The logical name of the calendar.
<i>date</i>	The date is recorded in the following format: DD- <i>MMM</i> -YYYY. Numbers as well letters may be used for months.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for ActiveX controls only.

This function is available for calendars included in Visual Studio version 6 and higher and in Internet Explorer Active Desktop version 4 and higher.

calendar_select_range

Context Sensitive • Calendar

selects a range of dates in the DD-MM-YYYY date format.

calendar_select_range (*calendar*, *start_date*, *end_date*);

<i>calendar</i>	The logical name of the calendar.
<i>start_date</i>	The first day in the range.
<i>end_date</i>	The last day in the range.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for the Month Calendar control with the multiple selection policy only.

This function is available for calendars included in Visual Studio version 6 and higher and in Internet Explorer Active Desktop version 4 and higher.

calendar_select_time**Context Sensitive • Calendar**

when a date is recorded with a time, WinRunner records the time using this function in the HH:MM:SS time format.

calendar_select_time (*calendar*, *time*);

calendar The logical name of the calendar.

time The time selected in the HH:MM:SS format.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is valid for the Date Time control only.

This function is available for calendars included in Visual Studio version 6 and higher and in Internet Explorer Active Desktop version 4 and higher.

calendar_set_status**Context Sensitive • Calendar**

sets the selection status.

calendar_set_status (*calendar*, *selection_status*);

calendar The logical name of the calendar.

selection_status The status of the date may be valid (1) or invalid (2). The valid selection status selects the check box and the invalid selection clears the check box.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is valid for the Date Time control only.

This function is available for calendars included in Visual Studio version 6 and higher and in Internet Explorer Active Desktop version 4 and higher.

call

Standard • Call Statements

invokes a test from within a test script.

call *test_name* ([*parameter*₁, *parameter*₂, ... *parameter*_{*n*}]);

test_name The name of the test to invoke.

parameter The parameters defined for the called test.

Note: You can parameterize a **call** statement using the **eval** function in order to call several tests and the relevant parameters for each within a single **call** loop. For more information, see **eval** on page 213.

Return Values

The **call** statement returns an empty string, unless the called test returns an expression using **return** or **textit**.

Availability

This function is always available.

Note: The **call** statement is not a function. Therefore, it does not appear in the Function Generator.

call_chain_get_attr**Standard • Call Statements**

returns information about a test or function in the call chain.

call_chain_get_attr (*property*, *level*, *out_value*);

<i>property</i>	One of the properties listed in the table below.
<i>level</i>	A number indicating the test or function in the call chain. 0 indicates the current test/function; 1 indicates the test/function that called the current item; 2 indicates two levels above the current item, etc.
<i>out_value</i>	The output variable that stores the value of the specified <i>property</i> .

Property	Description
testname	The name of the test/function specified by level.
line_no	The line number where the test call statement or function call appears.
type	Indicates whether the call item is a test or a function.
function	If the specified call item is a function, its name.

Return Values

This statement returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

call_chain_get_depth**Standard • Call Statements**

returns the number of items in the call chain.

call_chain_get_depth ();

The **call_chain_get_depth** statement returns the number of tests or functions in the current call chain.

Return Values

This statement returns the number of items in the call chain, or 0 when the call chain is empty.

Availability

This function is always available.

call_close

Standard • Call Statements

invokes a test from within a script and closes the test when the test is completed.

call_close *test_name* ([*parameter*₁, *parameter*₂, ... *parameter*_{*n*}]);

test_name The name of the test to invoke.

parameter The parameters defined for the called test.

Note: You can parameterize a **call_close** statement using the **eval** function in order to call several tests and the relevant parameters for each within a single **call_close** loop. For more information, see **eval** on page 213.

Return Values

The **call_close** statement returns an empty string, unless the called test returns an expression using **return** or **textit**.

Availability

This statement is always available.

Note: The **call_close** statement is not a function. Therefore, it does not appear in the Function Generator.

call_ex

Standard • Call Statements

invokes an Astra QuickTest test from within a WinRunner test script.

call_ex (*Astra_test_path*);

Astra_test_path The full pathname of the Astra QuickTest test in quotation marks. Alternatively you may enter a variable that has previously been defined with the full pathname of the test.

The **call_ex** statement invokes an Astra test from with a WinRunner test script. If Astra QuickTest is not installed on the computer running the calling test, the statement returns a "General Error" result.

If the statement does not return a "pass" return value, the calling test fails.

Return Values

The **call_ex** statement returns 1 if the Astra test runs completely and passes, or 0 for any other result.

Availability

This function is always available. If Astra QuickTest is not installed on the computer running the calling test, however, the statement returns a "General Error" result.

check_window

Analog • Bitmap Checkpoint

compares a bitmap of a window to an expected bitmap.

Note: This function is provided for backward compatibility only. You should use the corresponding Context Sensitive **win_check_bitmap** and **obj_check_bitmap** functions.

check_window (*time*, *bitmap*, *window*, *width*, *height*, *x*, *y* [, *relx₁*, *rely₁*, *relx₂*, *rely₂*]);

<i>time</i>	Indicates the interval between the previous input event and the bitmap capture, in seconds. This interval is added to the <i>timeout_msec</i> testing option. The sum is the interval between the previous event and the bitmap capture, in seconds.
<i>bitmap</i>	A string identifying the captured bitmap. The string length is limited to 6 characters.
<i>window</i>	A string indicating the name in the window banner.
<i>width</i> , <i>height</i>	The size of the window, in pixels.
<i>x</i> , <i>y</i>	The position of the upper left corner of the window (relative to the screen). In the case of an MDI child window, the position is relative to the parent window.
<i>relx₁</i> , <i>rely₁</i>	For an area bitmap: the coordinates of the upper left corner of the rectangle, relative to the upper left corner of the client window (the <i>x</i> and <i>y</i> parameters).
<i>relx₂</i> , <i>rely₂</i>	For an area bitmap: the coordinates of the lower right corner of the rectangle, relative to the lower right corner of the client window (the <i>x</i> and <i>y</i> parameters).

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

The **check_window** function is not available for LoadRunner GUI Vusers running on UNIX platforms. In this case, **check_window** statements are treated as **wait_window** statements.

click

Analog • Input Device

inputs a mouse button click.

click (*mouse_button* [, *time*]);

mouse_button The name of the mouse button to be activated. The names (Left, Right, Middle) are defined by the XR_INP_MKEYS system parameter in the system configuration file.

time The interval that elapses before the click is entered, in seconds. The default, if no *time* is specified, is 0.

Return Values

The return value of the function is always 0.

Availability

This function is always available.

click_on_text

Analog • Input Device

clicks on a string.

Note: This function is provided for backward compatibility only. You should use the corresponding Context Sensitive **obj_click_on_text** and **win_click_on_text** functions.

click_on_text (*string*, x_1 , y_1 , x_2 , y_2 [, *click_sequence*]);

string A complete string, preceded and followed by a space outside the quotation marks. A regular expression with no blank spaces can be specified.

x_1, y_1, x_2, y_2 The area of the screen to be searched, specified by the coordinates x_1, y_1, x_2, y_2 , which define any two diagonal corners of a rectangle. The interpreter searches for the text in the area defined by the rectangle.

click_sequence The mouse button clicks that are part of the string's input. The mouse button input is evaluated to a string using the conventions of the click function. (For further details, see the description under click.) The default, if no *click_sequence* is specified, is a single click of the left mouse button.

Return Values

This function returns 0 if the text is located. If the text is not found, the function returns 1.

Availability

This function is always available.

compare_text

Standard • String

compares two strings.

compare_text (*str*₁, *str*₂ [, *chars*₁, *chars*₂]);

<i>str</i> ₁ , <i>str</i> ₂	The two strings to be compared.
<i>chars</i> ₁	One or more characters in the first string.
<i>chars</i> ₂	One or more characters in the second string. These characters are substituted for those in <i>chars</i> ₁ .

Return Values

This function returns the value 1 when the two strings are the same, and 0 when they are different.

Availability

This function is always available.

COS**Standard • Arithmetic**

calculates the cosine of an angle.

`cos (x);`

x Specifies an angle, expressed in radians.

Return Values

This function returns a real number.

Availability

This function is always available.

create_browse_file_dialog**Customization • Custom User Interface**

displays a browse dialog box from which the user selects a file.

`create_browse_file_dialog (filter1 [; filter2; filter3; ...filtern]);`

filter Sets one or more filters for the files to display in the browse dialog box. You must use wildcards to display all files (*.*) or only selected files (*.exe or *.txt, etc.), even if an exact match exists. Multiple files are separated by semicolons and all the filters together are considered a single string.

Return Values

This function returns a string representing the label of the selected file.

Availability

This function is always available.

create_custom_dialog

Customization • Custom User Interface

creates a custom dialog box.

```
create_custom_dialog ( function_name, title, button_name, edit_name_1 [ , edit_name_2,  
                      check_name_1 [ , check_name_2 ] ] );
```

<i>function_name</i>	The name of the function that is executed when you press the "execute" button.
<i>title</i>	An expression that appears in the window banner of the dialog box.
<i>button_name</i>	The label that will appear on the "execute" button. You press this button to execute the contained function.
<i>edit_name</i>	The labels of the edit box(es) of the dialog box. Multiple edit box labels are separated by commas, and all the labels together are considered a single string. If the dialog box has no edit boxes, this parameter must be an empty string (empty quotation marks).
<i>check_name</i>	Contains the labels of the check boxes in the dialog box. Multiple check box labels are separated by commas, and all the labels together are considered a single string. If the dialog box has no check boxes, this parameter must be an empty string (empty quotation marks).

Return Values

This function returns a string representing the return value of the function executed when the **Execute** button is clicked and an empty string is returned when the **Cancel** button is clicked.

Availability

This function is always available.

create_input_dialog

Customization • Custom User Interface

creates a dialog box with an edit box.

```
create_input_dialog ( message );
```

<i>message</i>	Any expression. This expression will appear in the dialog box as a single line.
----------------	---

Return Values

This function returns a string. If no string is found or if the Cancel button is pressed within the dialog box, then the function returns NULL.

Availability

This function is always available.

create_list_dialog

Customization • Custom User Interface

creates a dialog box with a list of items.

```
create_list_dialog ( title, message, item_list );
```

<i>title</i>	The expression that appears in the banner of the dialog box.
--------------	--

<i>message</i>	The message for the user.
----------------	---------------------------

<i>item_list</i>	The items that make up the list, separated by commas.
------------------	---

Return Values

This function returns a string. If no string is found or if the Cancel button is pressed within the dialog box, then this function returns NULL.

Availability

This function is always available.

create_password_dialog

Customization • Custom User Interface

creates a password dialog box.

```
create_password_dialog ( login, password, login_out, password_out  
                        [ , encrypt_password ] );
```

<i>login</i>	The label of the first edit box, used for user-name input. If you specify an empty string (empty quotation marks), the default label "Login" is displayed.
<i>password</i>	The label of the second edit box, used for password input. If you specify an empty string (empty quotation marks), the default label "Password" is displayed. When the user enters input into this edit box, the characters do not appear on the screen, but are represented by asterisks.
<i>login_out</i>	The name of the parameter to which the contents of the first edit box (<i>login</i>) are passed. Use this parameter to verify the contents of the login edit box.
<i>password_out</i>	The name of the parameter to which the contents of the second edit box (<i>password</i>) are passed. Use this parameter to verify the contents of the password edit box.
<i>encrypt_password</i>	A Boolean parameter which allows the output edit field value to be encrypted. If this parameter is left blank, the default value is FALSE.

Return Values

This function returns the number “1” if the **OK** button is pressed and “0” if the **Cancel** button is pressed.

Availability

This function is always available.

datawindow_get_info

Context Sensitive • PowerBuilder

retrieves the value of a DataWindow object property.

datawindow_get_info (*DataWindow_object*, *property*, *out_value*);

<i>DataWindow_object</i>	The logical name of the DataWindow object.
<i>property</i>	The full property description (similar to the formats in the PowerBuilder Describe function, e.g. obj.property...).
<i>out_value</i>	The output variable that stores the value of the specified property (maximum size 2,000 characters).

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118 and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available whenever the PowerBuilder add-in is loaded.

datawindow_text_click

Context Sensitive • PowerBuilder

clicks a DataWindow text object.

datawindow_text_click (*DataWindow_object*, *DataWindow_text_object*);

<i>DataWindow_object</i>	The logical name of the DataWindow object.
<i>DataWindow_text_object</i>	The text property of the DataWindow object (and NOT the internal PowerBuilder name).

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118 and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available whenever the PowerBuilder add-in is loaded.

datawindow_text_dbl_click

Context Sensitive • PowerBuilder

double-clicks a DataWindow text object.

datawindow_text_dbl_click (*DataWindow_object*, *DataWindow_text_object*);

DataWindow_object The logical name of the DataWindow object.

DataWindow_text_object The text property of the DataWindow object (and NOT the internal PowerBuilder name).

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118 and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available whenever the PowerBuilder add-in is loaded.

db_check

Context Sensitive • Database

compares current database data to expected database data.

db_check (*checklist*, *expected_results_file* [, *max_rows* [, *paramater_array*]]);

checklist The name of the checklist specifying the checks to perform.

expected_results_file The name of the file storing the expected database data.

max_rows The maximum number of rows retrieved in a database. If no maximum is specified, then by default the number of rows is not limited. If you change this parameter in a **db_check** statement recorded in your test script, you must run the test in Update mode before you run it in Verify mode.

paramater_array The array of parameters for the SQL statement. For information on working with this advanced feature, refer to the “Checking Databases” chapter in the *WinRunner User’s Guide*.

Return Values

This function returns 1 for a successful bitmap capture or comparison. Otherwise, this function returns 0. For more information, see “General Return Values,” on page 118 and “Return Values for Database Functions,” on page 122.

Availability

This function is always available.

db_connect

Context Sensitive • Database

creates a new database session and establishes a connection to an ODBC database.

db_connect (*session_name*, *connection_string*);

session_name The logical name of the database session.

connection_string The connection parameters to the ODBC database.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118 and “Return Values for Database Functions,” on page 122.

Availability

This function is always available.

db_disconnect

Context Sensitive • Database

disconnects from the database and ends the database session.

db_disconnect (*session_name*);

session_name The logical name of the database session.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118 and “Return Values for Database Functions,” on page 122.

Availability

This function is always available.

db_dj_convert

Context Sensitive • Database

runs a Data Junction export file (*.djs file).

db_dj_convert (*djs_file* [, *output_file* [, *headers* [, *record_limit*]]]);

djs_file The Data Junction export file.

output_file An optional parameter to override the name of the target file.

headers An optional Boolean parameter that will include or exclude the column headers from the Data Junction export file.

record_limit The maximum number of records that will be converted.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118 and “Return Values for Database Functions,” on page 122.

Availability

This function is only available for users working with Data Junction.

db_execute_query

Context Sensitive • Database

executes the query based on the SQL statement and creates a record set.

db_execute_query (*session_name*, *SQL*, *record_number*);

<i>session_name</i>	The logical name of the database session.
<i>SQL</i>	The SQL statement.
<i>record_number</i>	An out parameter returning the number of records in the result query.

For information on this advanced feature, refer to the “Checking Databases” chapter in the *WinRunner User’s Guide*.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118 and “Return Values for Database Functions,” on page 122.

Availability

This function is always available.

db_get_field_value

Context Sensitive • Database

returns the value of a single field in the database.

db_get_field_value (*session_name*, *row_index*, *column*);

<i>session_name</i>	The logical name of the database session.
<i>row_index</i>	The numeric index of the row. (The first row is always numbered “#0”.)
<i>column</i>	The name of the field in the column or the numeric index of the column within the database. (The first column is always numbered “#0”.)

Return Values

In case of an error, an empty string will be returned. For more information, see “General Return Values,” on page 118 and “Return Values for Database Functions,” on page 122.

Availability

This function is always available.

db_get_headers

Context Sensitive • Database

returns the number of column headers in a query and the content of the column headers, concatenated and delimited by tabs.

db_get_headers (*session_name*, *header_count*, *header_content*);

<i>session_name</i>	The logical name of the database session.
<i>header_count</i>	The number of column headers in the query.
<i>header_content</i>	The column headers concatenated and delimited by tabs. Note that if this string exceeds 1024 characters, it is truncated.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118 and “Return Values for Database Functions,” on page 122.

Availability

This function is always available.

db_get_last_error

Context Sensitive • Database

returns the last error message of the last ODBC or Data Junction operation.

db_get_last_error (*session_name*, *error*);

<i>session_name</i>	The logical name of the database session.
<i>error</i>	The error message.

Note: When working with Data Junction, the *session_name* parameter is ignored.

Return Values

If there is no error message, an empty string will be returned.

Availability

This function is always available.

db_get_row

Context Sensitive • Database

returns the content of the row, concatenated and delimited by tabs.

db_get_row (*session_name*, *row_index*, *row_content*);

<i>session_name</i>	The logical name of the database session.
<i>row_index</i>	The numeric index of the row. (The first row is always numbered “0”.)
<i>row_content</i>	The row content as a concatenation of the fields values, delimited by tabs.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118 and “Return Values for Database Functions,” on page 122.

Availability

This function is always available.

db_record_check

Context Sensitive • Database

compares information that appears in the application under test during a test run with the current values in the corresponding record(s) in your database. You insert `db_record_check` statements by using the Runtime Record Verification wizard. For more information, refer to the *WinRunner User's Guide*.

db_record_check (*ChecklistFileName* , *SuccessConditions*, *RecordNumber*);

<i>ChecklistFileName</i>	A file created by WinRunner and saved in the test's checklist folder. The file contains information about the data to be captured during the test run and its corresponding field in the database. The file is created based on the information entered in the Runtime Record Verification wizard.
<i>SuccessConditions</i>	Contains one of the following values: DVR_ONE_OR_MORE_MATCH - The checkpoint passes if one or more matching database records are found. DVR_ONE_MATCH - The checkpoint passes if exactly one matching database record is found. DVR_NO_MATCH - The checkpoint passes if no matching database records are found.
<i>RecordNumber</i>	An out parameter returning the number of records in the database.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118 and "Return Values for Database Functions," on page 122.

Availability

This function is always available.

db_write_records

Context Sensitive • Database

writes the record set into a text file delimited by tabs.

db_write_records (*session_name*, *output_file* [, *headers* [, *record_limit*]]);

<i>session_name</i>	The logical name of the database session.
<i>output_file</i>	The name of the text file in which the record set is written.
<i>headers</i>	An optional Boolean parameter that will include or exclude the column headers from the record set written into the text file.
<i>record_limit</i>	The maximum number of records in the record set to be written into the text file. A value of NO_LIMIT (the default value) indicates there is no maximum limit to the number of records in the record set.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118 and “Return Values for Database Functions,” on page 122.

Availability

This function is always available.

dbl_click

Analog • Input Device

double-clicks a mouse button.

dbl_click (*mouse_button* [, *time*]);

<i>mouse_button</i>	The mouse button to activate. The names ("Left," "Right," "Middle") are defined by the XR_INP_MKEYS system parameter in the system configuration file.
<i>time</i>	The interval that elapses before the click is entered, in seconds. The default, if no <i>time</i> is specified, is 0.

Return Values

This function always returns 0.

Availability

This function is always available.

ddt_close

Context Sensitive • Data-Driven Test

closes a data table file.

ddt_close (*data_table_name*);

<i>data_table_name</i>	The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters.
------------------------	--

Note: **ddt_close** does NOT save changes to the data table. If you make any changes to the data table, you must use the **ddt_save** function to save your changes before using **ddt_close** to close the table.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

ddt_close_all_tables

Context Sensitive • Data-Driven Test

closes all open tables in all open tests.

```
ddt_close_all_tables();
```

Note: This close function includes any tables that are open in the table editor, tables that were opened using the **ddt_open** or **ddt_show** functions or using the DataDriven Tests Wizard.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

ddt_export

Context Sensitive • Data-Driven Test

exports the information of one data table file into a different data table file.

```
ddt_export ( data_table_filename1, data_table_filename2 );
```

data_table_filename₁ The source data table filename.

data_table_filename₂ The destination data table filename.

Note: You must use a **ddt_open** statement to open the source data table before you can use any other **ddt_** functions.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

ddt_get_current_row**Context Sensitive • Data-Driven Test**

retrieves the active row of a data table.

```
ddt_get_current_row ( data_table_name, out_row );
```

<i>data_table_name</i>	The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters. This row is labeled row 0.
<i>out_row</i>	The output variable that stores the active row in the data table.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

ddt_get_parameters**Context Sensitive • Data-Driven Test**

returns a list of all parameters in a data table.

```
ddt_get_parameters ( table, params_list, params_num );
```

<i>table</i>	The pathname of the data table.
<i>params_list</i>	This out parameter returns the list of all parameters in the data table, separated by tabs.
<i>params_num</i>	This out parameter returns the number of parameters in params_list.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

ddt_get_row_count**Context Sensitive • Data-Driven Test**

retrieves the number of rows in a data table.

ddt_get_row_count (*data_table_name*, *out_rows_count*);

data_table_name The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters.

out_rows_count The output variable that stores the total number of rows in the data table.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

ddt_is_parameter

Context Sensitive • Data-Driven Test

returns whether a parameter in a data table is valid.

ddt_is_parameter (*data_table_name*, *parameter*);

<i>data_table_name</i>	The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table.
<i>parameter</i>	The parameter name to check in the data table.

Return Values

This functions returns TRUE when rc=0. The function returns FALSE in all other cases.

Availability

This function is always available.

ddt_next_row

Context Sensitive • Data-Driven Test

changes the active row in a data table to the next row.

ddt_next_row (*data_table_name*);

<i>data_table_name</i>	The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters.
------------------------	--

Return Values

If the active row is the last row in a data table, then the E_OUT_OF_RANGE value is returned.

Availability

This function is always available.

ddt_open

Context Sensitive • Data-Driven Test

creates or opens a data table file so that WinRunner can access it.

ddt_open (*data_table_name*, *mode*);

<i>data_table_name</i>	The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters. This row is labeled row 0.
<i>mode</i>	The mode for opening the data table: DDT_MODE_READ (read-only) or DDT_MODE_READWRITE (read or write).

Note: If you make any changes to the data table, you must use the **ddt_save** function to save your changes before using **ddt_close** to close the table.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

ddt_report_row

Context Sensitive • Data-Driven Test

reports the active row in a data table to the test results.

ddt_report_row (*data_table_name*);

<i>data_table_name</i>	The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters. This row is labeled row 0.
------------------------	---

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

ddt_save

Context Sensitive • Data-Driven Test

saves the information in a data table.

ddt_save (*data_table_name*);

<i>data_table_name</i>	The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table.
------------------------	--

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

ddt_set_row

Context Sensitive • Data-Driven Test

sets the active row in a data table.

ddt_set_row (*data_table_name*, *row*);

<i>data_table_name</i>	The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters. This row is labeled row 0.
------------------------	---

<i>row</i>	The new active row in the data table.
------------	---------------------------------------

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

ddt_set_val

Context Sensitive • Data-Driven Test

sets a value in the current row of the data table.

ddt_set_val (*data_table_name*, *parameter*, *value*);

<i>data_table_name</i>	The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. This row is labeled row 0.
<i>parameter</i>	The name of the column into which the value will be inserted.
<i>value</i>	The value to be written into the table.

Notes:

You can only use this function if the data table was opened in DDT_MODE_READWRITE (read or write mode).

To save the new or modified contents of the table, add a **ddt_save** statement after the **ddt_set_val** statement. At the end of your test, use a **ddt_close** statement to close the table.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

ddt_set_val_by_row**Context Sensitive • Data-Driven Test**

sets a value in a specified row of the data table.

ddt_set_val_by_row (*data_table_name*, *row*, *parameter*, *value*);

<i>data_table_name</i>	The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters. This row is labeled row 0.
<i>row</i>	The row number in the table.
<i>parameter</i>	The name of the column into which the value will be inserted.
<i>value</i>	The value to be written into the table.

Notes:

You can only use this function if the data table was opened in DDT_MODE_READWRITE (read or write mode).

To save the new or modified contents of the table, add a **ddt_save** statement after the **ddt_set_val** statement. At the end of your test, use a **ddt_close** statement to close the table.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

ddt_show**Context Sensitive • Data-Driven Test**

shows or hides the table editor of a specified data table.

ddt_show (*data_table_name*, *show_flag*);

<i>data_table_name</i>	The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table.
<i>show_flag</i>	The value indicating whether the editor is to be shown. The <i>show_flag</i> value is 1 if the table editor is to be shown and is 0 if the table editor is to be hidden.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

ddt_update_from_db**Context Sensitive • Data-Driven Test**

imports data from a database into a data table.

ddt_update_from_db (*data_table_name*, *file*, *out_row_count* [, *max_rows*]);

<i>data_table_name</i>	The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table.
<i>file</i>	Either an <i>*.sql</i> file containing an ODBC query or a <i>*.djs</i> file containing a conversion defined by Data Junction.
<i>out_row_count</i>	An out parameter containing the number of rows retrieved from the data table.
<i>max_rows</i>	An in parameter specifying the maximum number of rows to be retrieved from a database. If no maximum is specified, then by default the number of rows is not limited.

Note: You must use a **ddt_open** statement to open the data table in READWRITE mode before you can use this function.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

ddt_val

Context Sensitive • Data-Driven Test

returns the value of a parameter in the active row in a data table.

ddt_val (*data_table_name*, *parameter*);

data_table_name The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters.

parameter The name of the parameter in the data table.

Return Values

This functions returns the value of a parameter in the active row in a data table.

In the case of an error, this function returns an empty string.

Availability

This function is always available.

ddt_val_by_row

Context Sensitive • Data-Driven Test

returns the value of a parameter in the specified row in a data table.

ddt_val_by_row (*data_table_name*, *row_number*, *parameter*);

data_table_name The name of the data table. The name may be the table variable name, the Microsoft Excel file or a tabbed text file name, or the full path and file name of the table. The first row in the file contains the names of the parameters. This row is labeled row 0.

row_number The number of the row in the data table.

parameter The name of the parameter in the data table.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

declare_rendezvous

Standard • Load Testing

declares a rendezvous.

declare_rendezvous (*rendezvous_name*);

rendezvous_name The name of the rendezvous. This must be a string constant and not a variable or an expression. The *rendezvous_name* can be a maximum of 128 characters. It cannot contain any spaces.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for LoadRunner GUI Vusers only.

declare_transaction

Standard • Load Testing

declares a transaction.

declare_transaction (*transaction_name*);

<i>transaction_name</i>	The name of the transaction. This must be a string constant and not a variable or an expression. The <i>transaction_name</i> can be a maximum of 128 characters. It cannot contain any spaces. The first character cannot be number.
-------------------------	--

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for LoadRunner GUI Vusers only.

define_object_exception

Standard • Exception Handling

defines an object exception.

define_object_exception (*exception_name*, *handler*, *window*, *object*, *property* [, *value*]);

<i>exception_name</i>	The name of the exception.
<i>handler</i>	The name of the handler function.
<i>window</i>	The logical name of the window.
<i>object</i>	The logical name of the object.
<i>property</i>	An object property.
<i>value</i>	The value of the object property to detect.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

define_popup_exception**Standard • Exception Handling**

defines a popup exception.

define_popup_exception (*exception_name*, *handler*, *window*);

exception_name The name of the exception.

handler The name of the handler function. The handler can be a built-in handler or a user-defined handler. For a list of built-in handlers, see below.

window The name of the popup window.

Built-In Handler Function	Description
<code>win_press_cancel</code>	Clicks the Cancel button in the window
<code>win_press_ok</code>	Clicks the OK button in the window
<code>win_press_return</code>	Presses the Return key (the equivalent of clicking the default button in the window)

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

define_tsl_exception

Standard • Exception Handling

defines a TSL exception.

```
define_tsl_exception ( exception_name, handler, return_code [ , function ] );
```

<i>exception_name</i>	The name of the exception.
<i>handler</i>	The name of the handler function.
<i>return_code</i>	The return code to detect. To detect any return code with a value less than zero, you can set E_ANY_ERROR as the argument.
<i>function</i>	The TSL function to monitor.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

delete

Standard • Array

removes an element from an array.

```
delete array [ subscript ];
```

<i>array</i>	The array from which the element is deleted.
<i>subscript</i>	An expression that specifies the subscript of the array element to delete.

Return Values

This function always returns an empty string.

Availability

This function is always available.

delete_record_attr

Customization • Custom Record

removes a custom property that was registered using **add_record_attr**.

```
delete_record_attr ( attr_name [ , dll_name, query_func_name, verify_func_name ] );
```

<i>attr_name</i>	The name of the custom property to remove. Note that you cannot remove any standard WinRunner properties.
<i>dll_name</i>	The full path of the DLL (Dynamic Link Library) in which the query and verify functions are defined.
<i>query_func_name</i>	The name of the user-defined query function that was called by the add_record_attr statement which registered the custom property.
<i>verify_func_name</i>	The name of the verify function that was called by the add_record_attr statement which registered the custom property (either a WinRunner standard property verification function or a custom property verification function included in the DLL).

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

dlph_edit_set

Context Sensitive • Delphi

replaces the entire contents of a Delphi edit object.

dlph_edit_set (*edit*, *text*);

edit The logical name of the Delphi edit object.

text The new contents of the Delphi edit object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available only for WinRunner with Delphi support.

dlph_list_select_item

Context Sensitive • Delphi

selects a Delphi list item.

dlph_list_select_item (*list*, *item*);

list The logical name of the Delphi list.

item The item to select in the Delphi list.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available only for WinRunner with Delphi support.

dlph_obj_get_info

Context Sensitive • Delphi

retrieves the value of a Delphi object.

dlph_obj_get_info (*name*, *property*, *out_value*);

<i>name</i>	The logical name of the Delphi object.
<i>property</i>	Any property associated with the Delphi object.
<i>out_value</i>	The value of the property.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available only for WinRunner with Delphi support.

dlph_obj_set_info

Context Sensitive • Delphi

sets the value of a Delphi object.

dlph_obj_set_info (*name*, *property*, *in_value*);

<i>name</i>	The logical name of the Delphi object.
<i>property</i>	Any property associated with the Delphi object.
<i>in_value</i>	The new value of the Delphi property.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available only for WinRunner with Delphi support.

dolph_panel_button_press

Context Sensitive • Delphi

clicks a button within a Delphi panel.

dolph_panel_button_press (*panel*, *button*, *x*, *y*);

<i>panel</i>	The object.
<i>button</i>	The Delphi name.
<i>x</i> , <i>y</i>	The location that is pressed on the button, expressed as x and y (pixel) coordinates, relative to the top left corner of the button.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available only for WinRunner with Delphi support.

dos_system

Standard • Operating System

executes a DOS system command from within a WinRunner test script.

dos_system (*expression*);

expression A string expression specifying the system command to be executed.

Note: When using MS-DOS Prompt (Windows 95/98), or command.com (Windows NT), then the expression in `dos_system` is limited to 127 characters. When using Command Prompt (Windows NT), the expression can hold more characters.

If the limitation is problematic, try to use shorter commands and split long commands into shorter ones. For example, if you want to copy file1 to file2 and both files have very long names, instead of using `dos_system("copy file1 file2")` use a third file with a shorter name (e.g. tmpfile) in the following commands:

```
dos_system("copy file1 tmpfile") ;  
dos_system("copy tmpfile file2") ;
```

Return Values

The return value of the function is the return value of the DOS system command that was executed.

Availability

This function is available for WinRunner and LoadRunner GUI Vusers running on PC platforms only. To execute Windows executables, use **invoke_application**. To execute UNIX system commands, use **system**. To execute OS2 commands, use **os2_system**.

edit_activate

Context Sensitive • Oracle

double-clicks an object in an Oracle application.

edit_activate (*object*);

object The logical name of the object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_check_info

Context Sensitive • Edit Object

checks the value of an edit object property.

edit_check_info (*edit*, *property*, *property_value*);

edit The logical name of the edit object.

property The property to check.

property_value The property value.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_check_selection

Context Sensitive • Edit Object

checks that a string is selected.

edit_check_selection (*edit*, *selected_string*);

<i>edit</i>	The logical name of the edit object.
<i>selected_string</i>	The selected string. The string is limited to 256 characters. It cannot be evaluated automatically when used with the Function Generator.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_check_text

Context Sensitive • Edit Object

checks the contents of an edit object.

edit_check_text (*edit*, *text*, *case_sensitive*);

<i>edit</i>	The logical name of the edit object.
<i>text</i>	The contents of the edit object (up to 256 characters).
<i>case_sensitive</i>	Indicates whether the comparison is case sensitive. This value is either TRUE or FALSE.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_delete

Context Sensitive • Edit Object

deletes the contents of an edit object.

edit_delete (*edit*, *start_column*, *end_column*);

<i>edit</i>	The logical name of the edit object.
<i>start_column</i>	The column at which the text starts.
<i>end_column</i>	The column at which the text ends. Note that if this is greater than the last column of the first line, then part of the following line will also be deleted.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_delete_block

Context Sensitive • Edit Object

deletes a text block from an edit object.

edit_delete_block (*edit*, *start_row*, *start_column*, *end_row*, *end_column*);

<i>edit</i>	The logical name of the edit object.
<i>start_row</i>	The row at which the text block starts.
<i>start_column</i>	The column at which the text block starts.
<i>end_row</i>	The row at which the text block ends.
<i>end_column</i>	The column at which the text block ends.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_get_block**Context Sensitive • Edit Object**

returns block of text in an edit object.

edit_get_block (*edit*, *start_row*, *start_column*, *end_row*, *end_column*, *out_string*);

<i>edit</i>	The logical name of the edit object.
<i>start_row</i>	The row at which the text block starts.
<i>start_column</i>	The column at which the text block starts.
<i>end_row</i>	The row at which the text block ends.
<i>end_column</i>	The column at which the text block ends.
<i>out_string</i>	The output variable that stores the text string.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_get_info

Context Sensitive • Edit Object

returns the value of an edit object property.

edit_get_info (*edit*, *property*, *out_value*);

<i>edit</i>	The logical name of the edit object.
<i>property</i>	Any of the properties listed in the <i>User's Guide</i> .
<i>out_value</i>	The output variable that stores the value of the specified property.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_get_row_length

Context Sensitive • Edit Object

returns the length of a row in an edit object.

edit_get_row_length (*edit*, *row*, *out_length*);

<i>edit</i>	The logical name of the edit object.
<i>row</i>	The row to measure.
<i>out_length</i>	The output variable that stores the number of characters in the row.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_get_rows_count

Context Sensitive • Edit Object

returns the number of rows written in an edit object.

edit_get_rows_count (*edit*, *out_number*);

<i>edit</i>	The logical name of the edit object.
<i>out_number</i>	The output variable that stores the number of rows written in the edit object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_get_selection

Context Sensitive • Edit Object

returns the selected string in an edit object.

edit_get_selection (*edit*, *out_string*);

<i>edit</i>	The logical name of the edit object.
<i>out_string</i>	The output variable that stores the selected string. The string is limited to 256 characters. It cannot be evaluated automatically when used with the Function Generator.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_get_selection_pos

Context Sensitive • Edit Object

returns the position at which the selected block starts and ends.

```
edit_get_selection_pos ( edit, out_start_row, out_start_column, out_end_row,  
                        out_end_column );
```

<i>edit</i>	The logical name of the edit object.
<i>out_start_row</i>	The output variable which stores the row at which the selected block starts.
<i>out_start_column</i>	The output variable which stores the column at which the selected block starts.
<i>out_end_row</i>	The output variable which stores the row at which the selected block ends.
<i>out_end_column</i>	The output variable which stores the column at which the selected block ends.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_get_text

Context Sensitive • Edit Object

returns the text in an edit object.

edit_get_text (*edit*, *out_string*);

<i>edit</i>	The logical name of the edit object.
<i>out_string</i>	The output variable that stores the string found in the edit object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_insert

Context Sensitive • Edit Object

inserts text in the first line of an edit object.

edit_insert (*edit*, *text*, *column*);

<i>edit</i>	The logical name of the edit object.
<i>text</i>	The text to be inserted in the edit object.
<i>column</i>	The column at which the insertion is made.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_insert_block

Context Sensitive • Edit Object

inserts text in a multi-line edit object.

edit_insert_block (*edit*, *text*, *row*, *column*);

<i>edit</i>	The logical name of the edit object.
<i>text</i>	The text to be inserted in the edit object.
<i>row</i>	The row at which the insertion is made.
<i>column</i>	The column at which the insertion is made.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_replace

Context Sensitive • Edit Object

replaces the contents of an edit object.

edit_replace (*edit*, *text*, *start_column*, *end_column*);

<i>edit</i>	The logical name of the edit object.
<i>text</i>	The new contents of the edit object.
<i>start_column</i>	The column at which the text block starts.
<i>end_column</i>	The column at which the text block ends.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_replace_block

Context Sensitive • Edit Object

replaces a block of text in an edit object.

edit_replace_block (*edit*, *text*, *start_row*, *start_column*, *end_row*, *end_column*);

<i>edit</i>	The logical name of the edit object.
<i>text</i>	The new contents of the edit object.
<i>start_row</i>	The row at which the text block starts.
<i>start_column</i>	The column at which the text block starts.
<i>end_row</i>	The row at which the text block ends.
<i>end_column</i>	The column at which the text block ends.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_set

Context Sensitive • Edit Object

replaces the entire contents of an edit object.

edit_set (*edit*, *text*);

<i>edit</i>	The logical name of the edit object.
<i>text</i>	The new contents of the edit object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_set_focus

Context Sensitive • Edit Object

focuses on an object in an Oracle application.

edit_set_focus (*object*);

object The logical name of the object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_set_insert_pos

Context Sensitive • Edit Object

places the cursor at a specified point in an edit object.

edit_set_insert_pos (*edit*, *row*, *column*);

edit The logical name of the edit object.

row The row position at which the insertion point is placed.

column The column position at which the insertion point is placed.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_set_selection

Context Sensitive • Edit Object

selects text in an edit object.

edit_set_selection (*edit*, *start_row*, *start_column*, *end_row*, *end_column*);

<i>edit</i>	The logical name of the edit object.
<i>start_row</i>	The row at which the selection starts.
<i>start_column</i>	The column at which the selection starts.
<i>end_row</i>	The row at which the selection ends.
<i>end_column</i>	The column at which the selection ends.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_type

Context Sensitive • Edit Object

types a string in an edit object.

edit_type (*edit*, *text*);

<i>edit</i>	The logical name of the edit object.
<i>text</i>	The string to type into the edit object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

edit_wait_info

Context Sensitive • Edit Object

waits for the value of an edit object property.

edit_wait_info (*edit*, *property*, *value*, *time*);

<i>edit</i>	The logical name of the edit object.
<i>property</i>	Any of the properties listed in the <i>WinRunner User's Guide</i> .
<i>value</i>	The property value.
<i>time</i>	The maximum amount of time the test will wait before resuming execution.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

end_transaction

Standard • Load Testing

marks the end of a transaction for performance analysis.

end_transaction (*transaction* [, *status*]);

<i>transaction</i>	A string, with no spaces, naming the transaction.
<i>status</i>	The status of the transaction: pass (0), or fail (any non-zero value). If no value is specified, the default value is pass.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for LoadRunner GUI Users only.

error_message**Standard • Load Testing**

sends an error message to the controller.

error_message (*message*);

message Any string.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for LoadRunner GUI Vusers only.

EURO_check_currency**Context Sensitive • EURO**

captures and compares the currencies in a window.

EURO_check_currency (*file_name*, *x₁*, *y₁*, *x₂*, *y₂*);

file_name The file containing the expected results of the EURO checkpoint.

x₁, *y₁* The position of the upper left corner of the area to be checked.

x₂, *y₂* The position of the lower right corner of the area to be checked.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner EURO only.

EURO_compare_columns

Context Sensitive • EURO

compares two currency columns (dual display) and returns the number of mismatches.

EURO_compare_columns (*check_name*, *column₁_field₁*, *column₁_field_n*, *column₂_field₁*, *column₂_field_n*);

<i>check_name</i>	The file name that stores the data.
<i>column₁_field₁</i>	The first column first field to be included in the comparison.
<i>column₁_field_n</i>	The first column last field to be included in the comparison.
<i>column₂_field₁</i>	The second column first field to be included in the comparison.
<i>column₂_field_n</i>	The second column last field to be included in the comparison.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner EURO only.

EURO_compare_fields

Context Sensitive • EURO

compares two fields while converting.

EURO_compare_fields (*field₁*, *field₂*, *currency₁*, *currency₂*, *align_mode*, *align_value*);

<i>field₁</i>	The name of the first field.
<i>field₂</i>	The name of the second field.
<i>currency₁</i>	The country whose currency you want to compare to <i>currency_2</i> . One of the following countries can be specified: Austria, Belgium, Finland, France, Germany, Ireland, Italy, Luxembourg, Netherlands, Portugal, Spain, Great Britain, Denmark, Greece, Sweden, and EURO.
<i>currency₂</i>	The country whose currency is compared to <i>currency_1</i> . One of the following countries can be specified: Austria, Belgium, Finland, France, Germany, Ireland, Italy, Luxembourg, Netherlands, Portugal, Spain, Great Britain, Denmark, Greece, Sweden, and EURO.
<i>align_mode</i>	One of the following modes can be specified: ALIGN_NONE: No currency alignment ALIGN_ROUND: Rounds the converted currency to the nearest multiple specified in <i>align_value</i> . ALIGN_SUFFIX_DOWN: Rounds down the converted currency value to end with the suffix value indicated in <i>align_value</i> . ALIGN_SUFFIX_UP: Rounds up the converted currency value to end with the suffix value indicated in <i>align_value</i> . ALIGN_TRUNC: Rounds the converted currency value down to the nearest unit.
<i>align_value</i>	The value to align the currency.

Return Values

The `EURO_compare_fields` function returns `E_OK` or `E_DIFF`.

Availability

This function is available for WinRunner EURO only.

EURO_compare_numbers

Context Sensitive • EURO

compares two numbers while converting.

`EURO_compare_numbers (number1, number2, currency1, currency2, align_mode, align_value);`

<i>number₁</i>	The first number to compare.
<i>number₂</i>	The second number to compare.
<i>currency₁</i>	The country whose currency you want to compare to <i>currency_2</i> One of the following countries can be specified: Austria, Belgium, Finland, France, Germany, Ireland, Italy, Luxembourg, Netherlands, Portugal, Spain, Great Britain, Denmark, Greece, Sweden, and EURO.
<i>currency₂</i>	The country whose currency is compared to <i>currency_1</i> . One of the following countries can be specified: Austria, Belgium, Finland, France, Germany, Ireland, Italy, Luxembourg, Netherlands, Portugal, Spain, Great Britain, Denmark, Greece, Sweden, and EURO.

align_mode

One of the following modes can be specified:

ALIGN_NONE: No currency alignment

ALIGN_ROUND: Rounds the converted currency to the nearest multiple specified in *align_value*.

ALIGN_SUFFIX_DOWN: Rounds down the converted currency value to end with the suffix value indicated in *align_value*.

ALIGN_SUFFIX_UP: Rounds up the converted currency value to end with the suffix value indicated in *align_value*.

ALIGN_TRUNC: Rounds the converted currency value down to the nearest unit.

align_value

The value to align the currency.

Return Values

The `EURO_compare_numbers` function returns `E_OK` or `E_DIFF`.

Availability

This function is available for WinRunner EURO only.

EURO_convert_currency

Context Sensitive • EURO

returns the converted currency value between two currencies.

EURO_convert_currency (*number*, *original_currency*, *new_currency*, *align_mode*, *align_value*);

<i>number</i>	The amount of currency to be converted.
<i>original_currency</i>	The country from whose currency you want to compute its value in the <i>new_currency</i> . One of the following countries can be specified: Austria, Belgium, Finland, France, Germany, Ireland, Italy, Luxembourg, Netherlands, Portugal, Spain, Great Britain, Denmark, Greece, Sweden, and EURO.
<i>new_currency</i>	The country to whose currency the <i>original_currency</i> is being computed. One of the following countries can be specified: Austria, Belgium, Finland, France, Germany, Ireland, Italy, Luxembourg, Netherlands, Portugal, Spain, Great Britain, Denmark, Greece, Sweden, and EURO.
<i>align_mode</i>	One of the following modes can be specified: ALIGN_NONE: No currency alignment ALIGN_ROUND: Rounds the converted currency to the nearest multiple specified in <i>align_value</i> . ALIGN_SUFFIX_DOWN: Rounds down the converted currency value to end with the suffix value indicated in <i>align_value</i> . ALIGN_SUFFIX_UP: Rounds up the converted currency value to end with the suffix value indicated in <i>align_value</i> . ALIGN_TRUNC: Rounds the converted currency value down to the nearest unit.
<i>align_value</i>	The value to align the currency.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner EURO only.

EURO_override_field**Context Sensitive • EURO**

overrides the original currency in a field to a new currency.

EURO_override_field (*field_name*, *original_currency*, *new_currency*, *align_mode*, *align_value*);

<i>field_name</i>	The name of the field in which you want to override the currency.
<i>original_currency</i>	The country from whose currency you want to override to <i>new_currency</i> . One of the following countries can be specified: Austria, Belgium, Finland, France, Germany, Ireland, Italy, Luxembourg, Netherlands, Portugal, Spain, Great Britain, Denmark, Greece, Sweden, and EURO.
<i>new_currency</i>	The country to whose currency the <i>original_currency</i> is being overridden. One of the following countries can be specified: Austria, Belgium, Finland, France, Germany, Ireland, Italy, Luxembourg, Netherlands, Portugal, Spain, Great Britain, Denmark, Greece, Sweden, and EURO.

align_mode

One of the following modes can be specified:

ALIGN_NONE: No currency alignment

ALIGN_ROUND: Rounds the converted currency to the nearest multiple specified in *align_value*.

ALIGN_SUFFIX_DOWN: Rounds down the converted currency value to end with the suffix value indicated in *align_value*.

ALIGN_SUFFIX_UP: Rounds up the converted currency value to end with the suffix value indicated in *align_value*.

ALIGN_TRUNC: Rounds the converted currency value down to the nearest unit.

align_value

The value to align the currency.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner EURO only.

EURO_set_auto_currency_verify

Context Sensitive • EURO

activates/deactivates automatic EURO verification.

EURO_set_auto_currency_verify (*mode*);

mode

The mode can be set to ON or OFF.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner EURO only.

EURO_set_capture_mode

Context Sensitive • EURO

determines how WinRunner EURO captures currency in terminal emulator applications.

EURO_set_capture_mode (*capture_mode*);

capture_mode

The currency capture mode. One of the following modes can be specified:

FIELD_METHOD: Captures currencies in the context of the screens and fields in your terminal emulator application (Context Sensitive). This is the default mode.

POSITION_METHOD: Identifies and captures currencies according to the unformatted view of the screen.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner EURO only.

EURO_set_conversion_mode

Context Sensitive • EURO

sets the EURO conversion run mode in the test script.

EURO_set_conversion_mode (*conversion_mode*);

conversion_mode

The EURO conversion run mode. One of the following modes can be specified:

NO_CHANGE: No change is made to objects containing numeric values during the test run.

CONVERT: Performs EURO conversion during the test run.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner EURO only.

EURO_set_conversion_rate

Context Sensitive • EURO

sets the conversion rate between the EURO currency and a national currency.

EURO_set_conversion_rate (*currency*, *rate*);

currency The country whose currency rate you want to set. One of the following countries can be specified: Austria, Belgium, Finland, France, Germany, Ireland, Italy, Luxembourg, Netherlands, Portugal, Spain, Great Britain, Denmark, Greece, Sweden, and EURO.

rate The conversion rate of the specified country’s currency to the EURO.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner EURO only.

EURO_set_cross_rate

Context Sensitive • EURO

sets the cross rate method between two currencies.

EURO_set_cross_rate (*currency₁*, *currency₂*, *conversion_mode*, *decimal*, *direct_rate*);

<i>currency₁</i>	The country whose currency you want to compare to <i>currency₂</i> . One of the following countries can be specified: Austria, Belgium, Finland, France, Germany, Ireland, Italy, Luxembourg, Netherlands, Portugal, Spain, Great Britain, Denmark, Greece, Sweden, and EURO.
<i>currency₂</i>	The country whose currency is compared to <i>currency₁</i> . One of the following countries can be specified: Austria, Belgium, Finland, France, Germany, Ireland, Italy, Luxembourg, Netherlands, Portugal, Spain, Great Britain, Denmark, Greece, Sweden, and EURO.
<i>conversion_mode</i>	The cross rate method of conversion. You can specify one of the following rates: EURO Triangulation (default): indicates that the cross rates conversion from one national currency unit into another is done via the EURO currency, and that the EURO amount is rounded to no less than three decimal places. Direct Cross Rate: indicates that the conversion is not done via triangulation.
<i>decimal</i>	Indicates the number of decimals to which the EURO amount is rounded (default is set to 3).
<i>direct_rate</i>	The direct cross rate to be used for the conversion between the two currencies.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner EURO only.

EURO_set_currency_threshold

Context Sensitive • EURO

sets the minimum value of an integer which will be considered a currency.

EURO_set_currency_threshold (*threshold*);

threshold The minimum value.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner EURO only.

EURO_set_decimals_precision

Context Sensitive • EURO

sets the number of decimals in the conversion results.

EURO_set_decimals_precision (*decimals*);

decimals Indicates the number of decimals to be displayed in the results (STANDARD, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15).

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner EURO only.

EURO_set_original_new_currencies

Context Sensitive • EURO

sets the original and new currencies of the application.

EURO_set_original_new_currencies (*original_currency*, *new_currency*, *align_mode*, *align_value*);

<i>original_currency</i>	The country whose currency you want to set to <i>new_currency</i> . One of the following countries can be specified: Austria, Belgium, Finland, France, Germany, Ireland, Italy, Luxembourg, Netherlands, Portugal, Spain, Great Britain, Denmark, Greece, Sweden, and EURO.
<i>new_currency</i>	The country to whose currency you want to convert <i>original_currency</i> .
<i>align_mode</i>	One of the following modes can be specified: ALIGN_NONE: No currency alignment ALIGN_ROUND: Rounds the converted currency to the nearest multiple specified in <i>align_value</i> . ALIGN_SUFFIX_DOWN: Rounds down the converted currency value to end with the suffix value indicated in <i>align_value</i> . ALIGN_SUFFIX_UP: Rounds up the converted currency value to end with the suffix value indicated in <i>align_value</i> . ALIGN_TRUNC: Rounds the converted currency value down to the nearest unit.
<i>align_value</i>	The value to align the currency.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner EURO only.

EURO_set_regional_symbols

Context Sensitive • EURO

sets the character used as decimal separator and the character used to separate groups of digits to the left of the decimal.

EURO_set_regional_symbols (*decimal_symbol*, *grouping_symbol*);

decimal_symbol The decimal symbol: "."

grouping_symbol The grouping symbol: ","

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is available for WinRunner EURO only.

EURO_set_triangulation_decimals

Context Sensitive • EURO

sets the default decimals precision for the EUR triangulation.

EURO_set_triangulation_decimals (*decimals*);

decimals The number of decimals to which the EURO amount is rounded. (The default is set to 3.)

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is available for WinRunner EURO only.

EURO_type_mode

Context Sensitive • EURO

disables/enables overriding of automatic currency recognition for all integer objects in a GUI application.

EURO_type_mode (*mode*);

mode

The type mode. One of the following modes can be specified:

DISABLE_OVERRIDE: Disables all overrides on integer objects.

ENABLE_OVERRIDE: Enables all overrides on integer objects.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner EURO only.

eval

Standard • Miscellaneous

evaluates and executes the enclosed TSL statements.

eval (*statement*₁ [; *statement*₂; ... *statement*_{*n*};]);

statement

Can be composed of one or more TSL statements.

Return Values

This function normally returns an empty string. For the **return** statement, **eval** returns the value of the enclosed parameter.

Availability

This function is always available.

exception_off

Standard • Exception Handling

disables exception handling.

```
exception_off ( exception_name );
```

exception_name The name of the exception.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

exception_off_all

Standard • Exception Handling

disables handling of all exceptions.

```
exception_off_all ( );
```

Return Values

This function has no return value.

Availability

This function is always available.

exception_on

Standard • Exception Handling

enables exception handling.

exception_on (*exception*);

exception

A string expression that names the exception. The string cannot contain any spaces.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

exp

Standard • Arithmetic

calculates the exponential function, e^x , where e is the natural logarithm base and “x” is the exponent.

exp (*x*);

Return Values

This function returns a real number.

Availability

This function is always available.

file_close

Standard • I/O

closes a file that was opened with `file_open`.

```
file_close ( file_name );
```

file_name The name of the file to close.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

file_compare

Standard • I/O

compares the contents of two files.

```
file_compare ( file_1, file_2 [ , save_file ] );
```

file_1 The name of a file to compare to *file_2*. If the file is not in the current test directory, then include the full path.

file_2 If the file is not in the current test directory, then include the full path.

save_file The name of a file that saves the files for future viewing.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

file_getline

Standard • I/O

reads the next line from a file and assigns it to a variable.

```
file_getline ( file_name, out_line );
```

<i>file_name</i>	The name of an open file.
<i>out_line</i>	The output variable that stores the line that is read.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

file_open

Standard • I/O

opens a file or creates a new file.

```
file_open ( file_name, mode );
```

<i>file_name</i>	The name of the file to open or create.
<i>mode</i>	The file mode: FO_MODE_READ, or 0 (read only); FO_MODE_WRITE, or 1 (write only); FO_MODE_APPEND, or 2 (write only, to the end of the file).

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

file_printf

Standard • I/O

prints formatted output to an open file.

```
file_printf ( file_name, format, exp1 [ , exp2, ... exp30 ] );
```

<i>file_name</i>	The file to which the output is printed.
<i>format</i>	May include both literal text to be printed and formatting specifications.
<i>exp₁, exp₂, ... exp₃₀</i>	The expressions to format and print.

Formatting Specifications

The first character of the format argument is always a percent sign (%). The last character of format is a letter code that determines the type of formatting. One or more format modifiers can appear between the first and last character of the format argument (see below). The possible letter codes are as follows:

<i>c</i>	Prints a character from its decimal ASCII code.
<i>d</i>	Prints the decimal integer portion of a number.
<i>e</i>	Converts input to scientific notation.
<i>f</i>	Pads with zeros to the right of the decimal point.
<i>g</i>	Prints a decimal value while suppressing non-significant zeros.
<i>o</i>	Prints the octal value of the integer portion of a number.
<i>s</i>	Prints an unmodified string.
<i>x</i>	Prints the hexadecimal value of the integer portion of a number.
<i>%</i>	Prints a literal percent sign (%).

Modifying Formats

The output generated by a particular formatting code can be modified. Three types of modifiers can appear between the percent sign (%) and the format code character:

- (*justification*) A hyphen (-) indicates that the printed output is to be left-justified in its field.

- field width* A number by itself or to the left of a decimal point, indicates how many characters the field should be padded. When this number is preceded by a 0, the padding is done with zeros to the left of the printed value.

- precision* A number to the right of a decimal point indicates the maximum width of the printed string or how many digits are printed to the right of the output decimal point.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

find_text

Analog • I/O

searches for a string in an area of the screen.

Note: This function is provided for backward compatibility only. You should use the corresponding Context Sensitive **win_find_text** and **obj_find_text** functions.

find_text (*string*, *out_coord_array*, *search_area* [, *string_def*]);

<i>string</i>	The string that is searched for. The string must be complete, contain no spaces, and it must be preceded and followed by a space outside the quotation marks. To specify a literal, case-sensitive string, enclose the string in quotation marks. Alternatively, you can specify the name of a string variable. In this case, the string variable can include a regular expression.
<i>out_coord_array</i>	The name of the array that stores the screen coordinates of the text (see explanation below).
<i>search_area</i>	The area to search, specified as coordinates x_1, y_1, x_2, y_2 . These define any two diagonal corners of a rectangle. The interpreter searches for the text in the area defined by the rectangle.
<i>string_def</i>	Defines the type of search to perform. If no value is specified, (0 or FALSE, the default), the search is for a single complete word only. When 1, or TRUE, is specified, the search is not restricted to a single, complete word.

Return Values

If the text is located, this function returns 0. If the text is not found, this function returns 1.

Availability

This function is always available.

generator_add_category

Customization • Function Generator

adds a category to the Function Generator.

```
generator_add_category ( category_name );
```

category_name The name of the category to add.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

generator_add_function

Customization • Function Generator

adds a TSL function to the Function Generator.

```
generator_add_function ( function_name, description, arg_number, arg_name1,  
                          arg_type1, default_value1 [ , ... arg_namen, arg_typen, default_valuen ] );
```

function_name The name of the function being defined, expressed as a string.

description A brief description of the function. This need not be a valid string expression, meaning it may have spaces within the sentence.

arg_number The number of arguments in the function being defined. This can be any number from zero to eight.

For each argument in the function being defined, repeat each of the parameters below; **generator_add_function** can be used to define a function with up to eight arguments.

<i>arg_name</i>	The name of the argument.
<i>arg_type</i>	Defines how the user fills in the value of the argument in the Function Generator. This can be: <i>browse()</i> : user points to a file in a browse file dialog box <i>point_window</i> : user points to a window <i>point_object</i> : user points to a GUI object <i>select_list(0 1)</i> : user selects a value from a list. The <i>select_list</i> argument is defined in the Function Generator by using a combo box. <i>type_edit</i> : user types in a value
<i>default_value</i>	The default value of the argument.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

generator_add_function_to_category Customization • Function Generator

adds a function in the Function Generator to a category.

```
generator_add_function_to_category ( category_name, function_name );
```

category_name The name of an existing category.

function_name The name of an existing function.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

generator_add_subcategory Customization • Function Generator

adds a subcategory to a category in the Function Generator.

```
generator_add_subcategory ( category_name, sub_category_name );
```

category_name The name of an existing category.

sub_category_name The name of an existing category.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

generator_set_default_function

Customization • Function Generator

sets a default function for a category in the Function Generator.

```
generator_set_default_function ( category_name, function_name );
```

category_name An existing category.

function_name An existing function.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

get_aut_var

Standard • Testing Option

returns the value of a variable that determines how WinRunner learns descriptions of objects, records tests, and runs tests on Java applets or applications.

```
get_aut_var ( variable, value );
```

variable The variable to get.

value The value of the variable.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available only for WinRunner with Java support.

get_class_map

Context Sensitive • GUI Map Configuration

returns the standard class associated with a custom class.

get_class_map (*custom_class*, *out_standard_class*);

custom_class The name of the custom class.

out_standard_class The output variable that stores the Mercury class or the standard MS Windows class associated with the custom class.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner and LoadRunner GUI Vusers on PC platforms only.

get_host_name

Standard • Load Testing

returns the name of a host.

get_host_name ();

Return Value

This function returns the host name if the operation is successful or null if the operation fails.

Availability

This function is available for LoadRunner GUI Vusers only.

get_master_host_name

Standard • Load Testing

returns the name of the controller's host.

```
get_master_host_name ( );
```

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is available for LoadRunner GUI Vusers only.

get_record_attr

Context Sensitive • GUI Map Configuration

returns the properties learned for an object class.

```
get_record_attr ( class, out_obligatory, out_optional, out_selector );
```

<i>class</i>	The name of the Mercury class, MSW_class, or X_class.
<i>out_obligatory</i>	The output variable that stores the list of obligatory properties that are always recorded.
<i>out_optional</i>	The output variable that stores the list of optional properties.
<i>out_selector</i>	The output variable that stores the selector used for this GUI object class.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

get_record_method

Context Sensitive • GUI Map Configuration

returns the record method used for an object class.

get_record_method (*class*, *out_method*);

class The name of the object class.

out_method The record method used for the object class, as described below:

Method	Description
RM_RECORD	Records operations using Context Sensitive functions. This is the default method for all the standard classes, except the object class (for which the default is MIC_MOUSE).
RM_IGNORE	Turns off recording.
RM_AS_OBJECT	Instructs WinRunner to record all functions on a GUI object as though its class were “object” class.
RM_PASSUP	Records mouse operations (relative to the parent of the object) and keyboard input.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

get_text

Analog • Text Checkpoint

reads text from the screen.

Note: This function is provided for backward compatibility only. You should use the corresponding Context Sensitive `win_get_text` and `obj_get_text` functions. When working with RTL-style windows, use the `str_map_logical_to_visual` function.

`get_text (location);`

The `get_text` function reads text from the area of the screen indicated by *location*. The *location* can be any one of the following:

x_1, y_1, x_2, y_2	Describes a rectangle that encloses the text to be read. The pairs of coordinates can designate any two diagonally opposite corners of the rectangle.
x, y	The coordinates of a particular point on the screen. This parameter causes the string closest to the specified point to be read. The search radius around the specified point is defined by the <code>XR_TEXT_SEARCH_RADIUS</code> parameter.
<code>()</code>	When no <i>location</i> is specified (empty parentheses), the string closest to the mouse pointer position is read. The search radius around the pointer position is defined by the <code>XR_TEXT_SEARCH_RADIUS</code> parameter.

Return Values

This function returns a string. By default, the returned string does not include blanks at the beginning or end of the string. (This is determined by the `XR_TEXT_REMOVE_BLANKS` parameter in the *wrun.ini* file). If no string is found, an empty string is returned.

Availability

This function is always available.

get_time**Standard • Time-Related**

returns the current system time, expressed in terms of the number of seconds that have elapsed since 00:00 GMT, January 1, 1970.

get_time ();

Return Values

This function returns an integer.

Availability

This function is always available.

get_x**Analog • Input Device**

returns the x-coordinate of the current position of the mouse pointer.

get_x ();

Return Values

This function returns an integer.

Availability

This function is always available.

get_y**Analog • Input Device**

returns the y-coordinate of the current position of the mouse pointer.

get_y ();

Return Values

This function returns an integer.

Availability

This function is always available.

getenv

Standard • Miscellaneous

returns the value of any environment variable, as defined in the [WrCfg] section of *wrun.ini* or in the WinRunner runtime environment.

getenv (*environment_variable*);

environment_variable A variable chosen from the environment variable list in the [WrCfg] section of the *wrun.ini* file.

Return Values

This function returns the value of the specified environment variable.

Availability

This function is always available.

getvar

Standard • Testing Option

returns the value of a testing option.

getvar (*option*);

option A testing option.

The **getvar** function reads the current value of a testing option. For a list and an in-depth explanation of **getvar** options, refer to the “Setting Testing Options from a Test Script” chapter in the *WinRunner User’s Guide*.

Return Values

This function returns the value of the specified testing option.

Availability

This function is always available.

GUI_add

Context Sensitive • GUI Map Editor

adds an object to a GUI map file.

GUI_add (*file path*, *window*, *object*, *physical_desc*);

<i>file</i>	The GUI map file to which the object is added. If an empty string is entered, the object is added to the temporary GUI map file.
<i>window</i>	The logical name of the window containing the object.
<i>object</i>	The logical name of the object.
<i>physical_desc</i>	The physical description of the object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

GUI_buf_get_desc

Context Sensitive • GUI Map Editor

returns the physical description of an object in a GUI map file.

GUI_buf_get_desc (*file*, *window*, *object*, *out_desc*);

<i>file</i>	The full path of the GUI map file containing the object.
<i>window</i>	The logical name of the window containing the object.
<i>object</i>	The logical name of the object. If a null string is specified, the function returns the physical description of the window itself.
<i>out_desc</i>	The output variable that stores the physical description.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

GUI_buf_get_desc_attr**Context Sensitive • GUI Map Editor**

returns the value of a GUI object property in a GUI map file.

GUI_buf_get_desc_attr (*file, window, object, property, out_prop_value*);

<i>file</i>	The full path of the GUI map file containing the object.
<i>window</i>	The logical name of the window containing the object.
<i>object</i>	The logical name of the object. If no object is specified, the function returns the physical description of the window itself.
<i>property</i>	The property whose value is to be returned.
<i>out_prop_value</i>	The output variable that stores the property value.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

GUI_buf_get_logical_name

Context Sensitive • GUI Map Editor

returns the logical name of an object in a GUI map file.

GUI_buf_get_logical_name (*file*, *physical_desc*, *window*, *out_name*);

<i>file</i>	The full path of the GUI map file containing the object.
<i>physical_desc</i>	The physical description of the GUI object.
<i>window</i>	The window containing the GUI object.
<i>out_name</i>	The output variable that stores the logical name.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

GUI_buf_new

Context Sensitive • GUI Map Editor

creates a new GUI map file.

GUI_buf_new (*file*);

<i>file</i>	The GUI map file to create.
-------------	-----------------------------

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

GUI_buf_set_desc_attr

Context Sensitive • GUI Map Editor

sets the value of a property for an object in a GUI map file.

GUI_buf_set_desc_attr (*file*, *window*, *object*, *property*, *value*);

<i>file</i>	The full path of the GUI map file containing the object.
<i>window</i>	The window containing the object.
<i>object</i>	The logical name of the object.
<i>property</i>	The property whose value is to be set.
<i>value</i>	The value set for the property.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

GUI_close

Context Sensitive • GUI Map Editor

closes a GUI map file.

GUI_close (*file*);

<i>file</i>	The full path of the GUI map file to be closed.
-------------	---

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

GUI_close_all

Context Sensitive • GUI Map Editor

closes all GUI map files except the temporary GUI map file. To close the temporary GUI map file, use the `GUI_close` function.

GUI_close_all ();

The `GUI_close_all` function closes all GUI map files that are currently loaded or open.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

GUI_delete

Context Sensitive • GUI Map Editor

deletes an object from a GUI map file.

GUI_delete (*file*, *window*, *obj*);

<i>file</i>	The full path of the GUI map file containing the object.
<i>window</i>	The logical name of the window containing the object.
<i>obj</i>	The logical name of the object to delete.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

GUI_desc_compare

Context Sensitive • GUI Map Editor

compares two physical descriptions.

GUI_desc_compare (*desc₁*, *desc₂*);

desc₁, *desc₂* The physical descriptions to compare.

Return Value

This function returns 1 when the comparison fails and returns 0 when it succeeds.

Availability

This function is always available.

GUI_desc_get_attr

Context Sensitive • GUI Map Editor

gets the value of a property from a physical description.

GUI_desc_get_attr (*physical_desc*, *property*, *out_property_value*);

physical_desc The physical description of a GUI object.

property The property to return.

out_property_value The output variable that stores the property value.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

GUI_desc_set_attr

Context Sensitive • GUI Map

sets the value of a property.

GUI_desc_set_attr (*physical_desc*, *property*, *value*);

<i>physical_desc</i>	The physical description of an object. This must be a variable and not a constant.
<i>property</i>	The property name.
<i>value</i>	The property value.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

GUI_get_name

Context Sensitive • GUI Map Editor

returns the type of GUI for the application under test.

GUI_get_name (*out_name*, *out_version*);

<i>out_name</i>	An output variable that stores the name of the current GUI.
<i>out_version</i>	An output variable that stores the current version of the GUI, as described below:

Operating System	Name	Version
Microsoft Windows 95	"Windows 95"	"4.0"
Microsoft Windows 98	"Windows 95"	"4.1"
Microsoft Windows NT	"Windows NT"	"4.0"
Microsoft Windows 2000 (when available)	"Windows 2000"	"5.0"

Note: Windows 98 is called Windows 95 for purposes of backward compatibility. The major version number for both operating systems is 4. The minor version number is 0 for Windows 95 or 1 for Windows 98.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

GUI_get_window

Context Sensitive • GUI Map Editor

returns the active window in the GUI map.

`GUI_get_window ();`

Return Values

This function returns the name of the active window if it succeeds, or an empty string if it fails.

Availability

This function is always available.

GUI_list_buf_windows

Context Sensitive • GUI Map Editor

lists all windows in a GUI map file.

GUI_list_buf_windows (*file*, *out_windows*, *out_number*);

<i>file</i>	The full path of the GUI map file.
<i>out_windows</i>	The output variable that stores all windows in the GUI map file in an array.
<i>out_number</i>	The output variable assigned to the number of windows in the GUI map file.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

GUI_list_buffers

Context Sensitive • GUI Map Editor

lists all open GUI map files.

GUI_list_buffers (*out_files*, *out_number*);

<i>out_files</i>	The output variable array that stores all open GUI map files in an array.
<i>out_number</i>	The output variable that stores the number of opened GUI map files.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

GUI_list_desc_attrs

Context Sensitive • GUI Map Editor

lists property values for a GUI object.

GUI_list_desc_attrs (*physical_desc*, *out_array*);

<i>physical_desc</i>	The physical description of a GUI object.
<i>out_array</i>	The output variable that stores the object's properties and values in an array. The subscript of each array element is the name of the property. The value of each array element is the value of the property. For instance, if the <i>out_array</i> is called <i>property_value</i> , then: <i>property_value</i> ["attr1"] = "val1".

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

GUI_list_map_buffers

Context Sensitive • GUI Map Editor

lists all loaded GUI map files.

GUI_list_map_buffers (*out_file*, *out_number*);

<i>out_file</i>	The output variable that stores all loaded GUI map files in an array.
<i>out_number</i>	The output variable that stores the number of loaded GUI map files.

Note: The GUI map files must be loaded and not simply open.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

GUI_list_win_objects**Context Sensitive • GUI Map Editor**

lists all objects in a window.

GUI_list_win_objects (*file*, *window*, *out_objects*, *out_number*);

<i>file</i>	The full path of the GUI map file.
<i>window</i>	The name of the window containing the objects.
<i>out_objects</i>	The output variable that stores all objects in the window in an array.
<i>out_number</i>	The output variable that stores the number of objects in the window.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

GUI_load

Context Sensitive • GUI Map Editor

loads a GUI map file.

GUI_load (*file_name*);

file_name The full path of the GUI map.

Note: If you do not specify a full path, then WinRunner searches for the GUI map relative to the current file system directory. Therefore, you must always specify a full path to ensure that WinRunner will find the GUI map.

Note: If you are working in the *GUI Map File per Test* mode, you should not manually load or unload GUI map files.

Return Values

This function always returns 0.

Availability

This function is always available.

GUI_map_get_desc

Context Sensitive • GUI Map Editor

returns the description of an object in the GUI map.

GUI_map_get_desc (*window*, *object*, *out_desc*, *out_file*);

<i>window</i>	The name of the window containing the GUI object.
<i>object</i>	The logical name of the GUI object.
<i>out_desc</i>	The output variable that stores the description of the GUI object.
<i>out_file</i>	The output variable that stores the GUI map file containing the description.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

GUI_map_get_logical_name

Context Sensitive • GUI Map Editor

returns the logical name of an object in the GUI map.

GUI_map_get_logical_name (*physical_desc*, *window*, *out_obj*, *out_file*);

<i>physical_desc</i>	The physical description of the object. For more information regarding <i>physical descriptions</i> , refer to the “Introducing the GUI Map” chapter in the <i>WinRunner User’s Guide</i> .
<i>window</i>	The logical name of the window containing the object. If no window is specified, the function looks for one.
<i>out_obj</i>	The output variable that stores the object’s logical name.
<i>out_file</i>	The output variable that stores the name of the GUI map file containing the object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

GUI_open

Context Sensitive • GUI Map Editor

opens a GUI map file.

GUI_open (*file_name*);

file_name The full path of the GUI map file to open.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

GUI_save

Context Sensitive • GUI Map Editor

saves a GUI map file.

GUI_save (*file_name*);

file_name The full path of the GUI map file to save.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

GUI_save_as

Context Sensitive • GUI Map Editor

saves a GUI map file under a new name.

GUI_save_as (*current_file_name*, *new_file_name*);

current_file_name The name of the GUI map file to save.

new_file_name The name of the new file.

Note: When you save the temporary GUI map file, which doesn't have a *current_file_name*, the statement should have the following syntax:

GUI_save_as ("", "new_file_name");

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

GUI_set_window

Context Sensitive • GUI Map Editor

sets the scope for GUI object identification within the GUI map.

GUI_set_window (*window_name*);

window_name The name of the window to be activated.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

gui_ver_add_check

Customization • GUI Checkpoint

registers a new GUI check.

```
gui_ver_add_check ( check_name, capture_function, comparison_function
                   [ , display_function [ , type ] ] );
```

<i>check_name</i>	The name of the check to add.
<i>capture_function</i>	The name of the capture function defined for the check.
<i>comparison_function</i>	The name of the comparison function defined for the check. If no <i>comparison_function</i> is specified, the default display is used.
<i>display_function</i>	The name of the function that displays check results.
<i>type</i>	The type of GUI object on which this check operates: 1 for a window, 0 for any other GUI object class. If no <i>type</i> is specified, the default 0 is assumed.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

gui_ver_add_check_to_class

Customization • GUI Checkpoint

adds a check to an object class, which can be viewed in the GUI Checkpoint dialog boxes.

```
gui_ver_add_check_to_class ( class, check_name );
```

<i>class</i>	The name of the class.
<i>check_name</i>	The name of the check to add, as defined with gui_ver_add_check .

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

gui_ver_add_class**Customization • GUI Checkpoint**

Creates a GUI checkpoint for a new class.

```
gui_ver_add_class ( TOOLKIT_class [ , ui_function [ , default_check_function ] ] );
```

<i>TOOLKIT_class</i>	The MSW_class or X_class of the object.
<i>ui_function</i>	The name of the function used to develop and display the GUI checkpoint dialog boxes with a customized user interface.
<i>default_check_function</i>	The name of the function that controls the default checks for the object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

gui_ver_set_default_checks**Customization • GUI Checkpoint**

sets the default GUI checks for an object class.

```
gui_ver_set_default_checks ( class, check_names );
```

<i>class</i>	The name of the object class.
<i>check_names</i>	The names of the checks set as defaults, separated by spaces.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

icon_move**Context Sensitive • Icon Object**

moves an icon to a new location on the screen.

icon_move (*icon*, *x*, *y*);

<i>icon</i>	The logical name of the icon.
<i>x</i> , <i>y</i>	The new position of the upper left corner of the icon.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner and LoadRunner GUI Vusers running on PC platforms only.

icon_select**Context Sensitive • Icon Object**

selects an icon with a mouse click.

icon_select (*icon*);

<i>icon</i>	The logical name of the icon.
-------------	-------------------------------

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner and LoadRunner GUI Vusers running on PC platforms only.

index

Standard • String

indicates the position of one string within another.

index (*string₁*, *string₂*);

string₁, *string₂* Two string expressions.

Return Values

The return value indicates the position of the string. The value 0 is returned if the string does not exist.

Availability

This function is always available.

int

Standard • Arithmetic

returns the integer portion of a positive real number.

int (*x*);

Return Values

This function returns an integer.

Availability

This function is always available.

invoke_application

Standard • Operating System

invokes a Windows application from within a test script.

invoke_application (*file*, *command_option*, *working_dir*, *show*);

file The full path of the application to invoke.

command_option The command line options to apply.

working_dir The working directory for the specified application.

show

Specifies how the application appears when opened. This parameter can be one of the following constants:

Value	Description
SW_HIDE	hides the window and passes activation to another window
SW_MINIMIZE	minimizes the window and activates the top-level window in the system list
SW_RESTORE	activates and displays the window. If the window is minimized or maximized, WinRunner restores it to its original size and position (same as SW_SHOWNORMAL).
SW_SHOW	activates the window and displays it in its current size and position
SW_SHOWMAXIMIZED	activates the window and displays it as a maximized window
SW_SHOWMINIMIZED	activates the window and displays it as an icon
SW_SHOWMINNOACTIVE	displays the window as an icon. The window that is currently active remains active.
SW_SHOWNA	displays the window in its current state. The currently active window remains active.
SW_SHOWNOACTIVATE	displays the window in its most recent size and position. The currently active window remains active.
SW_SHOWNORMAL	activates and displays the window. If the window is minimized or maximized, WinRunner restores it to its original size and position (same as SW_SHOWRESTORE).

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner and LoadRunner GUI Vusers running on PC platforms only.

java_activate_method

Context Sensitive • Java

invokes the requested Java method for the given object.

```
int java_activate_method ( object, method, retval [ , param1, ... param8 ] );
```

<i>object</i>	The object name
<i>method</i>	The name of the java method to invoke
<i>retval</i>	An output variable that will hold a return value from the invoked method*
	*Required even for void Java methods
<i>param</i> _{1...8}	Parameters to be passed to the Java method. The Parameters must belong to one of the following supported types: Boolean, boolean, Integer, int, String.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner with Java support only.

java_fire_event

Context Sensitive • Java

Simulates an event on a Java object.

```
java_fire_event ( object , class [ , constructor_param1,..., constructor_paramX ] );
```

<i>object</i>	The logical name of the Java object.
<i>class</i>	The name of the Java class representing the event to be activated.
<i>constructor_param</i> ₁ ... <i>constructor_param</i> _X	A list of all the constructor parameters.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner with Java support only.

jco_create**Context Sensitive • Java**

Creates a Java object within your application or applet, or within the context of an existing object in your application or applet.

jco_create (*object* , *jco* , *class* , [*param1* , ... , *param8*])

<i>object</i>	The object that is used as the context in which the new object will be created. This can be the main application or applet window, or any other Java object within the application or applet.
<i>jco</i>	The new object to be returned.
<i>class</i>	The Java class name.
<i>param1...param8</i>	The required parameters for that object constructor (excluding the object source, as this was specified in the <i>object</i> parameter). These parameters can be of type: int, float, boolean, string, or jco.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner with Java support only.

jco_free

Context Sensitive • Java

frees the specified jco object from memory.

```
jco_free ( object_name );
```

object_name The name of the jco object to be freed.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner with Java support only.

jco_free_all

Context Sensitive • Java

frees all jco objects from memory.

```
jco_free_all();
```

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner with Java support only.

jdc_aut_connect

Context Sensitive • Java

establishes a connection between WinRunner and Java applications.

jdc_aut_connect (*in_timeout*);

timeout

Time (in seconds) that is added to the regular **timeout for checkpoints and CS statements** (**Settings > General Options > Run Tab**), resulting in the maximum interval before the next statement is executed.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner with Java support only.

length

Standard • String

counts the number of characters in a string.

length (*string*);

string A valid string expression.

Return Values

The return value of the function indicates the number of characters in the argument string. If no string is included, **length** returns the value 0.

Availability

This function is always available.

list_activate_item

Context Sensitive • List Object

activates an item in a list.

list_activate_item (*list*, *item* [, *offset*]);

list The logical name of the list.
item The item to activate within the list.
offset The horizontal offset (in pixels) of the click location relative to the left margin of the item's text (optional).

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

list_check_info

Context Sensitive • List Object

checks the value of a list property.

list_check_info (*list*, *property*, *property_value*);

<i>list</i>	The logical name of the list.
<i>property</i>	The property to be checked.
<i>property_value</i>	The expected property value.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

list_check_item

Context Sensitive • List Object

checks the content of an item in a list.

list_check_item (*list*, *item_num*, *item_content*);

<i>list</i>	The logical name of the list.
<i>item_num</i>	The location of the item in the designated list. Note that the first item in a list is numbered 0.
<i>item_content</i>	The expected contents of the item.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

list_check_selected

Context Sensitive • List Object

checks that the specified item is selected.

list_check_selected (*list*, *selected_items*);

<i>list</i>	The logical name of the list.
<i>selected_item</i>	The item(s) that should be selected in the list. If there are multiple items, they should be separated by commas. This argument should be a string or a list of strings.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

list_collapse_item

Context Sensitive • List Object

hides items in a TreeView object.

list_collapse_item (*list*, *item* [, *mouse_button*]);

<i>list</i>	The logical name of the list.
<i>item</i>	The expanded heading under which the items appear.
<i>mouse_button</i>	A constant that specifies the mouse button to use. The value can be LEFT, MIDDLE, or RIGHT. The default is the left button.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for TreeView objects only.

list_deselect_item

Context Sensitive • List Object

deselects an item in a list.

```
list_deselect_item ( list, item [ , mouse_button [ , offset ] ] );
```

<i>list</i>	The logical name of the list.
<i>item</i>	The item to deselect from the list.
<i>mouse_button</i>	A constant that specifies the mouse button to use. The value can be LEFT, MIDDLE, or RIGHT. If no button is specified, the default is the left button.
<i>offset</i>	The horizontal offset (in pixels) of the click location relative to the left margin of the item's text (optional). This parameter may only be used if the <i>mouse_button</i> argument is used

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

list_deselect_range

Context Sensitive • List Object

deselects all items between two specified items.

```
list_deselect_range ( list, item1, item2 [ , offset ] );
```

<i>list</i>	The logical name of the list.
<i>item₁</i>	The first item of the range.
<i>item₂</i>	The last item of the range.
<i>offset</i>	The horizontal offset (in pixels) of the click location relative to the left margin of the item's text (optional).

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

list_drag_item**Context Sensitive • List Object**

drags an item from a source list.

list_drag_item (*source_list*, *item* [, *mouse_button*]);

<i>source_list</i>	The logical name of the list.
<i>item</i>	The item to drag from the list.
<i>mouse_button</i>	A constant that specifies the mouse button to hold down while dragging the item. The value can be LEFT, MIDDLE, or RIGHT. If no button is specified, the default is the button that performs the select function.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is not supported for ListBox objects.

list_drop_on_item**Context Sensitive • List Object**

drops an object onto a target list item.

list_drop_on_item (*target_list*, *target_item*);

<i>target_list</i>	The logical name of the list.
<i>target_item</i>	The list item on which to drop the source object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is not supported for ListBox objects.

list_expand_item**Context Sensitive • List Object**

displays hidden items in a TreeView object.

list_expand_item (*list*, *item* [, *mouse_button*]);

<i>list</i>	The logical name of the list.
<i>item</i>	The expandable heading under which the items will be displayed.
<i>mouse_button</i>	A constant that specifies the mouse button to use. The value can be LEFT, MIDDLE, or RIGHT. The default is the left button.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for TreeView objects only.

list_extend_item

Context Sensitive • List Object

adds an item to a list of selected items.

```
list_extend_item ( list, item [ , button [ , offset ] ] );
```

<i>list</i>	The logical name of the list.
<i>item</i>	The item to add from the list.
<i>button</i>	The mouse button used (optional). In the case of a combo object or a list that is not a ListView or a TreeView, only the left mouse button can be used.
<i>offset</i>	The horizontal offset (in pixels) of the click location relative to the left margin of the item's text (optional). This argument can be used only if the button argument is defined.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

list_extend_multi_items

Context Sensitive • List Object

adds multiple items to the items already selected in a list.

```
list_extend_multi_items ( list, item_list, [ , mouse_button [ , offset ] ] );
```

<i>list</i>	The logical name of the list.
<i>item_list</i>	The items to select, separated by commas.
<i>mouse_button</i>	A constant that specifies the mouse button to use. The value can be LEFT, MIDDLE, or RIGHT. The default is the left button.
<i>offset</i>	The horizontal offset (in pixels) of the click location relative to the left margin of the item's text (optional). This argument can be used only if the button argument is defined.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

list_extend_range**Context Sensitive • List Object**

selects a range of items and adds them to the current selection.

list_extend_range (*list*, *item*₁, *item*₂ [, *button* [, *offset*]]);

<i>list</i>	The logical name of the list.
<i>item</i> ₁	The first item of the range.
<i>item</i> ₂	The last item of the range.
<i>button</i>	The mouse button used (optional). In the case of a combo object or a list that is not a ListView or a TreeView, only the left mouse button can be used.
<i>offset</i>	The horizontal offset (in pixels) of the click location relative to the left margin of the item’s text (optional). This argument can be used only if the button argument is defined.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

list_get_checked_items

Context Sensitive • List Object

retrieves the number and the value of items marked as checked.

list_get_checked_items (*list*, *items*, *number*);

<i>list</i>	The logical name of the ListView or TreeView with check boxes.
<i>items</i>	The concatenated list of the returned values of the items with selected check boxes.
<i>number</i>	The number of items with selected check boxes.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

list_get_column_header

Context Sensitive • List Object

returns the value of a ListView column header.

list_get_column_header (*listview_object*, *in_column_index*, *out_header_value*);

<i>listview_object</i>	The name of the list.
<i>in_column_index</i>	The column index.
<i>out_header_value</i>	The column header that is returned.

Note: The **list_get_column_header** function is effective for ListView objects having a report view (style) only.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

The `list_get_column_header` function is effective for ListView objects having a report view (style) only.

list_get_info**Context Sensitive • List Object**

returns the value of a list property.

`list_get_info (list, property, out_value);`

<i>list</i>	The logical name of the list.
<i>property</i>	Any of the properties listed in the <i>WinRunner User's Guide</i> .
<i>out_value</i>	The output variable that stores the value of the specified property.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

list_get_item**Context Sensitive • List Object**

returns the contents of a list item.

`list_get_item (list, item_num, out_value);`

<i>list</i>	The logical name of the list.
<i>item_num</i>	The location of the item in the designated list. Note that the first item in a list is numbered 0.
<i>out_value</i>	The contents of the designated item.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

list_get_item_coord

Context Sensitive • List Object

returns the dimensions and coordinates of the list item.

list_get_item_coord (*list*, *item*, *out_x*, *out_y*, *out_width*, *out_height*);

<i>list</i>	The list name
<i>item</i>	The item string
<i>out_x</i> , <i>out_y</i>	The output variables that store the x,y coordinates of the item rectangle
<i>out_width</i> , <i>out_height</i>	The output variables that store the width and height of the item rectangle

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available only for list and tree objects in JFC.

list_get_item_info

Context Sensitive • List Object

returns the state of a list item.

list_get_item_info (*list*, *item*, *state*, *out_value*);

<i>list</i>	The logical name of the list.
<i>item</i>	The item in the list.
<i>state</i>	The state property of the item. The state property can be either CHECKED or SELECTED.
<i>out_value</i>	The output variable that stores the value of the state property.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

list_get_item_num

Context Sensitive • List Object

returns the position of a list item.

list_get_item_num (*list*, *item*, *out_num*);

<i>list</i>	The logical name of the list.
<i>item</i>	The string of the item.
<i>out_num</i>	The output variable that stores the position of the list item.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

list_get_selected

Context Sensitive • List Object

returns the numeric and string values of the selected item in a list.

list_get_selected (*list*, *out_item*, *out_num*);

<i>list</i>	The logical name of the list.
<i>out_item</i>	The output variable that stores the name of the selected items. For a multi-selection list, the variable contains a list of items, sorted alphabetically, and separated by the character that is set in the Miscellaneous tab of the Settings > General Options dialog box. The default character is a comma (,).

Note: When using this function with the Java Add-in, always use special character ASCII30 (^) as the separator, and not the character set in the Miscellaneous tab as described above.

<i>out_num</i>	The output variable that stores the items. Note that the first item in a list is numbered 0. For a standard list, stores the index of the selected item. For a multi-selection list, stores the number of selected items.
----------------	---

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

list_get_subitem

Context Sensitive • List Object

returns the value of a ListView subitem.

list_get_subitem (*list*, *item*, *subitem_index*, *subitem*);

<i>list</i>	The logical name of the ListView.
<i>item</i>	The name of the item.
<i>subitem_index</i>	The index indicating the field of the requested subitem.
<i>subitem</i>	The value of the returned subitem.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

list_rename_item

Context Sensitive • List Object

activates the edit mode on the label of a ListView or a TreeView item in order to rename it.

list_rename_item (*list*, *item*);

<i>list</i>	The logical name of the ListView or TreeView.
<i>item</i>	The item to select and rename.

Note: A **list_rename_item** statement must be followed by a type statement in order to rename the item. The item can be denoted by its logical name or numeric index.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

list_select_item

Context Sensitive • List Object

selects a list item.

list_select_item (*list*, *item* [,*button* [, *offset*]]);

<i>list</i>	The logical name of the list.
<i>item</i>	The item to select in the list.
<i>button</i>	The mouse button used (optional). In the case of a combo object or a list that is not a ListView or a TreeView, only the left mouse button can be used.
<i>offset</i>	The horizontal offset (in pixels) of the click location relative to the left margin of the item's text (optional). This argument can be used only if the button argument is defined.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

list_select_multi_items

Context Sensitive • List Object

selects multiple items in a list.

```
list_select_multi_items ( list, item_list [ , mouse_button [ , offset ] ] );
```

<i>list</i>	The logical name of the list.
<i>item_list</i>	The items to select, separated by commas.
<i>mouse_button</i>	A constant that specifies the mouse button to use. The value can be LEFT, MIDDLE, or RIGHT. The default is the left button.
<i>offset</i>	The horizontal offset (in pixels) of the click location relative to the left margin of the item's text (optional). This argument can be used only if the button argument is defined.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

list_select_range

Context Sensitive • List Object

selects all items between two specified items.

```
list_select_range ( list, item1, item2 [ , button [ , offset ] ] );
```

<i>list</i>	The logical name of the list.
<i>item₁</i>	The first item of the range.
<i>item₂</i>	The last item of the range.
<i>button</i>	The mouse button used (optional). In the case of a combo object or a list that is not a ListView or a TreeView, only the left mouse button can be used.

offset The horizontal offset (in pixels) of the click location relative to the left margin of the item's text (optional). This argument can be used only if the *button* argument is defined.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

list_set_item_state

Context Sensitive • List

sets the state of an icon of the specified ListView or TreeView.

list_set_item_state (*list*, *item*, *value* [, *button*]);

<i>list</i>	The logical name of the ListView or TreeView.
<i>item</i>	The name of the icon.
<i>value</i>	The value of the state icon (check box). The value can be 1 (ON) or 0 (OFF).
<i>button</i>	The mouse button (optional).

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

list_wait_info

Context Sensitive • List Object

waits for the value of a list property.

list_wait_info (*list*, *property*, *value*, *time*);

<i>list</i>	The logical name of the list.
<i>property</i>	Any of the properties listed in the <i>User's Guide</i> .
<i>value</i>	The property value.
<i>time</i>	Indicates the maximum interval, in seconds, before the next statement is executed.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

load

Standard • Compiled Module

loads a compiled module into memory.

load (*module_name* [,1|0 [,1|0]]);

<i>module_name</i>	A string expression indicating the name of an existing compiled module.
1 0	1 indicates a system module. 0 indicates a user module. The default value is 0.
1 0	1 indicates that a user module will not remain open after it is loaded. 0 indicates that the module remains open in the WinRunner window. The default value is 0.

Note: If you make changes to a function in a loaded compiled module, you must unload and reload the compiled module in order for the changes to take effect.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function returns 0 for success, and 1 for failure.

load_16_dll

Standard • Miscellaneous

performs a runtime load of a 16-bit dynamic-link (external) library.

`load_16_dll (pathname);`

<i>pathname</i>	The full pathname of the dynamic-link library (DLL) to be loaded.
-----------------	---

Note: To call an external function, you must declare it with the extern function declaration.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

load_dll

Standard • Miscellaneous

performs a runtime load of a dynamic-link (external) library.

load_dll (*pathname*);

pathname The full pathname of the dynamic-link library (DLL) to be loaded.

Note: To call an external function, you must declare it with the extern function declaration.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

log

Standard • Arithmetic

returns the natural (base e) logarithm of the specified number.

log (x);

x Specifies a positive, nonzero number.

Return Values

This function returns a real number.

Availability

This function is always available.

lov_get_item

Context Sensitive • Oracle

retrieves an item from a list of values in an Oracle application.

lov_get_item (*list*, *column*, *row*, *out_value*);

<i>list</i>	The name of the list of values.
<i>column</i>	The column number of the item.
<i>row</i>	The row number of the item.
<i>out_value</i>	The parameter where the item will be stored.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

lov_select_item

Context Sensitive • Oracle

selects an item from a list of values in an Oracle application.

lov_select_item (*list*, *item*);

<i>list</i>	The list name.
<i>item</i>	The logical name of the item.

Note: This function cannot be recorded.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Developer 2000 support only.

lr_whoami**Standard • Load Testing**

returns information about the Vuser executing the script.

lr_whoami (*vuser* [, *sgroup*]);

vuser The output variable that stores the ID of the Vuser.

sgroup The output variable that stores the name of the Sgroup.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for LoadRunner GUI Vusers only.

match**Standard • String**

finds the occurrence of a regular expression in a string.

match (*string*, *regular_expression*);

string The enclosing string.

regular_expression The expression to locate in the string.

Return Values

This function returns the character position at which the regular expression starts. If no match is found, the value 0 is returned.

Availability

This function is always available.

menu_get_desc

Context Sensitive • Menu Object

returns the physical description of a menu.

menu_get_desc (*menu*, *oblig*, *optional*, *selector*, *out_desc*);

<i>menu</i>	The full menu path, consisting of the menu's logical name and the menu item, separated by a semicolon (such as file;open). For submenus, the path includes the menu name, menu item, and submenu item.
<i>oblig</i>	The list of obligatory properties (separated by blank spaces).
<i>optional</i>	The list of optional properties (separated by blank spaces).
<i>selector</i>	The type of selector to be used (location or index).
<i>out_desc</i>	The output variable that stores the description of the menu.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

menu_get_info

Context Sensitive • Menu Object

returns the value of a menu property.

menu_get_info (*menu*, *property*, *out_value*);

<i>menu</i>	The full menu path, consisting of the menu's logical name and the menu item, separated by a semicolon (such as file;open). For submenus, the path includes the menu name, menu item, and submenu item.
<i>property</i>	The property to be checked. The following properties may be specified: class, label, value, enabled, MSW_id, sub_menu, count, sys_menu, and position.

out_value The output variable that stores the value of the specified property.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

menu_get_item

Context Sensitive • Menu Object

returns the contents of a menu item.

menu_get_item (*menu*, *item_number*, *out_contents*);

<i>menu</i>	The logical name of the menu. For submenus, the full path, consisting of the menu’s logical name and the menu item, separated by a semicolon (such as file;type).
<i>item_number</i>	The numeric position of the item in the menu. Note that the first position is numbered 0.
<i>out_contents</i>	The output variable to which the value of the designated menu item is assigned.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

menu_get_item_num

Context Sensitive • Menu Object

returns the position of a menu item.

```
menu_get_item_num ( menu, item, out_position );
```

<i>menu</i>	The logical name of the menu. For submenus, the full path, consisting of the menu's logical name and the menu item separated by a semicolon (such as file;type).
<i>item</i>	The name (string value) of the item as it appears in the menu.
<i>out_position</i>	The output variable which stores the numeric value of the item.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

menu_select_item

Context Sensitive • Menu Object

selects a menu item.

```
menu_select_item ( menu; item [ x, y ] );
```

<i>menu</i>	The logical name of the menu.
<i>item</i>	The item to select.
<i>x,y</i>	The position of the mouse click, expressed as x- and y- (pixel) coordinates.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118

Availability

This function is always available.

menu_wait_info**Context Sensitive • Menu Object**

waits for the value of a menu property.

menu_wait_info (*menu*, *property*, *value*, *time*);

<i>menu</i>	The logical name of the menu.
<i>property</i>	Any of the properties listed in the <i>User's Guide</i> .
<i>value</i>	The property value.
<i>time</i>	Indicates the maximum interval, in seconds, before the next statement is executed.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

method_wizard**Context Sensitive • Java**

Launches the Java Method wizard. The wizard enables you to view the methods associated with any jco object in your application or applet and to generate the appropriate **java_activate_method** statement for one of the displayed methods.

method_wizard ([*object*]);

<i>object</i>	The name of the object whose methods will be displayed in the Java Method wizard.
---------------	---

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is available for WinRunner with Java support only.

move_locator_abs

Analog • Input Device

moves the mouse pointer to a new absolute position.

```
move_locator_abs ( x, y [ , time ] );
```

<i>x</i> , <i>y</i>	The absolute screen coordinates of the new pointer position, in pixels.
<i>time</i>	The interval, in milliseconds, that elapses before the locator is moved.

Return Values

This function always returns 0.

Availability

This function is always available.

move_locator_rel

Analog • Input Device

moves the mouse pointer to a new relative position.

```
move_locator_rel ( x, y [ , time ] );
```

<i>x</i> , <i>y</i>	The screen coordinates of the new pointer position, in pixels, relative to the current pointer position.
<i>time</i>	The interval that elapses before the locator is moved, in milliseconds.

Return Values

The return value of the function is always 0.

Availability

This function is always available.

move_locator_text

Analog • Input Device

moves the screen pointer to a string.

```
move_locator_text ( string, search_area [ , x_shift [ , y_shift ] ] );
```

<i>string</i>	A valid string expression. The string must be complete, and preceded and followed by a space. A regular expression with no blank spaces can be specified.
<i>search_area</i>	The area to search, specified as x_1, y_1, x_2, y_2 coordinates that define any two diagonal corners of a rectangle. The interpreter searches for the text in the area defined by the rectangle.
<i>x_shift, y_shift</i>	Indicates the offset of the pointer position from the specified string, in pixels.

Return Values

This function returns 0 if the text is located, and 1 if the text is not found.

Availability

This function is always available.

move_locator_track

Analog • Input Device

moves the mouse pointer along a prerecorded track.

```
move_locator_track ( track_id );
```

<i>track_id</i>	A code that points to tracking information stored in the test database. The specified track is a series of continuous pointer movements uninterrupted by input from keyboard or mouse.
-----------------	--

Return Values

This function always returns the value 0.

Availability

This function is always available.

mtype

Analog • Input Device

specifies mouse button input.

```
mtype ( button_input [ , technical_id ] );
```

<i>button_input</i>	A string expression representing mouse button input.
<i>technical_id</i>	Points to internal timing and synchronization data. This parameter is only present when the mtype statement is recorded.

Return Values

This function always returns the value 0.

Availability

This function is always available.

nargs

Standard • Miscellaneous

returns the number of arguments passed.

```
nargs ( );
```

Return Values

This function returns the number of arguments actually passed, not the number specified in the definition of the function or test.

Availability

This function is always available.

obj_check_bitmap

Context Sensitive • Object

compares an object bitmap to an expected bitmap.

obj_check_bitmap (*object*, *bitmap*, *time* [, *x*, *y*, *width*, *height*]);

<i>object</i>	The logical name of the GUI object. The object may belong to any class.
<i>bitmap</i>	A string expression that identifies the captured bitmap.
<i>time</i>	The interval, which is added to the <i>timeout_msec</i> testing option, marking the maximum delay between the previous input event and the capture of the current bitmap, in seconds. For more information, refer to the “Setting Testing Options from a Test Script” chapter in the <i>WinRunner User’s Guide</i> .
<i>x</i> , <i>y</i>	For an area bitmap: the coordinates of the upper left corner, relative to the window in which the area is located.
<i>width</i> , <i>height</i>	For an area bitmap: the size of the area, in pixels.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

obj_check_gui

Context Sensitive • Object

compares current GUI object data to expected data.

obj_check_gui (*object*, *checklist*, *expected_results_file*, *time*);

<i>object</i>	The logical name of the GUI object. The object may belong to any class.
<i>checklist</i>	The name of the checklist defining the GUI checks.
<i>expected_results_file</i>	The name of the file that stores the expected GUI data.

time The interval, which is added to the timeout test option, marking the maximum delay between the previous input event and the capture of the current GUI data, in seconds. This interval is added to the timeout testing option during test execution.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

obj_check_info

Context Sensitive • Object

checks the value of an object property.

obj_check_info (*object*, *property*, *property_value*);

<i>object</i>	The logical name of the GUI object. The object may belong to any class.
<i>property</i>	The property to check.
<i>property_value</i>	The property value.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

obj_click_on_text

Context Sensitive • Object

clicks on text in an object.

```
obj_click_on_text ( object, string [ , search_area [ , string_def [ , mouse_button ] ] ] );
```

<i>object</i>	The logical name of the object to search.
<i>string</i>	The text to locate. To specify a literal, case sensitive string, enclose the string in quotation marks. Alternatively, you can specify a string variable, which can include a regular expression. The regular expression need not begin with an exclamation mark.
<i>search_area</i>	The region of the object to search, relative to the object. This area is defined as a pair of coordinates, with x_1, y_1, x_2, y_2 specifying any two diagonally opposite corners of the rectangular search region. If no <i>search_area</i> is defined, then the entire object is considered as the search area.
<i>string_def</i>	Defines how the text search is performed. If no <i>string_def</i> is specified (0 or FALSE, the default parameter), the interpreter searches for a single, complete word only. If 1, or TRUE, is specified, the search is not restricted to a single, complete word.
<i>mouse_button</i>	Specifies the mouse button that clicks on the text string. The value can be LEFT, MIDDLE, or RIGHT. If no button is specified, the default is the left button. Note that if you specify 1, or TRUE, for <i>string_def</i> , then you must specify the mouse button to use. Similarly, if you specify the mouse button to use, then you must specify the <i>string_def</i> .

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

obj_drag

Context Sensitive • Object

drags an object from a source object.

obj_drag (*source_object*, *x*, *y* [, *mouse_button*]);

<i>source_object</i>	The logical name of the GUI object. The object may belong to any class.
<i>x</i> , <i>y</i>	The <i>x,y</i> coordinates of the mouse pointer when clicked on the source object, relative to the upper left corner of the source object.
<i>mouse_button</i>	A constant that specifies the mouse button to hold down while dragging. The value can be LEFT, MIDDLE, or RIGHT. If no button is specified, the default is the button that performs the select function. This optional parameter is available for WinRunner only.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

obj_drop

Context Sensitive • Object

drops an object onto a target object.

obj_drop (*target_object*, *x*, *y*);

<i>target_object</i>	The logical name of the GUI object. The object may belong to any class.
<i>x</i> , <i>y</i>	The <i>x, y</i> coordinates of the pointer when released over the target object, relative to the upper left corner of the target object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

obj_exists**Context Sensitive • Object**

checks whether an object is displayed on the screen.

obj_exists (*object* [, *time*]);

<i>object</i>	The logical name of the object. The object may belong to any class.
<i>time</i>	The amount of time (in seconds) that is added to the default timeout setting (specified with the <i>timeout_msec</i> testing option), yielding a new maximum wait time before the subsequent statement is executed.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

obj_find_text

Context Sensitive • Object

returns the location of a string within an object.

obj_find_text (*object*, *string*, *result_array* [, *search_area* [, *string_def*]]);

<i>object</i>	The logical name of the object. The object may belong to any class.
<i>string</i>	A valid string expression or the name of a string variable, which can include a regular expression. The regular expression should not include an exclamation mark (!), however, which is treated as a literal character.
<i>result_array</i>	The name of the four-element array that stores the location of the string. The elements are numbered 1 to 4. Elements 1 and 2 store the x- and y-coordinates of the upper left corner of the enclosing rectangle; elements 3 and 4 store the coordinates for the lower right corner.
<i>search_area</i>	Indicates the area of the screen to search as coordinates that define any two diagonal corners of a rectangle, expressed as a pair of x,y coordinates. The coordinates are stored in <i>result_array</i> .
<i>string_def</i>	Defines the type of search to perform. If no value is specified (0 or FALSE, the default), the search is for a single, complete word only. When 1, or TRUE, is specified, the search is not restricted to a single, complete word.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

obj_get_desc

Context Sensitive • Object

returns an object's physical description.

obj_get_desc (*object*, *oblig*, *optional*, *selector*, *out_desc*);

<i>object</i>	The logical name of the GUI object. The object may belong to any class.
<i>oblig</i>	The list of obligatory properties (separated by blank spaces).
<i>optional</i>	The list of optional properties (separated by blank spaces).
<i>selector</i>	The type of selector used for this object class (location or index).
<i>out_desc</i>	The output variable that stores the description of the GUI object.

Return Values

If the *oblig*, *optional*, and *selector* parameters are null strings, **obj_get_desc** returns the current learning configuration for the object.

Availability

This function is always available.

obj_get_info

Context Sensitive • Object

returns the value of an object property.

obj_get_info (*object*, *property*, *out_value*);

<i>object</i>	The logical name of the GUI object. The object may belong to any class.
<i>property</i>	Any of the properties listed in the <i>User's Guide</i> .
<i>out_value</i>	The output variable that stores the value of the property.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

obj_get_text**Context Sensitive • Object**

reads text from an object.

obj_get_text (*object*, *out_text* [, *x*₁, *y*₁, *x*₂, *y*₂]);

<i>object</i>	The logical name of the GUI object. The object may belong to any class.
<i>out_text</i>	The name of the output variable that stores the captured text.
<i>x</i> ₁ , <i>y</i> ₁ , <i>x</i> ₂ , <i>y</i> ₂	An optional parameter that defines the location from which text will be read, relative to the specified object. The pairs of coordinates can designate any two diagonally opposite corners of a rectangle.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

obj_highlight

Context Sensitive • Object

highlights an object.

obj_highlight (*object* [, *flashes*]);

<i>object</i>	The logical name of the object. The object may belong to any class.
<i>flashes</i>	The number of times the object flashes. The default number is four.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

obj_mouse_click

Context Sensitive • Object

clicks on an object.

obj_mouse_click (*object*, *x*, *y* [, *mouse_button* [, *modifier*]]);

<i>object</i>	The logical name of the object. The object may belong to any class.
<i>x</i> , <i>y</i>	The position of the mouse click expressed as <i>x</i> and <i>y</i> (pixel) coordinates. Coordinates are relative to the upper left corner of the GUI object.
<i>mouse_button</i>	A constant that specifies the mouse button to click. The value can be LEFT, MIDDLE, or RIGHT. If no button is specified, the default is the button that performs the select function.
<i>modifier</i>	A constant that specifies the modifier key used with the mouse button. The value can be CONTROL, SHIFT, or CONTROL_SHIFT.

Note: When running a test with an `obj_mouse_click` statement, the object that the mouse clicks must be fully displayed.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

`obj_mouse_dbl_click`

Context Sensitive • Object

performs a double-click within an object.

`obj_mouse_dbl_click (object, x, y [, mouse_button [, modifier]]);`

<i>object</i>	The logical name of the GUI object. The object may belong to any class.
<i>x, y</i>	The position of the double-click expressed as x and y (pixel) coordinates. Coordinates are relative to the upper left corner of the GUI object.
<i>mouse_button</i>	A constant that specifies the mouse button to click. The value can be LEFT, MIDDLE, or RIGHT. If no button is specified, the default is the button that performs the select function.
<i>modifier</i>	A constant that specifies the modifier key used with the mouse button. The value can be CONTROL, SHIFT, or CONTROL_SHIFT.

Note: When running a test with an `obj_mouse_dbl_click` statement, the object that the mouse clicks must be fully displayed.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

obj_mouse_drag**Context Sensitive • Object**

drags the mouse within an object.

obj_mouse_drag (*object*, *start_x*, *start_y*, *end_x*, *end_y* [, *mouse_button* [, *modifier*]]);

<i>object</i>	The logical name of the object. The object may belong to any class.
<i>start_x</i> , <i>start_y</i>	The x and y coordinates of the start point of the mouse drag. The coordinates are relative to the upper left corner of the GUI object.
<i>end_x</i> , <i>end_y</i>	The x and y coordinates of the end point of the mouse drag. The coordinates are relative to the upper left corner of the GUI object.
<i>mouse_button</i>	A constant that specifies the mouse button to hold down. The value can be LEFT, MIDDLE, or RIGHT. If no button is specified, the default is the button that performs the select function.
<i>modifier</i>	A constant that specifies the modifier key used with the mouse button. The value can be CONTROL, SHIFT, or CONTROL_SHIFT.

Note: When running a test with an **obj_mouse_drag** statement, the object that the mouse drags must be fully displayed.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

obj_mouse_move

Context Sensitive • Object

moves the mouse pointer within an object.

obj_mouse_move (*object*, *x*, *y*);

<i>object</i>	The logical name of the GUI object. The object may belong to any class.
<i>x</i> , <i>y</i>	The position of the mouse pointer, expressed as x and y (pixel) coordinates. Note that the specified coordinates are relative to the upper left corner of the object. This position is relative to the upper left corner of the object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

obj_move_locator_text

Context Sensitive • Object

moves the mouse pointer to a string in an object.

```
obj_move_locator_text ( object, string [ , search_area [ , string_def ] ] );
```

<i>object</i>	The logical name of the object.
<i>string</i>	The text to locate. To specify a literal, case sensitive string, enclose the string in quotation marks. Alternatively, you can specify the name of a string variable. The value of the string variable can include a regular expression (the regular expression need not begin with an exclamation mark).
<i>search_area</i>	The region of the object to search, relative to the window. This area is defined as a pair of coordinates, with x_1, y_1, x_2, y_2 specifying any two diagonally opposite corners of the rectangular search region. If this parameter is not defined, then the entire <i>object</i> is considered the search area.
<i>string_def</i>	Defines how the text search is performed. If no <i>string_def</i> is specified, (0 or FALSE, the default parameter), the interpreter searches for a complete word only. If 1, or TRUE, is specified, the search is not restricted to a single, complete word.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

obj_set_info

Context Sensitive • Object

sets the value of an object property.

obj_set_info (*object*, *property*, *out_value*);

<i>object</i>	The logical name of the object. The object may belong to any class.
<i>property</i>	Any of the properties listed in the <i>User's Guide</i> .
<i>out_value</i>	The output variable that stores the value of the property.

Return Values

This function returns one of the standard return values. It returns `E_ATTR_NOT_SUPPORTED` for a specified property (for example, value) if one of the following events occur:

- The object does not have the method `setValue`.
- The method `setValue` exists, but it either has more than one parameter or the parameter does not belong to one of the following Java classes: `String`, `int`, `boolean`, `Integer`, `Boolean`.
- The parameter given in a TSL call statement cannot be converted to one of the Java classes mentioned above.
- The method `setValue` throws a Java exception when using the parameters provided in the call statement.

Availability

This function is always available.

obj_type

Context Sensitive • Object

sends keyboard input to an object.

obj_type (*object*, *keyboard_input*);

<i>object</i>	The logical name of the GUI object.
<i>keyboard_input</i>	A string expression that represents keystrokes.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

obj_wait_bitmap

Context Sensitive • Object

waits for an object bitmap to be drawn on the screen.

obj_wait_bitmap (*window*, *bitmap*, *time* [, *x*, *y*, *width*, *height*]);

<i>object</i>	The logical name of the object. The object may belong to any class.
<i>bitmap</i>	A string expression that identifies the captured bitmap.
<i>time</i>	Indicates the interval between the previous input event and the capture of the current bitmap, in seconds. This parameter is added to the <i>timeout_msec</i> testing option and the sum indicates how much time WinRunner will wait for the capture of the bitmap.
<i>x</i> , <i>y</i>	For an area bitmap: the coordinates of the upper left corner, relative to the object in which the selected region is located.
<i>width</i> , <i>height</i>	For an area bitmap: the size of the selected region, in pixels.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

obj_wait_info

Context Sensitive • Object

waits for the value of an object property.

obj_wait_info (*object*, *property*, *value*, *time*);

<i>object</i>	The logical name of the object.
<i>property</i>	Any of the properties listed in the <i>User's Guide</i> .
<i>value</i>	The property value for which the function waits.
<i>time</i>	The interval, in seconds, before the next statement is executed.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

output_message

Standard • Load Testing

sends a message to the controller.

output_message (*message*);

message Any string.

The **output_message** function sends a message from a Vuser script to the controller's Output window.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118

Availability

This function is available for LoadRunner GUI Vusers only.

password_edit_set

Standard • Password

sets the value of a password edit field to a given value.

password_edit_set (*edit_object*, *encrypted_password*);

edit_object The logical name of the edit object.

encrypted_password The encrypted password as it appears in the script.

Note: You can also use the **edit_set**, **type**, and **obj_type** TSL functions to set a password, however the **password_edit_set** function provides extra security by eliminating the password from the test script.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

password_encrypt

Context Sensitive • Password

encrypts a plain password.

password_encrypt (*password*);

password The plain password.

Return Values

This function returns the encrypted password.

Availability

This function is always available.

pause

Standard • I/O

pauses test execution and displays a message box.

pause ([*expression*]);

expression Any valid expression.

Return Values

This function always returns 0.

Availability

This function is always available.

phone_append_text

Context Sensitive • WAP

appends the specified text string to the current contents of the phone editor.

phone_append_text (*text*);

text The text string to append in the phone editor.

Note: This function works only while the phone is in editing mode. Trying to use this function while the phone is not in editing mode will return an illegal operation.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for the WAP Add-in. This function is supported for both the Nokia and Phone.com emulators.

phone_edit_set

Context Sensitive • WAP

replaces the contents of the phone editor with the specified text string.

phone_edit_set (*text*);

text The text string to insert in the phone editor.

Note: This function works only while the phone is in editing mode. Trying to use this function while the phone is not in editing mode will return an illegal operation.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for the WAP Add-in. This function is supported for both the Nokia and Phone.com emulators.

phone_get_name

Context Sensitive • WAP

returns the model name of the phone.

phone_get_name (*name*);

name The model name of the phone.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for the WAP Add-in. This function is supported for both the Nokia and Phone.com emulators.

phone_GUI_load

Context Sensitive • WAP

unloads the currently loaded GUI map file and loads the GUI map for the specified Phone.com phone.

phone_GUI_load ([*name*]);

name The model name of the phone.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for the WAP Add-in. This function is supported for the Phone.com emulator only.

phone_key_click**Context Sensitive • WAP**

clicks a phone key.

phone_key_click (*key* [, *delay* [, *timeout*]]);

<i>key</i>	The logical name of the phone key.
<i>delay</i>	The Boolean parameter indicating that there is an additional delay to compensate for inserting a new letter while editing.
<i>timeout</i>	The amount of time (in milliseconds) between pressing and releasing the key.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for the WAP Add-in. This function is supported for both the Nokia and Phone.com emulators.

phone_navigate**Context Sensitive • WAP**

directs the phone to connect to the specified site.

phone_navigate (*URL* [, *timeout*]);

<i>URL</i>	The URL to which the phone navigates.
<i>timeout</i>	The amount of time (in milliseconds) the phone waits while trying to establish a connection.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for the WAP Add-in. This function is supported for both the Nokia and Phone.com emulators.

phone_sync

Context Sensitive • WAP

recorded after any phone navigation on the Nokia emulator, and instructs WinRunner to wait until the phone is ready to handle the next operation.

phone_sync ([*redirect* [, *timeout*]]);

<i>redirect</i>	An optional Boolean parameter indicating that the phone will wait an additional amount of time to redirect to another URL.
<i>timeout</i>	The amount of time (in milliseconds) that the phone will wait to try to establish a connection.

Note: This function is inserted automatically to the test scripts after a **phone_key_click** statement is recorded on a Nokia phone that included navigation. The timeout is the expected period of time during which WinRunner expects the navigation to be concluded.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for the WAP Add-in.

This function is supported for recording on the Nokia emulator only. This function is supported for running tests on both the Nokia and the phone.com emulators.

Tip: If you want to run the same test on both the Nokia and phone.com emulators, you should record tests on the Nokia emulator.

qt_force_send_key

Standard • QuickTest 2000

instructs WinRunner with the Year 2000 add-in to recognize an edit field which prompts a screen change when information is inserted.

qt_force_send_key (*window_name*, *field_name* [, *additional_key*]);

<i>window_name</i>	The name of the window.
<i>field_name</i>	The name of the edit field.
<i>additional_key</i>	The key which causes the screen change.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for QuickTest 2000 only.

qt_reset_all_force_send_key

Standard • QuickTest 2000

negates screen change configurations previously made using the **qt_force_send_key** function.

qt_reset_all_force_send_key ();

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

rand

Standard • Arithmetic

returns a pseudo-random floating point number (n) in the range of $0 \leq n < 1$.

rand ();

Return Values

This function returns a real number.

Availability

This function is always available.

reload

Standard • Compiled Module

removes a compiled module from memory and loads it again.

reload (*module_name* [,1|0 [,1|0]]);

<i>module_name</i>	A string expression indicating the name of an existing compiled module.
1 0	1 indicates a system module. 0 indicates a user module. The default values is 0.
1 0	This parameter is optional and only implemented if the second parameter is implemented. 1 indicates that a user module will not remain open after it is loaded. 0 indicates that the module remains open in the WinRunner window. The default value is 0.

Note: If you make changes to a function in a loaded compiled module, you must unload and reload the compiled module in order for the changes to take effect. For additional information, refer to the “Creating Compiled Modules” chapter in the *WinRunner User’s Guide*.

Return Values

This function returns 0 for success, and 1 for failure.

Availability

This function is always available.

rendezvous**Standard • Load Testing**

sets a rendezvous point in a Vuser script.

rendezvous (*rendezvous_name*);

rendezvous_name The name of the rendezvous declared in a **declare_rendezvous** statement.

Return Value

This function returns 0 if the operation is successful, or one of the following error codes if it fails:

Error code	Number	Description
E_OK	0	operation successful
E_TIMEOUT	-10016	timeout reached before operation performed
E_REND_NF	-10218	rendezvous not defined
E_REND_NOT_MEM	-10219	vuser not defined as a participant in the rendezvous
E_REND_INVALID	-10220	rendezvous disabled

Availability

This function is available for LoadRunner GUI Vusers only.

report_msg

Standard • I/O

writes a message in the test report.

```
report_msg ( message );
```

message A valid string expression.

Return Values

This function always returns 0.

Availability

This function is always available.

return

Standard • Call Statements

returns an expression to the calling function or test.

```
return [ expression ];
```

expression The expression to return.

The **return** statement returns an expression to the calling function or test. It is used exclusively in functions. It also halts execution of the called function and passes control back to the calling function or test.

Note about arrays: You cannot return an array from a function. In order to return values in an array, you must declare the array as an OUT parameter in the function.

The return value of a function can be one of the following:

- char (signed and unsigned)
 - string (equivalent to C char*)
 - short (signed and unsigned)
 - int (signed and unsigned)
 - long (signed and unsigned)
 - float
 - double
-

Return Values

If no expression is used, then an empty string is returned. Otherwise, the return statement does not have a return value.

Availability

This statement is always available.

Note: The **return** statement is not a function. Therefore, it does not appear in the Function Generator.

scroll_check_info

Context Sensitive • Scroll Object

checks the value of a scroll property.

scroll_check_info (*scroll*, *property*, *property_value*);

<i>scroll</i>	The logical name of the scroll.
<i>property</i>	The property to be checked.
<i>property_value</i>	The expected property value.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

scroll_check_pos

Context Sensitive • Scroll Object

checks the current position of a scroll.

scroll_check_pos (*scroll*, *position*);

<i>scroll</i>	The logical name of the scroll.
<i>position</i>	A number indicating the expected scroll position.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

scroll_drag

Context Sensitive • Scroll Object.

scrolls to the specified location.

scroll_drag (*scroll*, *orientation*, *position*);

<i>scroll</i>	The logical name of the scroll.
<i>orientation</i>	The direction of the scroll; either VSCROLL (vertical) or HSCROLL (horizontal).
<i>position</i>	The absolute position within the scroll.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function can be used for scroll bar and slider objects.

scroll_drag_from_min

Context Sensitive • Scroll Object

scrolls from the minimum position.

scroll_drag_from_min (*scroll*, *orientation*, *position*);

<i>scroll</i>	The logical name of the scroll object.
<i>orientation</i>	The direction of the scroll; either VSCROLL (vertical) or HSCROLL (horizontal).
<i>position</i>	The number of units from the minimum position to drag the scroll.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function can be used for scroll bar and slider objects.

scroll_get_info

Context Sensitive • Scroll Object

returns the value of a scroll property.

scroll_get_info (*scroll*, *property*, *out_value*);

<i>scroll</i>	The logical name of the scroll.
<i>property</i>	Any of the properties listed in the <i>WinRunner User's Guide</i> .
<i>out_value</i>	The output variable that stores the value of the specified property.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function can be used for scroll bar and slider objects.

scroll_get_max

Context Sensitive • Scroll Object

returns the maximum (end) position of a scroll.

scroll_get_max (*scroll*, *orientation*, *out_max*);

<i>scroll</i>	The logical name of the scroll.
<i>orientation</i>	The direction of the scroll; either VSCROLL (vertical) or HSCROLL (horizontal).
<i>out_max</i>	The output variable which stores the maximum value of the scroll.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function can be used for scroll bar and slider objects.

scroll_get_min

Context Sensitive • Scroll Object

returns the minimum (start) position of a scroll.

scroll_get_min (*scroll*, *orientation*, *out_min*);

<i>scroll</i>	The logical name of the scroll.
<i>orientation</i>	The direction of the scroll; either VSCROLL (vertical) or HSCROLL (horizontal).
<i>out_min</i>	The output variable that stores the minimum (starting) value of the scroll.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function can be used for scroll bar and slider objects.

scroll_get_pos

Context Sensitive • Scroll Object

returns the current scroll position.

scroll_get_pos (*scroll*, *orientation*, *out_pos*);

<i>scroll</i>	The logical name of the scroll.
<i>orientation</i>	The direction of the scroll; either VSCROLL (vertical) or HSCROLL (horizontal).
<i>out_pos</i>	The output variable which stores the current position of the scroll.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function can be used for scroll bar and slider objects.

scroll_get_selected

Context Sensitive • Scroll Object

returns the minimum and maximum values of the selected range on a slider.

scroll_get_selected (*slider*, *min_value*, *max_value*);

<i>slider</i>	The logical name of the slider.
<i>min_value</i>	The output variable that stores the minimum value of the selected range.
<i>max_value</i>	The output variable that stores the maximum value of the selected range.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

The scroll_get_selected function works only for slider objects, for which the TBS_ENABLESELRANGE flag is set. This flag allows a selection range within the scroll to be displayed.

scroll_line

Context Sensitive • Scroll Object

scrolls the specified number of lines.

scroll_line (*scroll*, *orientation*, [*+*|-] *lines*);

<i>scroll</i>	The logical name of the scroll.
<i>orientation</i>	The direction of the scroll; either VSCROLL (vertical) or HSCROLL (horizontal).
[<i>+</i> -] <i>lines</i>	The number of scrolled lines. "+" indicates the scroll is performed downward or to the right; "-" indicates the scroll is performed upward or to the left. The default is "+".

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function can be used for scroll bar and slider objects.

scroll_max**Context Sensitive • Scroll Object**

sets a scroll to its maximum (end) position.

scroll_max (*scroll*, *orientation*);

<i>scroll</i>	The logical name of the scroll.
<i>orientation</i>	The direction of the scroll; either VSCROLL (vertical) or HSCROLL (horizontal).

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function can be used for scroll bar and slider objects.

scroll_min**Context Sensitive • Scroll Object**

sets the scroll to its minimum (start) position.

scroll_min (*scroll*, *orientation*);

<i>scroll</i>	The logical name of the scroll object.
<i>orientation</i>	The direction of the scroll; either VSCROLL (vertical) or HSCROLL (horizontal).

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function can be used for scroll bar and slider objects.

scroll_page

Context Sensitive • Scroll Object

moves the scroll the specified number of pages.

scroll_page (*scroll*, *orientation*, [*+|-*] *pages*);

<i>scroll</i>	The logical name of the scroll.
<i>orientation</i>	The direction of the scroll; either VSCROLL (vertical) or HSCROLL (horizontal).
[<i>+ -</i>] <i>pages</i>	The number of scrolled pages. "+" indicates that the scroll is performed downward or to the right; "-" indicates that the scroll is performed upward or to the left. The default is '+'.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function can be used for scroll bar and slider objects.

scroll_wait_info

Context Sensitive • Scroll Object

waits for the value of a scroll property.

scroll_wait_info (*scroll*, *property*, *value*, *time*);

<i>scroll</i>	The logical name of the scroll.
<i>property</i>	Any of the properties listed in the <i>WinRunner User's Guide</i> .
<i>value</i>	The property value.
<i>time</i>	The interval, in seconds, before the next statement is executed.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function can be used for scroll bar and slider objects.

set_aut_var**Standard • Testing Option**

sets how WinRunner learns descriptions of objects, records tests, and runs tests on Java applets or applications.

set_aut_var (*variable*, *value*);

variable The variable to set.

value The value of the variable.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available only for WinRunner with Java support.

set_class_map**Context Sensitive • GUI Map Configuration**

associates a custom class with a standard class.

set_class_map (*custom_class*, *standard_class*);

custom_class The name of the custom class used in the application.

standard_class The name of the Mercury class or the MS Windows standard class with the same behavior as the custom class.

Note: You should store **set_class_map** statements in a startup test.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner and GUI Vusers running on PC platforms only.

set_record_attr

Context Sensitive • GUI Map Configuration

sets the properties to learn for an object class.

set_record_attr (*class*, *oblig_prop*, *optional_prop*, *selector*);

<i>class</i>	The name of the Mercury class, MSW_class, or X_class.
<i>oblig_prop</i>	A list of properties (separated by blank spaces) to always learn.
<i>optional_prop</i>	A list of descending properties (separated by blank spaces) to add to the description until unique identification of the object is achieved.
<i>selector</i>	The type of selector to be applied in case both obligatory and optional properties do not achieve a unique object identification. This may be either index or location.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

set_record_method

Context Sensitive • GUI Map Configuration

specifies the record method for a class.

set_record_method (*class*, *method*);

class The name of a standard class, MSW_class, or TOOLKIT_class.

method The record method to use, as described in the table below.

Method	Description
RM_RECORD	Records operations using Context Sensitive functions. This is the default method for all the standard classes, except the object class (for which the default is MIC_MOUSE).
RM_IGNORE	Turns off recording.
RM_PASSUP	Records mouse operations (relative to the parent of the object) and keyboard input.
RM_AS_OBJECT	Records all windows or objects as general “object” class objects (obj_mouse_click or win_mouse_click).

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

set_window

Context Sensitive • Window Object

specifies the window to receive subsequent input and (optionally) specifies the amount of time to wait for the specified window.

set_window (*window* [,*time*]);

<i>window</i>	The logical name of the window.
<i>time</i>	The amount of time, in seconds, added to the timeout option (set in the Run tab of the Settings > General Options dialog box) to give the maximum interval before the next statement is executed (WinRunner). If the Window is found before the maximum time is reached, the test continues to run.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

_set_window

Context Sensitive • Window Object

specifies a window to receive input.

_set_window (*desc*, *time*);

<i>desc</i>	The physical description of the window.
<i>time</i>	The time is added to the <i>timeout_msec</i> testing option to give the maximum interval, in seconds, before the next statement is executed.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

setvar**Standard • Testing Option**

sets the value of a testing option.

setvar (*option*, *value*);

option A testing option.

value The value to assign to the testing option.

The **setvar** function changes the value of a testing option. For a list and an in-depth explanations of **setvar** options, refer to the “Setting Testing Options from a Test Script” chapter in the *WinRunner User's Guide*.

Return Values

This function always returns 0.

Availability

This function is always available.

siebel_click_history**Context Sensitive • Siebel**

clicks the Siebel History button.

siebel_click_history (*thread_bar_object*);

thread_bar_object The logical name of the Siebel bar object containing the History button.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Siebel support.

siebel_connect_repository

Context Sensitive • Siebel

connects to the Siebel repository database.

siebel_connect_repository (*connection_string*);

<i>connection_string</i>	The string that activates the connection to the Siebel repository database.
--------------------------	---

Note: You only need to call this function once per connection.

If you encounter difficulties connecting the repository using an existing DSN, use the ODBC Data Source Administrator from the Windows Control Panel to define a new User Data Source (DSN) that refers to the Siebel Repository database.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Siebel support.

siebel_get_active_applet

Context Sensitive • Siebel

returns the active applet name.

siebel_get_active_applet (*applet_name*);

<i>applet_name</i>	The output variable that stores the name of the active applet.
--------------------	--

Note: A **set_window** statement must precede this function in order to direct the input to the correct application window.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Siebel support.

siebel_get_active_buscomp**Context Sensitive • Siebel**

returns the active business component name.

siebel_get_active_buscomp (*bus_comp_name*);

<i>bus_comp_name</i>	The output variable that stores the name of the active business component.
----------------------	--

Note: A **set_window** statement must precede this function in order to direct the input to the correct application window.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Siebel support.

siebel_get_active_busobj

Context Sensitive • Siebel

returns the active business object name.

siebel_get_active_busobj (*bus_obj_name*);

<i>bus_obj_name</i>	The output variable that stores the name of the active business object.
---------------------	---

Note: A **set_window** statement must precede this function in order to direct the input to the correct application window.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Siebel support.

siebel_get_active_control

Context Sensitive • Siebel

returns the active control name.

siebel_get_active_control (*control_name*);

<i>control_name</i>	The output variable that stores the name of the active control.
---------------------	---

Notes: This function makes it possible to use the **siebel_get_control_value** and **siebel_set_control_value** functions. A **set_window** statement must precede this function in order to direct the input to the correct application window.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Siebel support.

siebel_get_active_view**Context Sensitive • Siebel**

returns the active view name.

siebel_get_active_view (*view_name*);

<i>view_name</i>	The output variable that stores the name of the active View object.
------------------	---

Note: A **set_window** statement must precede this function in order to direct the input to the correct application window.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Siebel support.

siebel_get_chart_data

Context Sensitive • Siebel

returns the legend data and chart values from the specified chart.

siebel_get_chart_data (*chart_object*, *ret_legend_array*, *ret_values_array*);

<i>chart_object</i>	The logical name of the chart or the chart's legend.
<i>ret_legend_array</i>	The output variable that stores the array of legend elements.
<i>ret_values_array</i>	The output variable that stores the array of chart values.

Note: Either the legend or the chart may be selected, and that both will return the same data.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Siebel support.

siebel_get_control_value

Context Sensitive • Siebel

returns the value of the active control.

siebel_get_control_value (*value*);

<i>value</i>	The output variable that stores the value of the active control.
--------------	--

Note: The **siebel_set_active_control** function must precede this statement in order to establish the active control.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Siebel support.

siebel_goto_record

Context Sensitive • Siebel

navigates to the specified record.

siebel_goto_record (*direction*);

<i>direction</i>	The direction in which to move to get to the desired record from the current location. Possible values are: "First", "Last", "Previous", or "Next".
------------------	---

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Siebel support.

siebel_navigate_view

Context Sensitive • Siebel

navigates to the specified view.

siebel_navigate_view (*view_name*);

<i>view_name</i>	The internal name of the view to be reached.
------------------	--

Note: Navigation is sensitive to the record context.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Siebel support.

siebel_obj_get_info

Context Sensitive • Siebel

returns the value of a single Siebel property from the Siebel repository database.

siebel_obj_get_info (*obj_type, obj_name, applet_name, property_name, ret_prop_val*);

<i>obj_type</i>	The Siebel type for which to retrieve the attribute.
-----------------	--

Possible values for this parameter are:

S_APPLET, S_BUSCOMP, S_BUSOBJ, S_CONTROL, S_FIELD, or S_VIEW

<i>obj_name</i>	The internal object name for which to retrieve the attribute.
-----------------	---

<i>applet_name</i>	The applet name
--------------------	-----------------

Required only with *obj_type*: CONTROL or FIELD. For all other *obj_types*, enter "".

<i>property_name</i>	The name of the property to retrieve.
----------------------	---------------------------------------

ret_prop_val The output variable that stores the value of the specified object property.

Note: You must connect to the Siebel repository database with a **siebel_connect_repository** statement before you use this function.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Siebel support.

siebel_obj_get_properties

Context Sensitive • Siebel

returns all properties of a specified Siebel in the Siebel repository database.

siebel_obj_get_properties (*obj_type*, *obj_name*, *applet_name*, *ret_prop_array*);

<i>obj_type</i>	The Siebel type for which to retrieve the properties. Possible values for this parameter are: S_APPLET, S_BUSCOMP, S_BUSOBJ, S_CONTROL, S_FIELD, or S_VIEW
<i>obj_name</i>	The internal object name for which to retrieve the properties.
<i>applet_name</i>	The applet name. Required only with <i>obj_type</i> : CONTROL or FIELD. For all other <i>obj_types</i> , enter "".
<i>ret_prop_array</i>	The output variable that stores the array of values for the specified object property.

Note: You must connect to the Siebel repository database with a `siebel_connect_repository` statement before you use this function.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Siebel support.

siebel_select_alpha

Context Sensitive • Siebel

selects a letter key from the alpha tab bar.

siebel_select_alpha (*alpha_tab_object*, *key*);

alpha_tab_object The logical name of the alpha tab object; usually "alpha tab".

key The letter key to select from the alpha tab.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Siebel support.

siebel_set_active_applet

Context Sensitive • Siebel

sets the specified applet as the active applet.

siebel_set_active_applet (*applet_name*);

applet_name The internal name of the of the applet to activate.

If you do not know the applet's internal name, you may use the **siebel_get_active_applet** to retrieve it.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is supported for WinRunner with Siebel support.

siebel_set_active_control

Context Sensitive • Siebel

sets the specified control as the active control.

siebel_set_active_control (*control_name*);

control_name The internal name of the control to activate.

If you do not know the control's internal name, you can use the **siebel_get_active_applet** function to retrieve it.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is supported for WinRunner with Siebel support.

siebel_set_control_value

Context Sensitive • Siebel

sets the value of the active control.

siebel_set_control_value (*new_value*);

new_value The value to be assigned to the active control.

Note: The **siebel_set_active_control** function must precede this statement in order to establish the active control.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Siebel support.

siebel_terminate

Context Sensitive • Siebel

closes the Siebel application.

siebel_terminate ();

Note: Call this function to terminate the Siebel application or immediately after manually closing the application.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Siebel support.

sin**Standard • Arithmetic**

calculates the sine of an angle expressed in radians.

sin (*x*);

Return Values

This function returns a real number.

Availability

This function is always available.

spin_down**Context Sensitive • Spin Object**

scrolls a spin control down a specified number of times.

spin_down (*spin_obj*, *spins*);

spin_obj The name of the spin control.

spins The number of times the control is moved down.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Visual Basic support.

spin_get_info

Context Sensitive • Spin Object

returns the value of a spin property.

spin_get_info (*spin*, *property*, *out_value*);

<i>spin</i>	The logical name of the spin object.
<i>property</i>	Any of the properties listed in the <i>User's Guide</i> .
<i>out_value</i>	The output variable that stores the value of the specified property.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

spin_get_pos

Context Sensitive • Spin Object

returns the current position of a spin object.

spin_get_pos (*spin*, *out_value*);

<i>spin</i>	The logical name of the spin object.
<i>property</i>	Any of the properties listed in the <i>User's Guide</i> .
<i>out_value</i>	The output variable that stores the value of the specified property.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

spin_get_range

Context Sensitive • Spin Object

returns the minimum and maximum positions of a spin object.

spin_get_range (*spin*, *out_min_pos*, *out_max_pos*);

<i>spin</i>	The logical name of the spin object.
<i>out_min_pos</i>	The output variable that stores the minimum position of the spin object.
<i>out_max_pos</i>	The output variable that stores the maximum position of the spin object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

spin_max

Context Sensitive • Spin Object

sets a spin object to its maximum value.

spin_max (*spin*);

<i>spin</i>	The logical name of the spin object.
-------------	--------------------------------------

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

spin_min

Context Sensitive • Spin Object

sets a spin object to its minimum value.

spin_min (*spin*);

spin The logical name of the spin object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

spin_next

Context Sensitive • Spin Object

sets a spin object to its next value.

spin_next (*spin* [, *index*]);

spin The logical name of the spin object.

index The number of the text field in the spin object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

spin_prev

Context Sensitive • Spin Object

sets a spin object to its previous value.

spin_prev (*spin*);

spin The logical name of the spin object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

spin_set

Context Sensitive • Spin Object

sets a spin object to an item.

spin_set (*spin*, *item*);

spin The logical name of the spin object.

item The item to select in the spin object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

spin_up

Context Sensitive • Spin Object

scrolls a spin control up the specified number of times.

spin_up (*spin_obj*, *spins*);

<i>spin_obj</i>	The name of the spin control.
<i>spins</i>	The number of times the control is moved up.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Visual Basic support.

spin_wait_info

Context Sensitive • Spin Object

waits for a spin property to attain a specified value.

spin_wait_info (*spin*, *property*, *value*, *time*);

<i>spin</i>	The logical name of the spin.
<i>property</i>	Any of the properties listed in the <i>WinRunner User's Guide</i> .
<i>value</i>	The property value for which the function waits.
<i>time</i>	The interval, in seconds, before the next statement is executed.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

split

Standard • Array

divides an input string into fields and stores them in an array.

split (*string*, *array* [, *field_separators*]);

<i>string</i>	A valid string expression.
<i>array</i>	The name of the storage array.
<i>field_separators</i>	The characters in the string which designate where the string is to be split into fields. Each single character is used as a separator.

Note: The first element in the array index is numbered 1. The number of elements in the array equals the result of the split. As in any array, they are sequential integers.

Return Values

This function returns the number of elements in the array.

Availability

This function is always available.

sprintf

Standard • I/O

returns a formatted string to a variable.

sprintf (*format*, *exp*₁, *exp*₂, ... *exp*_{*n*});

<i>format</i>	May include both a literal string to be printed and formatting specifications.
<i>exp</i>	The expressions to format.

Return Values

This function returns a formatted string.

start_transaction

Standard • Load Testing

marks the beginning of a transaction for performance analysis.

start_transaction (*transaction_name*);

<i>transaction_name</i>	A string expression that names the transaction. The string must not contain any spaces.
-------------------------	---

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for LoadRunner GUI Vusers only.

static_check_info

Context Sensitive • Static Text Object

checks the value of a static text object property.

static_check_info (*static*, *property*, *property_value*);

<i>static</i>	The logical name of the static text object.
<i>property</i>	The property to check.
<i>property_value</i>	The expected property value.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

static_check_text

Context Sensitive • Static Text Object

checks the content of a static text object.

static_check_text (*static*, *text*, *case_sensitive*);

<i>static</i>	The logical name of the static text object.
<i>text</i>	The contents of the static text object.
<i>case_sensitive</i>	Indicates whether the comparison is case sensitive. This value is either TRUE or FALSE.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

static_get_info

Context Sensitive • Static Text Object

returns the value of a static text object property.

static_get_info (*static*, *property*, *out_value*);

<i>static</i>	The logical name of the static text object.
<i>property</i>	Any of the properties listed in the <i>User's Guide</i> .
<i>out_value</i>	The output variable that stores the value of the specified property.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

static_get_text

Context Sensitive • Static Text Object

returns the contents of a static text object.

static_get_text (*static*, *out_string*);

<i>static</i>	The logical name of the static text object.
<i>out_string</i>	The output variable that stores the string found in the static text object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

static_wait_info

Context Sensitive • Static Text Object

waits for the value of a static text object property.

static_wait_info (*static*, *property*, *value*, *time*);

<i>static</i>	The logical name of the static text object.
<i>property</i>	Any of the properties listed in the <i>User’s Guide</i> .
<i>value</i>	The expected property value.
<i>time</i>	The maximum interval, in seconds, before the next statement is executed.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

statusbar_get_field_num

Context Sensitive • Statusbar

returns the numeric index of a field on a status bar.

statusbar_get_field_num (*statusbar*, *field*, *field_index*);

<i>statusbar</i>	The logical name of the status bar.
<i>field</i>	The text in the status bar field. If the text in the field changes, you can use a regular expression.
<i>field_index</i>	The output variable that stores the numeric index of the field. Note that the first field in the status bar is numbered 0.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

statusbar_get_info

Context Sensitive • Statusbar

returns the value of a status bar property.

statusbar_get_info (*statusbar*, *property*, *out_value*);

<i>statusbar</i>	The logical name of the status bar.
<i>property</i>	The following properties may be specified: <i>abs_x</i> , <i>abs_y</i> , <i>active</i> , <i>attached_text</i> , <i>class</i> , <i>count</i> , <i>displayed</i> , <i>enabled</i> , <i>focus</i> , <i>handle</i> , <i>height</i> , <i>label</i> , <i>MSW_class</i> , <i>MSW_id</i> , <i>nchildren</i> , <i>parent_value</i> (default), <i>width</i> , <i>x</i> , <i>y</i>
<i>out_value</i>	The output variable that stores the value of the specified property.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

statusbar_get_text**Context Sensitive • Statusbar**

reads text from a field on a status bar.

statusbar_get_text (*statusbar*, *field_index*, *out_text*);

<i>statusbar</i>	The logical name of the status bar.
<i>field_index</i>	The index number of the field containing the text you want to read. The first field in the status bar is numbered 0.
<i>out_text</i>	The name of the output variable that stores the text.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

statusbar_wait_info**Context Sensitive • Statusbar**

waits for the value of a status bar property.

statusbar_wait_info (*statusbar*, *property*, *value*, *time*);

<i>statusbar</i>	The logical name of the status bar.
<i>property</i>	The property to wait for. The following properties may be specified: <i>abs_x</i> , <i>abs_y</i> , <i>active</i> , <i>attached_text</i> , <i>class</i> , <i>count</i> , <i>displayed</i> , <i>enabled</i> , <i>focus</i> , <i>handle</i> , <i>height</i> , <i>label</i> , <i>MSW_class</i> , <i>MSW_id</i> , <i>nchildren</i> , <i>parent</i> , <i>value</i> (default), <i>width</i> , <i>x</i> , <i>y</i>
<i>value</i>	The property value.
<i>time</i>	Indicates the interval, in seconds, before the next statement is executed.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

str_map_logical_to_visual**Standard • I/O**

converts a logical string to a visual string or vice-versa.

str_map_logical_to_visual (*logical_string*, *visual_string*);

logical_string A valid logical string expression.

visual_string The corresponding returned valid visual string expression.

The **str_map_logical_to_visual** function returns a valid visual string expression for a valid logical string expression. Alternatively, it returns a valid logical string expression for a valid visual string expression.

Note: This function is primarily intended for use with RTL-style windows. When working with applications with RTL-style windows, the **get_text** function sometimes returns a logical string instead of a visual string.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

substr

Standard • String

extracts a substring from a string.

substr (*string*, *position* [, *length*]);

<i>string</i>	A valid string expression.
<i>position</i>	An integer that indicates the position of the first character of the substring. The position of the first character of the string is 1, the second is 2, etc.
<i>length</i>	Defines the number of characters (starting from <i>position</i>) to include in the substring.

Return Values

This function returns a string. If the value of *position* is greater than the length of the specified string, then the function returns the null string.

Availability

This function is always available.

system

Standard • Operating System

executes an operating system command.

system (*expression*);

<i>expression</i>	A string expression that specifies the system command to execute.
-------------------	---

Return Values

The return value of the function is the value of the operating system command executed.

Availability

This function is available for WinRunner and LoadRunner GUI Vusers on UNIX platforms. The **system** function is also supported on other platforms for purposes of porting and backward compatibility.

tab_get_info

Context Sensitive • Tab Object

returns the value of a tab property.

tab_get_info (*tab*, *property*, *out_value*);

<i>tab</i>	The logical name of the tab object.
<i>property</i>	Any of the properties listed in the <i>WinRunner User's Guide</i> .
<i>out_value</i>	The output variable that stores the value of the specified property.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

tab_get_item

Context Sensitive • Tab Object

returns the name of a tab item.

tab_get_item (*tab*, *item_num*, *out_item*);

<i>tab</i>	The logical name of the tab.
<i>item_num</i>	The location of the tab item. Note that the first tab item in a property sheet is numbered 0.
<i>out_item</i>	The output variable that stores the tab name.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

tab_get_selected

Context Sensitive • Tab Object

returns the name and number of the selected tab item.

tab_get_selected (*tab*, *out_item*, *out_num*);

<i>tab</i>	The logical name of the tab.
<i>out_item</i>	The output variable that stores the name of the selected tab item. Note that the first tab item in a property sheet is numbered 0.
<i>out_num</i>	The output variable that stores the index of the selected tab item.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

tab_select_item

Context Sensitive • Tab Object

selects a tab item.

tab_select_item (*tab*, *item*);

<i>tab</i>	The logical name of the tab.
<i>item</i>	The item to select. The item can be denoted by either its name or its numeric index. The index is specified as a string preceded by the character #. The first tab item is numbered 0.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

tab_wait_info

Context Sensitive • Tab Object

waits for the value of a tab property.

tab_wait_info (*tab*, *property*, *value*, *time*);

<i>tab</i>	The logical name of the tab.
<i>property</i>	Any of the properties listed in the <i>User's Guide</i> .
<i>value</i>	The property value for which the function waits.
<i>time</i>	The maximum interval, in seconds, before the next statement is executed.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

tbl_activate_cell**Context Sensitive • Table**

double-clicks the specified cell in a table.

tbl_activate_cell (*table*, *row*, *column*);

<i>table</i>	The logical name of the table.
<i>row</i>	<p>By location: # <row_location> The location of the row within the table, specified by a string preceded by the character #, such as "#2".</p> <p>For WinRunner with PowerBuilder support, the <i>row</i> can also be in the following format:</p> <p>By content: <Column_name>=<column_content₁ [column_content_n,...]></p> <p>The contents of one or more cells in the row, separated by semicolons and preceded by the name of the column in which they appear and an equal sign, such as "Flight_Number=306;From=LAX". The contents of all the cells specified must be present in order to specify the row. Choose this format to specify a row by the contents of cells in that row. If the contents of some cells appear in multiple rows, specify multiple cells whose contents will uniquely identify the row.</p>
<i>column</i>	<p>The <i>column</i> can be either:</p> <p>By location: # <column_location> The location of the column within the table, specified by a string preceded by the character #, such as "#2".</p> <p>By content: <Column_name> The column name, such as "Flight_Number". When the column name is specified, WinRunner takes the name from the database itself, and not from the application.</p>

Note for PowerBuilder users: When *row* is specified **by content**, *column* must also be specified **by content**.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is not supported for WebTest.

This function is supported for WinRunner with Java support. It is supported for the following Java toolkit packages: JFC, EWT (Oracle), and KLG.

This function is supported for WinRunner with PowerBuilder or Siebel support.

This function is supported for the following ActiveX controls:

ActiveX Control	MSW_class
Data Bound Grid Control	MSDBGrid.DBGrid
FarPoint Spreadsheet Control	FPSpread.Spread.1
MicroHelp MH3d List Control	MHGLBX.Mh3dListCtrl.1
Microsoft Grid Control	MSGrid.Grid
Sheridan Data Grid Control	SSDataWidgets.SSDBGridCtrl.1
True DBGrid Control	TrueDBGrid50.TDBGrid, TrueDBGrid60.TDBGrid, and TrueOleDBGrid60.TDBGrid

tbl_activate_col

Context Sensitive • Table

double-clicks the specified column in a table.

tbl_activate_col (*table*, *column*);

table The logical name of the table.

column The *column* can be either:

By location: # <column_location> The location of the column within the table, specified by a string preceded by the character #, such as "#2".

By content: <Column_name> The column name, such as "Flight_Number".

When the column name is specified, WinRunner takes the name from the database itself, and not from the application.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118, and "Return Values for PowerBuilder and Table Functions," on page 123.

Availability

This function is available only for WinRunner with Java support. It is supported for the following Java toolkit packages: JFC and KLG.

tbl_activate_header

Context Sensitive • Table

double-clicks the specified column header in a table.

tbl_activate_header (*table*, *column*);

table The logical name of the table.

column The *column* can be either:

By location: # <column_location> The location of the column within the table, specified by a string preceded by the character #, such as "#2".

By content: <Column_name> The column name, such as "Flight_Number".

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118, and "Return Values for PowerBuilder and Table Functions," on page 123.

Availability

This function is not supported for WebTest.

This function is supported for WinRunner with Siebel support.

This function is supported for the following ActiveX controls:

ActiveX Control	MSW_class
Data Bound Grid Control	MSDBGrid.DBGrid
FarPoint Spreadsheet Control	FPSpread.Spread.1
MicroHelp MH3d List Control	MHGLBX.Mh3dListCtrl.1
Microsoft Grid Control	MSGrid.Grid
Sheridan Data Grid Control	SSDataWidgets.SSDBGridCtrl.1
True DBGrid Control	TrueDBGrid50.TDBGrid, TrueDBGrid60.TDBGrid, and TrueOleDBGrid60.TDBGrid

tbl_activate_row**Context Sensitive • Table**

double-clicks the specified row in a table.

tbl_activate_row (*table*, *row*);

<i>table</i>	The logical name of the table.
<i>row</i>	The <i>row</i> can be either By location: # <column_location> The location of the row within the table, specified by a string preceded by the character #, such as "#2". By content: <row_name> The row name, such as "Flight_2".

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available for WinRunner with Java support. It is supported for the following Java toolkit packages: JFC, Visual Cafe, and KLG.

This function is supported for WinRunner with Siebel support.

tbl_click_cell**Analog • Table**

clicks in a cell in a JFC JTable object.

tbl_click_cell (*table_name*, *cell_index*, *column_name* [, *mouse_button*]);

<i>table_name</i>	The name of the table.
<i>cell_index</i>	An index number denoting the position of the cell in the column. The index number is preceded by #, for example #2.
<i>column_name</i>	The name of the column in which the cell is located.
<i>mouse_button</i>	The mouse button used to click on the cell (optional).

Note: WinRunner records this function only after the **set_aut_var** function is used to set the TABLE_RECORD_METHOD variable to ANALOG.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available only for WinRunner with Java Add-in support.

tbl_dbl_click_cell

Analog • Table

double-clicks in a cell in a JFC JTable object.

tbl_dbl_click_cell (*table_name*, *cell_index*, *column_name* [, *mouse_button*]);

<i>table_name</i>	The name of the table.
<i>cell_index</i>	An index number denoting the position of the cell in the column. The index number is preceded by #, for example #2.
<i>column_name</i>	The name of the column in which the cell is located.
<i>mouse_button</i>	The mouse button used to click on the cell (optional).

Note: WinRunner records this function only after the **set_aut_var** function is used to set the TABLE_RECORD_METHOD variable to ANALOG.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available only for WinRunner with Java Add-in support.

tbl_deselect_col**Context Sensitive • Table**

deselects the specified column in a table.

tbl_deselect_col (*table*, *column*);

table

The logical name of the table.

column

The *column* can be either:

By location: # <column_location> The location of the column within the table, specified by a string preceded by the character #, such as "#2".

By content: <column_name> The column name, such as "Flight_Number".

When the column name is specified, WinRunner takes the name from the database itself, and not from the application.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118, and "Return Values for PowerBuilder and Table Functions," on page 123.

Availability

This function is available only for WinRunner with Java support. It is supported for the JFC Java toolkit package.

tbl_deselect_cols_range

Context Sensitive • Table

deselects the specified range of columns in a table.

tbl_deselect_cols_range (*table*, *from_column*, *to_column*);

<i>table</i>	The logical name of the table.
<i>from_column</i>	The <i>from_column</i> can be either: By location: # <column_location> The location of the column within the table, specified by a string preceded by the character #, such as "#2". By content: <column_name> The column name, such as "Flight_Number". When a column name is specified, WinRunner takes the name from the database itself, and not from the application.
<i>to_column</i>	The <i>to_column</i> can be either: By location: # <column_location> The location of the column within the table, specified by a string preceded by the character #, such as "#2". By content: <column_name> The column name, such as "Flight_Number". When a column name is specified, WinRunner takes the name from the database itself, and not from the application.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118, and "Return Values for PowerBuilder and Table Functions," on page 123.

Availability

This function is available only for WinRunner with Java support. It is supported for the JFC Java toolkit package.

tbl_deselect_row

Context Sensitive • Table

deselects the specified row in a table.

tbl_deselect_row (*table*, *row*);

table The logical name of the table.

row The *row* can be either:

By location: # <row_location> The location of the row within the table, specified by a string preceded by the character #, such as "#2".

By content: <row_name> The row name, such as "Flight_2".

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available only for WinRunner with Java Add-in support. It is supported for the following Java toolkit packages: JFC, Visual Cafe, and KLG.

tbl_deselect_rows_range

Context Sensitive • Table

deselects the specified range of rows in a table.

tbl_deselect_rows_range (*table*, *from_row*, *to_row*);

table The logical name of the table.

from_row The *from_row* can be either:

By location: # <row_location> The location of the row within the table, specified by a string preceded by the character #, such as "#2".

By content: <row_name> The row name, such as "Flight_2".

*to_row*The *to_row* can be either:

By location: # <row_location> The location of the row within the table, specified by a string preceded by the character #, such as "#2".

By content: <row_name> The row name, such as "Flight_2".

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available only for WinRunner with Java Add-in support. It is supported for the following Java toolkit packages: JFC and Visual Cafe.

tbl_drag

Analog • Table

drags a cell to a different location within a JFC JTable object.

tbl_drag (*table_name*, *start_row*, *start_col*, *end_row*, *end_col* [, *mouse_button*]);

table_name

The name of the table.

start_row

The row name or an index number denoting the row which contains the cell before the drag operation is performed. The index number is preceded by #, for example #3.

start_col

The column name or an index number denoting the column which contains the cell before the drag operation is performed. The index number is preceded by #, for example #2.

end_row

The row name or an index number denoting the row which contains the cell after the drag operation is performed. The index number is preceded by #, for example #5.

<i>end_col</i>	The column name or an index number denoting the column which contains the cell after the drag operation is performed. The index number is preceded by #, for example #7.
<i>mouse_button</i>	The mouse button used to drag the cell (optional).

Note: WinRunner records this function only after the **set_aut_var** function is used to set the TABLE_RECORD_METHOD variable to ANALOG.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available only for WinRunner with Java Add-in support.

tbl_extend_col

Context Sensitive • Table

adds a column to the currently selected columns in a table.

tbl_extend_col (*table*, *column*);

table The logical name of the table.

column The column can be either:

By location: # <column_location> The location of the column within the table, specified by a string preceded by the character #, such as "#2".

By content: <Column_name> The column name, such as "Flight_Number".

When the column name is specified, WinRunner takes the name from the database itself, and not from the application.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available only for WinRunner with Java Add-in support. It is supported for the JFC Java toolkit package.

tbl_extend_cols_range

Context Sensitive • Table

adds columns to the currently selected columns in a table.

tbl_extend_cols_range (*table*, *from_column*, *to_column*);

table The logical name of the table.

from_column The *from_column* can be either:

By location: # <column_location> The location of the column within the table, specified by a string preceded by the character #, such as "#2".

By content: <column_name> The column name, such as "Flight_Number".

When a column name is specified, WinRunner takes the name from the database itself, and not from the application.

to_column The *to_column* can be either:

By location: # <column_location> The location of the column within the table, specified by a string preceded by the character #, such as "#2".

By content: <column_name> The column name, such as "Flight_Number".

When a column name is specified, WinRunner takes the name from the database itself, and not from the application.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available only for WinRunner with Java Add-in support. It is supported for the JFC Java toolkit package.

tbl_extend_row

Context Sensitive • Table

adds a row to the currently selected rows in a table.

tbl_extend_row (*table*, *row*);

table The logical name of the table.

row The *row* can either:

By location: # <row_location> The location of the row within the table, specified by a string preceded by the character #, such as "#2".

By content: <row_name> The row name, such as "Flight_2".

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available only for WinRunner with Java Add-in support. It is supported for the following Java toolkit packages: JFC, Visual Cafe, and KLG.

tbl_extend_rows_range

Context Sensitive • Table

adds rows to the currently selected rows in a table.

tbl_extend_rows_range (*table*, *from_row*, *to_row*);

<i>table</i>	The logical name of the table.
<i>from_row</i>	The <i>from_row</i> can be either: By location: # <row_location> The location of the row within the table, specified by a string preceded by the character #, such as "#2". By content: <row_name> The row name, such as "Flight_2".
<i>to_row</i>	The <i>to_row</i> can be either: By location: # <row_location> The location of the row within the table, specified by a string preceded by the character #, such as "#2". By content: <row_name> The row name, such as "Flight_2".

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available only for WinRunner with Java Add-in support. It is supported for the following Java toolkit packages: JFC and Visual Cafe.

tbl_get_cell_data

Context Sensitive • Table

retrieves the contents of the specified cell from a table.

tbl_get_cell_data (*table*, *row*, *column*, *out_text*);

table

The logical name of the table.

row

By location: # <row_location> The location of the row within the table, specified by a string preceded by the character #, such as "#2".

For WinRunner with PowerBuilder or Java, support, the *row* can also be in the following format:

By content: <Column_name>=<column_content1 [column_contentn....]>

The contents of one or more cells in the row, separated by semicolons and preceded by the name of the column in which they appear and an equal sign, such as "Flight_Number=306;From=LAX". The contents of all the cells specified must be present in order to specify the row. Choose this format to specify a row by the contents of cells in that row. If the contents of some cells appear in multiple rows, specify multiple cells whose contents will uniquely identify the row.

column

By location: # <column_location> The location of the column within the table, specified by a string preceded by the character #, such as "#2".

By content: <Column_name> The column name, such as "Flight_Number". When the column name is specified, WinRunner takes the name from the database itself, and not from the application.

out_text

For WinRunner with Oracle, Java, or WebTest support, *out_text* is the output variable that stores the string found in the specified cell.

For WinRunner with PowerBuilder support, *out_text* is the output variable that stores the string found in the specified cell; the actual string retrieved depends on the style of the cell, as follows:

DropDown: The name of the item selected.

Radio Button: The label of the selected radio button in the cell. (PowerBuilder only)

Edit: The contents of the cell.

EditMask: The contents of the cell.

Checkbox: Either "OFF" or "ON".

Note: The maximum table size supported by WinRunner is 327,680 bytes. If the table is larger than this, the value of the *out_text* parameter may be "!" or "Null".

Note for PowerBuilder, Java, and WebTest support users: When row is specified by content, column must also be specified by content.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118, and "Return Values for PowerBuilder and Table Functions," on page 123.

Availability

This function is available for WinRunner the Java Add-in support. It is supported for the following Java toolkit packages: JFC, Visual Cafe, EWT (Oracle), and KLG.

This function is supported for WebTest and for WinRunner with Oracle, PowerBuilder, or Siebel support.

This function is supported for the following ActiveX controls:

ActiveX Control	MSW_class
Data Bound Grid Control	MSDBGrid.DBGrid
FarPoint Spreadsheet Control	FPSpread.Spread.1
MicroHelp MH3d List Control	MHGLBX.Mh3dListCtrl.1
Microsoft Grid Control	MSGrid.Grid
Sheridan Data Grid Control	SSDataWidgets.SSDBGridCtrl.1
True DBGrid Control	TrueDBGrid50.TDBGrid, TrueDBGrid60.TDBGrid, and TrueOleDBGrid60.TDBGrid

tbl_get_cols_count

Context Sensitive • Table

retrieves the number of columns in a table.

tbl_get_cols_count (*table*, *out_cols_count*);

table The logical name of the table.

out_cols_count The output variable that stores the total number of columns in the table.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available for WinRunner with Java Add-in support. It is supported for the following Java toolkit packages: JFC, Visual Cafe, EWT (Oracle), and KLG.

This function is supported for WebTest and for WinRunner with Oracle, PowerBuilder, or Siebel support.

This function is supported for the following ActiveX controls:

ActiveX Control	MSW_class
Data Bound Grid Control	MSDBGrid.DBGrid
FarPoint Spreadsheet Control	FPSpread.Spread.1
MicroHelp MH3d List Control	MHGLBX.Mh3dListCtrl.1
Microsoft Grid Control	MSGrid.Grid
Sheridan Data Grid Control	SSDataWidgets.SSDBGridCtrl.1
True DBGrid Control	TrueDBGrid50.TDBGrid, TrueDBGrid60.TDBGrid, and TrueOleDBGrid60.TDBGrid

tbl_get_column_name

Context Sensitive • Table

retrieves the column header name of the specified column in a table.

tbl_get_column_name (*table*, *col_index*, *out_col_name*);

<i>table</i>	The logical name of the table.
<i>col_index</i>	The numeric index of the column within the table, specified by an integer.
<i>out_col_name</i>	The parameter into which the retrieved name is stored.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available for WinRunner with Java Add-in support. It is supported for the following Java toolkit packages: JFC, Visual Cafe, and KLG.

This function is supported for WebTest and for WinRunner with Siebel support.

This function is supported for the following ActiveX controls:

ActiveX Control	MSW_class
Data Bound Grid Control	MSDBGrid.DBGrid
FarPoint Spreadsheet Control	FPSpread.Spread.1
MicroHelp MH3d List Control	MHGLBX.Mh3dListCtrl.1
Microsoft Grid Control	MSGrid.Grid
Sheridan Data Grid Control	SSDataWidgets.SSDBGridCtrl.1
True DBGrid Control	TrueDBGrid50.TDBGrid, TrueDBGrid60.TDBGrid, and TrueOleDBGrid60.TDBGrid

tbl_get_column_names

Context Sensitive • Table

retrieves the names and number of columns in a table.

tbl_get_column_names (*table*, *out_col_names*, *out_cols_count*);

<i>table</i>	The name of the table.
<i>out_col_names</i>	The output variable that stores the names of the columns in the table.
<i>out_cols_count</i>	The output variable that stores the total number of columns in the table.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is supported only for WinRunner with PowerBuilder support. The corresponding function for WinRunner without PowerBuilder support is **tbl_get_column_name**.

This function is not supported for WebTest.

tbl_get_rows_count

Context Sensitive • Table

retrieves the number of rows in the specified table.

tbl_get_rows_count (*table*, *out_rows_count*);

<i>table</i>	The logical name of the table.
<i>out_rows_count</i>	The output variable that stores the total number of rows in the table.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available for WinRunner with Java Add-in support. It is supported for the following Java toolkit packages: JFC, Visual Cafe, EWT (Oracle), and KLG.

This function is supported for WebTest and for WinRunner with Oracle, PowerBuilder or Siebel support.

This function is supported for the following ActiveX controls:

ActiveX Control	MSW_class
FarPoint Spreadsheet Control	FPSpread.Spread.1
MicroHelp MH3d List Control	MHGLBX.Mh3dListCtrl.1
Microsoft Grid Control	MSGrid.Grid
Sheridan Data Grid Control	SSDataWidgets.SSDBGridCtrl.1
True DBGrid Control	TrueDBGrid50.TDBGrid, TrueDBGrid60.TDBGrid, and TrueOleDBGrid60.TDBGrid

tbl_get_selected_cell

Context Sensitive • Table

returns the cell currently in focus in a table.

tbl_get_selected_cell (*table*, *out_row*, *out_column*);

<i>table</i>	The logical name of the table.
<i>out_row</i>	<p>By location: # <row_location> The location of the row within the table, specified by a string preceded by the character #, such as "#2".</p> <p>For WinRunner with PowerBuilder support, the <i>out_row</i> can also be in the following format:</p> <p>By content: <Column_name>=<column_content1 [column_contentn....]></p> <p>The contents of one or more cells in the row, separated by semicolons and preceded by the name of the column in which they appear and an equal sign, such as "Flight_Number=306;From=LAX". The contents of all the cells specified must be present in order to specify the row. Choose this format to specify a row by the contents of cells in that row. If the contents of some cells appear in multiple rows, specify multiple cells whose contents will uniquely identify the row.</p>
<i>out_column</i>	The output variable that stores the column name of the cell.

Note for PowerBuilder users: When *out_row* is specified **by content**, *out_column* must also be specified **by content**.

Note for Java add-in users: When using this function for Java tables, the row and column parameters are returned as numeric indexes (without the # character).

Note: The column name is taken from the database itself and not from the application.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available for WinRunner with Java Add-in support. It is supported for the following Java toolkit packages: JFC, EWT (Oracle), and KLG.

This function is supported for WebTest and for WinRunner with PowerBuilder, Oracle, or Siebel support.

This function is supported for the following ActiveX controls:

ActiveX Control	MSW_class
Data Bound Grid Control	MSDBGrid.DBGrid
FarPoint Spreadsheet Control	FPSpread.Spread.1
MicroHelp MH3d List Control	MHGLBX.Mh3dListCtrl.1
Microsoft Grid Control	MSGrid.Grid
Sheridan Data Grid Control	SSDataWidgets.SSDBGridCtrl.1
True DBGrid Control	TrueDBGrid50.TDBGrid, TrueDBGrid60.TDBGrid, and TrueOleDBGrid60.TDBGrid

tbl_get_selected_row

Context Sensitive • Table

returns the row currently selected in a table.

tbl_get_selected_row (*table*, *row*);

table

The logical name of the table.

row

By location: # <row_location> The location of the row within the table, specified by a string preceded by the character #, such as "#2".

For WinRunner with PowerBuilder support, the *row* can also be in the following format:

By content: <Column_name>=<column_content1 [column_contentn....]>

The contents of one or more cells in the row, separated by semicolons and preceded by the name of the column in which they appear and an equal sign, such as "Flight_Number=306;From=LAX". The contents of all the cells specified must be present in order to specify the row. Choose this format to specify a row by the contents of cells in that row. If the contents of some cells appear in multiple rows, specify multiple cells whose contents will uniquely identify the row.

For WinRunner with PowerBuilder support, *row* specifies the selected row following the row whose index is specified in the function.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118, and "Return Values for PowerBuilder and Table Functions," on page 123.

Availability

This function is available for WinRunner with Java Add-in support. It is supported for the following Java toolkit packages: JFC, Visual Cafe, and KLG.

This function is supported for WinRunner with PowerBuilder, Oracle or Siebel support.

This function is not supported for WebTest.

This function is supported for the following ActiveX controls:

ActiveX Control	MSW_class
Data Bound Grid Control	MSDBGrid.DBGrid
FarPoint Spreadsheet Control	FPSpread.Spread.1
MicroHelp MH3d List Control	MHGLBX.Mh3dListCtrl.1
Sheridan Data Grid Control	SSDataWidgets.SSDBGridCtrl.1
True DBGrid Control	TrueDBGrid50.TDBGrid, TrueDBGrid60.TDBGrid, and TrueOleDBGrid60.TDBGrid

tbl_select_cells_range

Context Sensitive • Table

clicks the specified range of cells in a table.

tbl_select_cells_range (*table*, *start_row*, *start_col*, *end_row*, *end_col*);

table The logical name of the table.

start_row The *start_row* can be either:

By location: # <row_location> The location of the row within the table, specified by a string preceded by the character #, such as "#2".

By content: <row_name> The row name, such as "Flight_2".

start_col The *start_column* can be either:

By location: # <column_location> The location of the column within the table, specified by a string preceded by the character #, such as "#2".

By content: <Column_name> The column name, such as "Flight_Number".

<i>end_row</i>	<p>The <i>end_row</i> can be either:</p> <p>By location: # <row_location> The location of the row within the table, specified by a string preceded by the character #, such as "#2".</p> <p>By content: <row_name> The row name, such as "Flight_2"</p>
<i>end_col</i>	<p>The <i>end_column</i> can be either:</p> <p>By location: # <column_location> The location of the column within the table, specified by a string preceded by the character #, such as "#2".</p> <p>By content: <Column_name> The column name, such as "Flight_Number".</p>

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available only for WinRunner with Java support. It is supported for the following Java toolkit packages: JFC and KLG

tbl_select_col_header

Context Sensitive • Table

selects the specified column header of a table.

tbl_select_col_header (*table*, *column*);

<i>table</i>	The logical name of the table.
<i>column</i>	<p>The <i>column</i> can be either:</p> <p>By location: # <column_location> The location of the column within the table, specified by a string preceded by the character#, such as "#2".</p> <p>By content: <Column_name> The column name, such as "Flight_Number".</p>

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available for WinRunner with Siebel or Java add-in support. It is supported for the following Java toolkit packages: JFC, Visual Cafe, and KLG.

This function is not supported for WebTest.

This function is supported for the following ActiveX controls:

ActiveX Control	MSW_class
Data Bound Grid Control	MSDBGrid.DBGrid
FarPoint Spreadsheet Control	FPSpread.Spread.1
MicroHelp MH3d List Control	MHGLBX.Mh3dListCtrl.1
Microsoft Grid Control	MSGrid.Grid
Sheridan Data Grid Control	SSDataWidgets.SSDBGridCtrl.1
True DBGrid Control	TrueDBGrid50.TDBGrid, TrueDBGrid60.TDBGrid, and TrueOleDBGrid60.TDBGrid

tbl_select_cols_range

Context Sensitive • Table

clicks the specified range of columns in a table.

tbl_select_cols_range (*table*, *from_column*, *to_column*);

table The logical name of the table.

from_column The *from_column* can be either:

By location: # <column_location> The location of the column within the table, specified by a string preceded by the character #, such as "#2".

By content: <column_name> The column name, such as "Flight_Number".

When a column name is specified, WinRunner takes the name from the database itself, and not from the application.

to_column The *to_column* can be either:

By location: # <column_location> The location of the column within the table, specified by a string preceded by the character #, such as "#2".

By content: <column_name> The column name, such as "Flight_Number".

When a column name is specified, WinRunner takes the name from the database itself, and not from the application.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available only for WinRunner with Java Add-in support. It is supported for the following Java toolkit packages: JFC and KLG.

tbl_select_rows_range

Context Sensitive • Table

selects the specified range of rows in a table.

tbl_select_rows_range (*table*, *from_row*, *to_row*);

<i>table</i>	The logical name of the table.
<i>from_row</i>	<p>The <i>from_row</i> can be either:</p> <p>By location: # <row_location> The location of the row within the table, specified by a string preceded by the character #, such as "#2".</p> <p>By content: <row_name> The row name, such as "Flight_2".</p> <p>When a row name is specified, WinRunner takes the name from the database itself, and not from the application.</p>
<i>to_row</i>	<p>The <i>to_row</i> can be either:</p> <p>By location: # <row_location> The location of the row within the table, specified by a string preceded by the character #, such as "#2".</p> <p>By content: <row_name> The row name, such as "Flight_2".</p> <p>When a row name is specified, WinRunner takes the name from the database itself, and not from the application.</p>

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118, and "Return Values for PowerBuilder and Table Functions," on page 123.

Availability

This function is available only for WinRunner with Java Add-in support. It is supported for the following Java toolkit packages: JFC, Visual Cafe, and KLG.

tbl_set_cell_data

Context Sensitive • Table

sets the contents of a cell to the specified text in a table.

tbl_set_cell_data (*table*, *row*, *column*, *data*);

table

The logical name of the table.

row

By location: # <row_location>

The location of the row within the table, specified by a string preceded by the character #, such as "#2".

For WinRunner with PowerBuilder, Java, or WebTest support, the *row* can also be in the following format:

By content: <Column_name>=<column_content1 [column_contentn....]>

The contents of one or more cells in the row, separated by semicolons and preceded by the name of the column in which they appear and an equal sign, such as "Flight_Number=306;From=LAX". The contents of all the cells specified must be present in order to specify the row. Choose this format to specify a row by the contents of cells in that row. If the contents of some cells appear in multiple rows, specify multiple cells whose contents will uniquely identify the row.

column

By location: # <column_location> The location of the column within the table, specified by a string preceded by the character #, such as "#2".

By content: <Column_name> The column name, such as "Flight_Number".

data

For WinRunner with Oracle, Java, or WebTest support, the *data* is a string denoting the contents to be entered into the specified cell.

For WinRunner with PowerBuilder support, *data* is a string denoting the contents to be entered into the specified cell; the nature of the string depends on the style of the cell, as follows:

DropDown DataWindow: The name of the item selected.

Radio Button: The label of the selected radio button in the cell.

Edit: The contents of the cell.

EditMask: The contents of the cell.

Checkbox: Either "OFF" or "ON".

Note for PowerBuilder users: When *row* is specified by **content**, *column* must also be specified by **content**.

When a column name is specified, WinRunner takes the name from the database itself and not from the application.

For a column with a DropDown DataWindow style, *data* can specify the contents of any of the columns, and not only the one that is displayed in the table. (See the example below.) For a column with a DropDown DataWindow or DropDown list style, the item can be a string denoting the row number of the cell, preceded by the character #.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118, and "Return Values for PowerBuilder and Table Functions," on page 123.

Availability

This function is available for WinRunner with Java Add-in support. It is supported for the following Java toolkit packages: JFC, EWT (Oracle), and KLG.

This function is not supported for WebTest.

This function is supported for WinRunner with PowerBuilder, Oracle, or Siebel support.

This function is supported for the following ActiveX controls:

ActiveX Control	MSW_class
Data Bound Grid Control	MSDBGrid.DBGrid
FarPoint Spreadsheet Control	FPSpread.Spread.1
MicroHelp MH3d List Control	MHGLBX.Mh3dListCtrl.1
Microsoft Grid Control	MSGrid.Grid
Sheridan Data Grid Control	SSDataWidgets.SSDBGridCtrl.1
True DBGrid Control	TrueDBGrid50.TDBGrid, TrueDBGrid60.TDBGrid, and TrueOleDBGrid60.TDBGrid

tbl_set_cell_focus

Context Sensitive • Table

sets the focus to the specified cell in a table.

tbl_set_cell_focus (*table*, *row*, *column*);

table The logical name of the table.

row The column can be either:

By location: # <row_location>

The location of the row within the table, specified by a string preceded by the character #, such as "#2".

By content: <column_name>=<column_content1
[column_contentn....]>

The contents of one or more cells in the row, separated by semicolons and preceded by the name of the column in which they appear and an equal sign, such as "Flight_Number=306;From=LAX". The contents of all the cells specified must be present in order to specify the row. Choose this format to specify a row by the contents of cells in that row. If the contents of some cells appear in multiple rows, specify multiple cells whose contents will uniquely identify the row. If the values match more than one row WinRunner refers to the first matching row.

column

The column can be either:

By location: # <column_location> The location of the column within the table, specified by a string preceded by the character #, such as "#2".

By content: <column_name> The column name, such as "Flight_Number".

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118, and "Return Values for PowerBuilder and Table Functions," on page 123.

Availability

This function is supported only for WinRunner with Siebel support.

tbl_set_selected_cell

Context Sensitive • Table

selects (clicks) the specified cell in a table.

tbl_set_selected_cell (*table*, *row*, *column*);

<i>table</i>	The logical name of the table.
<i>row</i>	<p>By location: # <row_location> The location of the row within the table, specified by a string preceded by the character #, such as "#2".</p> <p>For WinRunner with PowerBuilder support, the <i>row</i> can also be in the following format:</p> <p>By content: <Column_name>=<column_content1 [column_contentn....]></p> <p>The contents of one or more cells in the row, separated by semicolons and preceded by the name of the column in which they appear and an equal sign, such as "Flight_Number=306;From=LAX". The contents of all the cells specified must be present in order to specify the row. Choose this format to specify a row by the contents of cells in that row. If the contents of some cells appear in multiple rows, specify multiple cells whose contents will uniquely identify the row.</p>
<i>column</i>	<p>By location: # <column_location> The location of the column within the table, specified by a string preceded by the character #, such as "#2".</p> <p>By content: <Column_name> The column name, such as "Flight_Number". When a column name is specified, WinRunner takes the name from the database itself and not from the application.</p>

Note for PowerBuilder users: When *row* is specified **by content**, *column* must also be specified **by content**.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for PowerBuilder and Table Functions,” on page 123.

Availability

This function is available for WinRunner with Java Add-in support. It is supported for the following Java toolkit packages: JFC, Visual Cafe, EWT (Oracle), and KLG.

This function is not supported for WebTest.

This function is supported for WinRunner with PowerBuilder, Oracle, or Siebel support.

This function is supported for the following ActiveX controls:

ActiveX Control	MSW_class
Data Bound Grid Control	MSDBGrid.DBGrid
FarPoint Spreadsheet Control	FPSpread.Spread.1
MicroHelp MH3d List Control	MHGLBX.Mh3dListCtrl.1
Microsoft Grid Control	MSGrid.Grid
Sheridan Data Grid Control	SSDataWidgets.SSDBGridCtrl.1
True DBGrid Control	TrueDBGrid50.TDBGrid, TrueDBGrid60.TDBGrid, and TrueOleDBGrid60.TDBGrid

tbl_set_selected_col

Context Sensitive • Table

selects the specified column in a table.

tbl_set_selected_col (*table*, *column*);

table The logical name of the table.

column The *column* can be either:

By location: # <column_location> The location of the column within the table, specified by a string preceded by the character #, such as "#2".

By content: <Column_name> The column name, such as "Flight_Number".

When a column name is specified, WinRunner takes the name from the database itself, and not from the application.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118, and "Return Values for PowerBuilder and Table Functions," on page 123.

Availability

This function is available only for WinRunner with Java Add-in support. It is supported for the following Java toolkit packages: JFC and EWT (Oracle).

tbl_set_selected_row

Context Sensitive • Table

selects the specified row in a table.

tbl_set_selected_row (*table*, *row*);

table

The logical name of a table.

row

By location: # <row_location>

The location of the row within the table, specified by a string preceded by the character #, such as "#2".

For WinRunner with PowerBuilder support, the *row* can also be in the following format:

By content: <Column_name>=<column_content₁
[column_content_n...]>

The contents of one or more cells in the row, separated by semicolons and preceded by the name of the column in which they appear and an equal sign, such as "Flight_Number=306;From=LAX". The contents of all the cells specified must be present in order to specify the row. Choose this format to specify a row by the contents of cells in that row. If the contents of some cells appear in multiple rows, specify multiple cells whose contents will uniquely identify the row.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118, and "Return Values for PowerBuilder and Table Functions," on page 123.

Availability

This function is available for WinRunner with Java Add-in support. It is supported for the following Java toolkit packages: JFC, Visual Cafe, EWT (Oracle), and KLG.

This function is not supported for WebTest.

This function is supported for WinRunner with PowerBuilder, Oracle, or Siebel support.

This function is supported for the following ActiveX controls:

ActiveX Control	MSW_class
Data Bound Grid Control	MSDBGrid.DBGrid
FarPoint Spreadsheet Control	FPSpread.Spread.1
MicroHelp MH3d List Control	MHGLBX.Mh3dListCtrl.1
Microsoft Grid Control	MSGrid.Grid
True DBGrid Control	TrueDBGrid50.TDBGrid, TrueDBGrid60.TDBGrid, and TrueOleDBGrid60.TDBGrid

tddb_get_step_value

Standard • TestDirector

returns the value of a field in the "dessteps" table in a TestDirector project database.

tddb_get_step_value (*field*, *step_index* [, *td_path*]);

<i>field</i>	The logical name of the field.
<i>step_index</i>	The index of the step.
<i>td_path</i>	The TestDirector test path (optional argument - the default is the current test).

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

The **tddb_get_step_value** function is only available when WinRunner is connected to a TestDirector project database.

tddb_get_test_value

Standard • TestDirector

returns the value of a field in the "test" table in a TestDirector project database.

tddb_get_test_value (*field* [, *td_path*]);

<i>field</i>	The logical name of the field.
<i>td_path</i>	The TestDirector test path (optional argument - the default is the current test).

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

The **tddb_get_test_value** function is only available when WinRunner is connected to a TestDirector project database.

tddb_get_testset_value

Standard • TestDirector

returns the value of a field in the "testcycl" table in a TestDirector project database.

tddb_get_testset_value (*field* [, *td_path* [, *test_set*]]);

<i>field</i>	The logical name of the field.
<i>td_path</i>	The TestDirector test path (optional argument - the default is the current test).
<i>test_set</i>	The name of the test_set (optional argument - the default is the current TestSet).

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

The **tddb_load_attachment** function is only available when WinRunner is connected to a TestDirector project database.

tddb_load_attachment

Standard • TestDirector

loads a test's file attachment and returns the file system path of the location where it was loaded.

tddb_load_attachment (*attachment*, *path*);

<i>attachment</i>	The name of the file attachment.
<i>path</i>	The system path of the location where it was loaded.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

The **tddb_load_attachment** function is only available when WinRunner is connected to a TestDirector project database.

TE_add_screen_name_location

Context Sensitive • Terminal Emulator

adds a screen name location.

TE_add_screen_name_location (*x*, *y*, *length*);

<i>x</i>	The x-coordinate of the new area to search.
<i>y</i>	The y-coordinate of the new area to search.
<i>length</i>	The number of characters to the right of the Y position that WinRunner will search for a string.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_bms2gui

Context Sensitive • Terminal Emulator

teaches WinRunner the user interface from a BMS file.

TE_bms2gui (*bms_filename*, *gui_filename*, LEARN|RELEARN);

<i>bms_filename</i>	The full path of the BMS file containing the description of the application's user interface.
<i>gui_file_name</i>	The full path of the GUI map file into which the descriptions are learned. If no file name is given, the default is the temporary GUI map file of the test.
LEARN RELEARN	Instructs WinRunner how to deal with name/description conflicts in the BMS file.

Return Values

This function has no return value.

Availability

This function is available for applications running on 3270 mainframes only.

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_check_text

Context Sensitive • Terminal Emulator

captures and compares the text in a terminal emulator window.

TE_check_text (*file_name* [, *start_column*, *start_row*, *end_column*, *end_row*]);

<i>file_name</i>	A string expression given by WinRunner that identifies the captured window.
<i>start_column/row</i>	The column/row at which the captured text begins.
<i>end_column/row</i>	The column/row at which the captured text ends.

Return Values

This function returns 0 if the function succeeds, -1, if it fails, and 1 if a mismatch is found; otherwise, it returns a standard value. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_create_filter**Context Sensitive • Terminal Emulator**

creates a filter in the test database.

TE_create_filter (*filter_name*, *start_column*, *start_row*,
end_column, *end_row*, EXCLUDE|INCLUDE, *screen_name*);

<i>filter_name</i>	The filter name.
<i>start_column/row</i>	The column/row at which the filter starts.
<i>end_column/row</i>	The column/row at which the filter ends.
EXCLUDE/INCLUDE	The type of filter.
<i>screen_name</i>	The name of the screen to which you want to create the filter or ALL_SCREEN to create the filter for all screens in the application.

Return Values

This function returns 0 if the function succeeds; -1 in the case of an illegal number of parameters; 2 if the filter already exists; and 5 in case of an IO error. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_define_sync_keys

Context Sensitive • Terminal Emulator

sets keys that enable automatic synchronization in `type`, `win_type` and `obj_type` commands.

TE_define_sync_keys (*keys*, *string*, *mode* [, *x1*, *y1*, *x2*, *y2*]);

<i>keys</i>	The keys that will enable automatic synchronization. Use a comma as the delimiter between keys.
<i>string</i>	The string that WinRunner waits for to appear or disappear on the screen.
<i>mode</i>	The waiting mode: SYNC_WHILE: WinRunner waits until the string disappears. SYNC_UNTIL: WinRunner waits until the string appears. SYNC_DEFAULT: WinRunner waits the default synchronization time used by the <code>TE_wait_sync</code> function.
<i>x1</i> , <i>y1</i> , <i>x2</i> , <i>y2</i>	Optional parameters that define a rectangle on the screen in which to search for the string. If these parameters are missing, the entire screen is used.

Return Values

This function always returns 0.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_delete_filter

Context Sensitive • Terminal Emulator

deletes a specified filter from the test database.

TE_delete_filter (*filter_name*);

filter_name The filter to be deleted.

Return Values

This function returns 0 if the function succeeds; -1 in the case of an illegal number of parameters; 1 if the filter cannot be found in the database; and 5 in case of an IO error. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_edit_field

Context Sensitive • Terminal Emulator

inserts text into an unprotected field.

TE_edit_field (*field_logical_name*, *string* [, *x_shift*]);

field_logical_name The logical name of the field into which the string is inserted.

string The text to be inserted in the field.

x_shift Indicates the offset of the insertion position from the first character in the field, in characters. If no offset is specified, the default is 0.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_edit_hidden_field**Context Sensitive • Terminal Emulator**

inserts text into a hidden field.

TE_edit_hidden_field (*field_logical_name*, *coded_string*);

<i>field_logical_name</i>	The logical name of the field.
<i>coded_string</i>	A pointer to a coded string that WinRunner decodes and inserts into the field.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_edit_screen**Context Sensitive • Terminal Emulator**

types a string in the specified location in a screen.

TE_edit_screen (*x*, *y*, *string*);

<i>x,y</i>	The screen coordinates at which the string is inserted.
<i>string</i>	The text to be written on the screen.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_find_text**Context Sensitive • Terminal Emulator**

returns the location of a specified string

TE_find_text (*string*, *out_x_location*, *out_y_location* [, *x₁*, *y₁*, *x₂*, *y₂*]);

<i>string</i>	The text that you want to locate.
<i>out_x_location</i>	The output variable that stores the x coordinate of the test string.
<i>out_y_location</i>	The output variable that stores the x coordinate of the test string.
<i>x₁</i> , <i>y₁</i> , <i>x₂</i> , <i>y₂</i>	Describe a rectangle that define the limits of the search area.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_force_send_key

Context Sensitive • Terminal Emulator

defines a key causing a screen to change.

TE_force_send_key (*in_screen*, *in_field* [, *in_key*]);

<i>in_screen</i>	The name of the screen containing the field.
<i>in_field</i>	The name of the field.
<i>in_key</i>	The name of the key causing the screen to change (optional). The key name can be a mnemonic (such as @E for Enter) or one of the WinRunner macros. See the TE_send_key function for details.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_get_active_filter

Context Sensitive • Terminal Emulator

returns the coordinates of a specified active filter.

TE_get_active_filter (*filter_num* [, *out_start_column*, *out_start_row*, *out_end_column*,
out_end_row] , *screen_name*);

<i>filter_num</i>	The filter number representing the order in which filters were activated for the test, beginning with 0.
<i>out_start_column</i>	The output variable that stores the starting column of the filter.
<i>out_start_row</i>	The output variable that stores the starting row.
<i>out_end_column</i>	The output variable that stores the end column.
<i>out_end_row</i>	The output variable that stores the end row.

screen_name The output variable that stores the name of the screen in which the active filter is located. If the filter appears on all screens in the application, the function returns ALL_SCREEN.

Return Values

This function returns 0 if the filter exists, -1 if there is an illegal number of parameters and 1 if the filter cannot be found in the database. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_get_auto_reset_filters

Context Sensitive • Terminal Emulator

indicates whether or not filters are automatically deactivated at the end of a test run.

TE_get_auto_reset_filters ();

Return Values

This function returns ON to indicate that all filters are automatically deactivated at the end of a test run; OFF indicates that filters are not automatically deactivated. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_get_auto_verify

Context Sensitive • Terminal Emulator

indicates whether automatic text verification is on or off.

TE_get_auto_verify ();

Return Values

This function returns ON if automatic text verification is active; OFF indicates that automatic text verification is not active. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_get_cursor_position

Context Sensitive • Terminal Emulator

returns the position of the cursor.

TE_get_cursor_position (*x*, *y*);

x,y The current screen coordinates of the cursor.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_get_field_content

Context Sensitive • Terminal Emulator

returns the contents of a field to a variable.

TE_get_field_content (*field_name*, *content*);

<i>field_name</i>	The logical name of the field.
<i>content</i>	The output variable that stores the contents of the field as a string.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_get_filter

Context Sensitive • Terminal Emulator

returns the properties of a specified filter.

TE_get_filter (*filter_name* [, *out_start_column*, *out_start_row*, *out_end_column*,
out_end_row, *out_type*, *out_active*, *screen_name*]);

<i>filter_name</i>	The name of the filter.
<i>out_start_column</i>	The output variable that stores the starting column of the filter.
<i>out_start_row</i>	The output variable that stores the starting row.
<i>out_end_column</i>	The output variable that stores the end column.
<i>out_end_row</i>	The output variable that stores the end row.
<i>out_type</i>	The output variable that stores the filter type (INCLUDE EXCLUDE).

<i>out_active</i>	The output variable that stores the filter state.
<i>screen_name</i>	The variable that stores the screen name.

Return Values

This function returns 0 if the function succeeds; -1 if illegal parameters are used; 1 if a filter is not found; 2 if the parameter value is incorrect. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_get_merge_rule

Context Sensitive • Terminal Emulator

gets the rule for merging fields in a terminal emulator application.

TE_get_merge_rule (*from_field*, *to_field*, *rule*);

<i>from_field</i>	The logical name of the first field to be merged.
<i>to_field</i>	The logical name of the last field to be merged.
<i>rule</i>	The merging rule.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_get_refresh_time

Context Sensitive • Terminal Emulator

returns the time WinRunner waits for the screen to refresh.

```
TE_get_refresh_time ( );
```

Return Values

The return value of this function is an integer representing the refresh time. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_get_screen_name_location

Context Sensitive • Terminal Emulator

returns the screen name location.

```
TE_get_screen_name_location ( index, x, y, length );
```

<i>index</i>	A number between 0 - 10. 0 indicates that the screen name location was set by the function TE_set_screen_name_location. 1 – 10 indicates that the screen name was added with the function TE_add_screen_name_location.
<i>x,y</i>	The screen coordinates where WinRunner locates the logical name of the screen name.
<i>length</i>	The number of characters to the right of the y position that WinRunner locates the screen name string. The default length is 30 (maximum).

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_get_sync_time

Context Sensitive • Terminal Emulator

returns the system synchronization time.

TE_get_sync_time ();

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_get_text

Context Sensitive • Terminal Emulator

reads text from screen and stores it in a string.

TE_get_text (x_1, y_1, x_2, y_2);

x_1, y_1, x_2, y_2

Describes a rectangle that encloses the text to be read. The pairs of coordinates can designate any two diagonally opposite corners of the rectangle.

Return Values

This function returns the text read from the screen. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_get_timeout

Context Sensitive • Terminal Emulator

returns the current synchronization time.

TE_get_timeout ();

Return Values

The return value is the current value of the timeout. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_merge_fields

Context Sensitive • Terminal Emulator

sets the rule for merging fields in a terminal emulator application.

TE_merge_fields (*rule*);

rule The merging rule.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_reset_all_filters

Context Sensitive • Terminal Emulator

deactivates all filters in a test.

TE_reset_all_filters ();

Return Values

The return value of this function is always 0.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_reset_all_force_send_key

Context Sensitive • Terminal Emulator

deactivates the execution of **TE_force_send_key** functions.

TE_reset_all_force_send_key ();

Return Values

This function always returns 0.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_reset_all_merged_fields

Context Sensitive • Terminal Emulator

deactivates the merging of fields in a Terminal Emulator application.

TE_reset_all_merged_fields ();

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_reset_filter**Context Sensitive • Terminal Emulator**

deactivates a specified filter.

TE_reset_filter (*filter_name*);

filter_name Indicates the name of the filter to be deactivated.

Return Values

This function returns 0 if the function succeeds; -1 if illegal parameters are used; 1 if a filter is not found; 2 if the parameter value is incorrect. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_reset_screen_name_location**Context Sensitive • Terminal Emulator**

Resets the screen name location to 0.

TE_reset_screen_name_location ();

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_send_key

Context Sensitive • Terminal Emulator

sends to the mainframe the specified F-key function.

TE_send_key (*key*);

key

The F-key that is sent. The keys supported for this function are described in *the TSL Online Reference*.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_set_auto_reset_filters

Context Sensitive • Terminal Emulator

deactivates the automatic reset of filters when a test run is completed.

TE_set_auto_reset_filters (ON|OFF);

ON|OFF

ON indicates that upon completion of a test run, all filters are deactivated. OFF indicates that filters are not automatically deactivated. The default value is ON.

Return Values

This function returns 0 if it succeeds and -1 if it fails. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_set_auto_transaction

Context Sensitive • Terminal Emulator

defines a recorded TE_wait_sync statement as a transaction.

TE_set_auto_transaction (ON|OFF);

ON|OFF

ON activates set automatic transaction. OFF (the default) disables set automatic transaction is disabled.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_set_auto_verify

Context Sensitive • Terminal Emulator

activates/deactivates automatic text verification.

TE_set_auto_verify (ON|OFF [, x_1, y_1, x_2, y_2 [, FIRST|LAST]]);

ON|OFF

Activates or deactivates automatic text verification during recording.

x_1, y_1, x_2, y_2

Describes a rectangle that encloses the text to be verified. The pairs of coordinates can designate any two diagonally opposite corners of the rectangle.

FIRST|LAST

An optional parameter indicating the partial check coordinates to use: FIRST indicates the first incidence of partial text capture in the script, LAST indicates the partial text immediately before the current statement.

Return Values

The return value of this function is always 0.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_set_BMS_name_tag

Context Sensitive • Terminal Emulator

allows you to change a name tag that appears in your BMS file.

TE_set_BMS_name_tag (*name*);

name The name being set.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is available for applications running on 3270 mainframes only.

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_set_cursor_position

Context Sensitive • Terminal Emulator

defines the position of the cursor at the specified location on the screen of your mainframe application.

TE_set_cursor_position (*x, y*);

x,y The current screen coordinates of the cursor.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_set_field**Context Sensitive • Terminal Emulator**

specifies the field that will receive subsequent input.

TE_set_field (*field_logical_name* [, *x_offset*]);

<i>field_logical_name</i>	The name of the field.
<i>x_offset</i>	Indicates the offset of the insertion position from the first character in the field, in characters. If no offset is specified, the default is 0. The property byte is -1.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_set_filter**Context Sensitive • Terminal Emulator**

creates and activates a filter.

TE_set_filter (*filter_name* [, *start_column*, *start_row*, *end_column*, *end_row*, EXCLUDE|INCLUDE, *screen_name*]);

<i>filter_name</i>	The name of the filter.
<i>start_column/row</i>	The column/row at which the filter starts.
<i>end_column/row</i>	The column/row at which the filter ends.
EXCLUDE/INCLUDE	The type of filter.

screen_name

The name of the screen in which you want to set the filter or ALL_SCREEN to set the filter in all screens in the application.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_set_filter_mode

Context Sensitive • Terminal Emulator

specifies whether to assign filters to all screens or to the current screen.

TE_set_filter_mode (*mode*);

mode

The mode:

ALL_SCREEN: assigns filters to all screens.

CURRENT_SCREEN: assigns filters to the current screen (default).

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_set_record_method

Context Sensitive • Terminal Emulator

specifies the recording method for operations on terminal emulator objects.

TE_set_record_method (*method*);

method

This can be one of two constants: FIELD_METHOD (or 2), or POSITION_METHOD (or 1). FIELD_METHOD, the default, is full Context Sensitive recording. When POSITION_METHOD (partial Context Sensitive) is specified, keyboard and mouse input only is recorded for operations on objects in mainframe applications.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

For applications running on VT100, only POSITION_METHOD is available.

TE_set_refresh_time

Context Sensitive • Terminal Emulator

sets the interval that WinRunner waits for the screen to refresh.

TE_set_refresh_time (*time*);

time

The interval (in seconds) WinRunner waits for the screen to refresh. The default time is one second.

Return Values

This function always returns 0.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_set_screen_name_location

Context Sensitive • Terminal Emulator

resets the screen name location to 0 and then instructs WinRunner where to look for the logical name of a screen.

TE_set_screen_name_location (*x, y, length*);

<i>x,y</i>	The screen coordinates where WinRunner begins looking for the logical name of all screens in the test. The default location is 1,1.
<i>length</i>	The number of characters to the right of the y position that WinRunner will search for a string. The default length is 30 (maximum).

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_set_sync_time

Context Sensitive • Terminal Emulator

defines the system synchronization time.

TE_set_sync_time (*time*);

<i>time</i>	The minimum number of seconds that WinRunner will wait for the host to respond in order to determine that synchronization has been achieved before continuing test execution.
-------------	---

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_set_timeout**Context Sensitive • Terminal Emulator**

sets the maximum time WinRunner waits for a response from the server.

TE_set_timeout (*timeout*);

timeout

The interval (in seconds) WinRunner waits for a response from the server before continuing test execution. The default timeout is 60 seconds.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_set_trailing**Context Sensitive • Terminal Emulator**

Determines whether WinRunner types spaces and tabs in fields during test execution.

TE_set_trailing (*mode*, *field_length*);

mode

One of two modes can be specified. ON or OFF.

field_length

The field length affected by the trailing mode. For example, if the field length is 5, the trailing mode affects all fields containing up to five spaces. Fields above the designated field length are not affected.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_user_attr_comment

Context Sensitive • Terminal Emulator

enables a user to add a user-defined comment property to the physical description of fields in the GUI map.

TE_user_attr_comment (*name*);

name The name of the user-defined comment property.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_user_reset_all_attr_comments

Context Sensitive • Terminal Emulator

Resets all user-defined comment properties.

TE_user_reset_all_attr_comments ();

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_wait_field**Context Sensitive • Terminal Emulator**

waits for a specified string in a specified field to appear on screen.

TE_wait_field (*field_logical_name*, *content*, *timeout*);

<i>field_logical_name</i>	The logical name of the field.
<i>content</i>	The text string WinRunner waits for.
<i>timeout</i>	The number of seconds that WinRunner waits for the string to appear before continuing test execution.

Return Values

This function returns 0 if the string is found; 1 if the string is not found; -1 if the function fails. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_wait_string**Context Sensitive • Terminal Emulator**

waits for a string to appear on screen.

TE_wait_string (*string* [, *start_column*, *start_row*, *end_column*, *end_row* [, *timeout*]]);

<i>string</i>	The text WinRunner waits for.
<i>start_column/row</i>	The starting column/row at which the text will be searched for.
<i>end_column/row</i>	The end column/row at which the text will be searched for.

timeout

The number of seconds that the interpreter waits for the string to appear before continuing test execution.

Return Values

This function returns 0 if the string is found; 1 if the string is not found; -1 if the function fails. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

TE_wait_sync

Context Sensitive • Terminal Emulator

instructs WinRunner to wait for the terminal emulator screen to be redrawn.

TE_wait_sync ();

The **TE_wait_sync** function instructs WinRunner to wait, during execution, for the screen of the terminal emulator to be redrawn before continuing test execution.

Return Values

This function returns the actual time that the terminal emulator screen took to redraw. For more information, see “General Return Values,” on page 118, and “Return Values for Terminal Emulator Functions,” on page 124.

Availability

This function is supported for WinRunner EURO and WinRunner with Year 2000 Add-in support only.

It is superfluous for the VT100.

textit

Standard • Call Statement

stops execution of the current test.

textit ([*expression*]);

expression

The value that is returned to the call statement that invokes the called test. If no value is specified, then the return value of the call statement is 0.

Return Values

The **textit** statement is a keyword, not a function. It does not have a return value.

Availability

This statement is always available.

Note: The **textit** statement is not a function. Therefore, it does not appear in the Function Generator.

time_str

Standard • Time-Related

converts the integer returned by the **get_time** function to a string.

time_str ([*expression*]);

expression

The value of this expression must be expressed in the format generated by **get_time** (the time expressed in the number of seconds that have elapsed since 00:00 GMT, January 1, 1970). If expression is not included (null), **time_str** converts the current value returned by **get_time**.

Return Values

This function returns a string in the format "Day Month Date Hour:Min:Sec Year."

Availability

This function is always available.

tl_step

Standard • Miscellaneous

divides a test script into sections and inserts a status message in the test results for the previous section.

tl_step (*step_name*, *status*, *description*);

<i>step_name</i>	the name of the test step.
<i>status</i>	sets whether the step passed or failed. Set to 0 for pass, or any other integer for failure.
<i>description</i>	a short explanation of the step.

The **tl_step** function divides test scripts into sections and determines whether each section passes or fails. When the test run is completed, you view the test results in the Test Results window. The report displays a result (pass/fail) for each step you defined.

When WinRunner is connected to a TestDirector project, the message is inserted in the TestDirector “step” table as well.

Return Values

This function returns 0 if the step passes. If the return value is not zero, the step fails.

Availability

This function is always available.

tl_step_once

Standard • Miscellaneous

divides a test script into sections and inserts a status message in the test results for the previous section.

tl_step_once (*step_name*, *status*, *description*);

<i>step_name</i>	the name of the test step.
<i>status</i>	sets whether the step passed or failed. Set to 0 for pass, or any other integer for failure.
<i>description</i>	a short explanation of the step.

The **tl_step_once** function divides test scripts into sections and determines whether each section passes or fails. When the test run is completed, you view the test results in the Test Results window. The report displays a result (pass/fail) for each step you defined.

When WinRunner is connected to a TestDirector project, the message is inserted in the TestDirector “step” table as well. Note that the message is inserted in the TestDirector “step” table once per *step_name*.

Return Values

This function returns 0 if the step passes. If the return value is not zero, the step fails.

Availability

This function is always available.

tolower

Standard • String

converts all uppercase characters in a string to lowercase.

tolower (*string*);

string A string expression.

Return Values

This function returns a lower case string.

Availability

This function is always available.

toolbar_button_press

Context Sensitive • Toolbar Object

clicks on a toolbar button.

toolbar_button_press (*toolbar*, *button*, *mouse_button*);

<i>toolbar</i>	The logical name of the toolbar.
<i>button</i>	The button to press. This can be either the logical name or the numeric index of the button. The logical name reflects the button's attached text (tooltip). The index is specified as a string preceded by the character #. The first button in a toolbar is #0.
<i>mouse_button</i>	The name of the mouse button pressed when pressing the button in the toolbar. The names (Left, Right, Middle) are defined by the XR_INP_MKEYS system parameter in the system configuration file. This parameter is optional.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

toolbar_get_button

Context Sensitive • Toolbar Object

returns the name of toolbar button.

toolbar_get_button (*toolbar*, *button_num*, *out_text*);

<i>toolbar</i>	The logical name of the toolbar.
<i>button_num</i>	The numeric index of the button in the toolbar.
<i>out_text</i>	The output variable that stores the text.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

toolbar_get_buttons_count**Context Sensitive • Toolbar Object**

returns the number of buttons in a toolbar.

```
toolbar_get_buttons_count ( toolbar, out_num );
```

<i>toolbar</i>	The logical name of the toolbar.
<i>out_num</i>	The output variable that stores the number of buttons on the toolbar.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

toolbar_get_button_info**Context Sensitive • Toolbar Object**

returns the value of a toolbar button property.

```
toolbar_get_button_info ( toolbar, button, property, out_value );
```

<i>toolbar</i>	The logical name of the toolbar.
<i>button</i>	The logical name or the numeric index of the button. The logical name reflects the button’s attached text (tooltip). The index is specified as a string preceded by the character #. The first button in a toolbar is #0.
<i>property</i>	Any of the properties listed in the "Configuring the GUI Map" in the <i>WinRunner User’s Guide</i> .
<i>out_value</i>	The output variable that stores the value of the specified property.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

toolbar_get_button_num

Context Sensitive • Toolbar Object

returns the position of a toolbar button.

toolbar_get_button_num (*toolbar*, *button*, *out_num*);

<i>toolbar</i>	The logical name of the toolbar.
<i>button</i>	The logical name of the button. The logical name reflects the button’s attached text. The index is specified as a string preceded by the character #. The first button in a toolbar is #0.
<i>out_num</i>	The output variable that stores the numeric position of the button on the toolbar. The first button is automatically number 0.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

toolbar_get_buttons_count

Context Sensitive • Toolbar Object

returns the number of buttons in a toolbar.

```
toolbar_get_buttons_count ( toolbar, out_num );
```

<i>toolbar</i>	The logical name of the toolbar.
<i>out_num</i>	The output variable that stores the number of buttons on the toolbar.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

toolbar_select_item

Context Sensitive • Toolbar Object

selects an item from a menu-like toolbar, as in Microsoft Internet Explorer 4.0 or the Start menu in Windows 98.

```
toolbar_select_item ( toolbar, toolbar_item_chain [ , mouse_button ] );
```

<i>toolbar</i>	The logical name of the toolbar containing the first item in toolbar_item_chain .
<i>toolbar_item_chain</i>	The chain of toolbar items separated by the TreeView separator (by default, a semi-colon). You can configure the separator in the General Options dialog box. If the item string is not available, then the item index will be recorded instead.
<i>mouse_button</i>	The name of the mouse button pressed when selecting the last item in toolbar_item_path . The names (Left, Right, Middle) are defined by the XR_INP_MKEYS system parameter in the system configuration file. This parameter is optional.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

toupper

Standard • String

converts all lowercase characters in a string to uppercase.

toupper (*string*);

string A string expression.

Return Values

This function returns an uppercase string.

Availability

This function is always available.

treturn

Standard • Call Statements

stops a called test and returns control to the calling test.

treturn [(*expression*)];

expression The value that is returned to the call statement invoking the called test. If no value is specified, then the return value of the call statement is 0.

Return Values

The **treturn** statement is a keyword, not a function, and does not have a return value.

Availability

This statement is always available.

Note: The **return** statement is not a function. Therefore, it does not appear in the Function Generator.

type

Analog • Input Device

specifies keyboard input.

type (*keyboard_input* [, *technical_id*]);

<i>keyboard_input</i>	A string expression that represents keystrokes.
<i>technical_id</i>	Points to timing and synchronization data. This parameter is only present when the type statement is generated during recording.

The **type** function depicts the keyboard input sent to the application under test. Keyboard input is evaluated to a string using the following conventions. The *TSL Online Reference* contains the conventions for evaluating keyboard input to a string.

Return Values

The return value of the function is always 0.

Availability

This function is always available.

unload

Standard • Compiled Module

removes a compiled module or selected functions from memory.

unload ([*module* | *test* [, *function_name*]]);

module | *test* A string expression indicating the name of an existing compiled module or test.

function_name A string expression indicating the name of an existing compiled function.

The unload function can remove an entire module from memory, or a selected function. When only a module or test name is specified, all functions within that module/test are removed.

If no arguments are specified, unload removes all compiled modules from memory.

A system module is generally a closed module that is “invisible” to the tester. It is not displayed when it is loaded, cannot be stepped into, and is not stopped by a pause command. A system module is not unloaded when you execute an unload statement with no parameters (global unload).

A user module is the opposite of a system module in these respects. Generally, a user module is one that is still being developed. In such a module you might want to make changes and compile them incrementally.

Note: If you make changes to a function in a loaded compiled module, you must unload and reload the compiled module in order for the changes to take effect.

Return Values

This function returns 0 for success, and 1 for failure.

Availability

This function is always available.

unload_16_dll

Standard • Miscellaneous

unloads a 16-bit DLL from memory.

unload_16_dll (*pathname*);

pathname The full pathname of the Dynamic Link Library (DLL) to be unloaded.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

unload_dll

Standard • Miscellaneous

unloads a DLL from memory.

unload_dll (*pathname*);

pathname The full pathname of the Dynamic Link Library (DLL) to be unloaded.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

unset_class_map

Context Sensitive • GUI Map Configuration

unbinds a custom class from a standard class.

```
unset_class_map ( custom_class );
```

custom_class The name of the custom class to unbind.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner and GUI Vusers running on PC platforms only.

user_data_point

Standard • Load Testing

records a user-defined data sample.

```
int user_data_point ( sample_name, value );
```

sample_name A string indicating the name of the sample type.

value The value to record.

Return Values

This function returns 0 if it succeeds, and -1 if it fails to write the sampled data.

Availability

This function is available for LoadRunner GUI Vusers only.

vb_get_label_names

Context Sensitive • ActiveX/Visual Basic

retrieves the names of all label controls in the given form window. The names are stored as subscripts of an array.

vb_get_label_names (*window*, *name_array*, *count*);

<i>window</i>	The logical name of the Visual Basic form.
<i>name_array</i>	The out parameter containing the name of the storage array.
<i>count</i>	The out parameter containing the number of elements in the array.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available only for the Visual Basic add-in.

wait

Standard • Time-Related

pauses test execution.

wait (*seconds* [, *milliseconds*]);

<i>seconds</i>	The length of the pause, in seconds. The valid range of this parameter is from 0 to 32,767 seconds.
<i>milliseconds</i>	The number of milliseconds that are added to the <i>seconds</i> .

Return Values

The return value of the function is always 0.

Availability

This function is always available.

wait_window

Analog • Synchronization Functions

waits for a window bitmap to appear.

Note: This function is provided for backward compatibility only. The Context Sensitive versions of this function are **win_check_bitmap** and **obj_check_bitmap**. You should use these functions instead.

wait_window (*time*, *image*, *window*, *width*, *height*, *x*, *y* [, *relx₁*, *rely₁*, *relx₂*, *rely₂*]);

<i>time</i>	The <i>time</i> is added to the <i>timeout_msec</i> testing option to give the maximum interval between the previous input even and the screen capture.
<i>image</i>	A string expression identifying the captured bitmap.
<i>window</i>	A string expression indicating the name in the window banner.
<i>width</i> , <i>height</i>	The size of the window, in pixels.
<i>x</i> , <i>y</i>	The position of the upper left corner of the window.
<i>relx₁</i> , <i>rely₁</i>	For an area bitmap: the coordinates of the upper left corner of the rectangle, relative to the upper left corner of the window, expressed in pixels (the <i>x</i> and <i>y</i> parameters).
<i>relx₂</i> , <i>rely₂</i>	For an area bitmap: the coordinates of the lower right corner of the rectangle, relative to the lower right corner of the window (the <i>x</i> and <i>y</i> parameters).

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

web_browser_invoke

Context Sensitive • Web

invokes the browser and opens a specified site.

web_browser_invoke (*browser*, *site*);

<i>browser</i>	The name of browser (Microsoft Internet Explorer or Netscape).
<i>site</i>	The address of the site.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_cursor_to_image

Context Sensitive • Web

moves the cursor to an image on a page.

web_cursor_to_image (*image*, *x*, *y*);

<i>image</i>	The logical name of the image.
<i>x,y</i>	The x- and y-coordinates of the mouse pointer when moved to an image, relative to the upper left corner of the image.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_cursor_to_label

Context Sensitive • Web

moves the cursor to a label on a page.

web_cursor_to_label (*label*, *x*, *y*);

<i>label</i>	The name of the label.
<i>x,y</i>	The x- and y- coordinates of the mouse pointer when moved to a label, relative to the upper left corner of the label.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_cursor_to_link

Context Sensitive • Web

moves the cursor to a link on a page.

web_cursor_to_link (*link*, *x*, *y*);

<i>link</i>	The name of the link.
<i>x,y</i>	The x- and y- coordinates of the mouse pointer when moved to a link, relative to the upper left corner of the link.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_cursor_to_obj

Context Sensitive • Web

moves the cursor to an object on a page.

web_cursor_to_obj (*object*, *x*, *y*);

<i>object</i>	The name of the object.
<i>x,y</i>	The x- and y-coordinates of the mouse pointer when moved to an object, relative to the upper left corner of the object.

The **web_cursor_to_obj** function moves the cursor to an object on a frame. The x- and y-coordinates of the mouse pointer when moved to an object are relative to the upper left corner of the object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

This function is available only when using Microsoft Internet Explorer.

web_event

Context Sensitive • Web

runs an event on the specified object.

web_event (*object*, *event_name* [, *x*, *y*]);

<i>object</i>	The logical name of the recorded object.
<i>event_name</i>	The name of an event handler. Use one of the following events: blur : An event occurs when an object loses focus, or when a window or a frame loses focus. change : An event occurs when a value of an object has been modified. click : An event occurs when an object is clicked.

focus: An event occurs when an object receives focus by clicking the mouse or by tabbing with the keyboard.

mousedown: An event occurs when the mouse button is clicked down.

mouseout: An event occurs when the mouse pointer leaves an object from inside that object.

mouseover: An event occurs when the mouse pointer moves over an object from outside that object.

mouseup: An event occurs when the mouse button is released.

x,y

The x- and y-coordinates of the mouse pointer when moved to an object, relative to the upper left corner of the object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_file_browse

Context Sensitive • Web

clicks a browse button.

web_file_browse (*object*);

object

A file-type object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

This function is available only when using Microsoft Internet Explorer.

web_file_set

Context Sensitive • Web

sets the text value in a file-type object.

web_file_set (*object*, *value*);

object A file-type object.

value A text string.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

This function is available only when using Microsoft Internet Explorer.

web_find_text

Context Sensitive • Web

returns the location of text within a frame.

web_find_text (*frame*, *text_to_find*, *result_array* [, *text_before*, *text_after*, *index*, *show*]);

frame The name of the frame.

text_to_find The specified text string to locate.

result_array The name of the output variable that stores the location of the string as a four-element array.

text_before Defines the start of the search area for a particular text string.

text_after Defines the end of the search area for a particular text string.

index The occurrence number to locate. (The default parameter number is numbered 1.)

show Indicates whether to highlight the location of the string. If TRUE (default parameter) is specified, the text location is highlighted. If FALSE is specified, the text location is not highlighted.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_frame_get_text

Context Sensitive • Web

retrieves the text content of a frame.

web_frame_get_text (*frame*, *out_text* [, *text_before*, *text_after*, *index*]);

<i>frame</i>	The name of the frame.
<i>out_text</i>	The captured text content.
<i>text_before</i>	Defines the start of the search area for a particular text string.
<i>text_after</i>	Defines the end of the search area for a particular text string.
<i>index</i>	The occurrence number to locate. (The default parameter number is numbered 1).

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_frame_get_text_count

Context Sensitive • Web

returns the number of occurrences of a regular expression in a frame.

web_frame_get_text_count (*frame*, *regex_text_to_find*, *count*);

<i>frame</i>	The name of the frame.
<i>regex_text_to_find</i>	The specified regular expression to locate.
<i>count</i>	The output variable that stores the count number.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_frame_text_exists

Context Sensitive • Web

returns a text value if it is found in a frame.

web_frame_text_exists (*frame*, *text_to_find* [, *text_before*, *text_after*]);

<i>frame</i>	The name of the frame to search.
<i>text_to_find</i>	The string that is searched for.
<i>text_before</i>	Defines the start of the search area for a particular text string.
<i>text_after</i>	Defines the end of the search area for a particular text string.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_get_run_event_mode

Context Sensitive • Web

returns the current run mode.

```
web_get_run_event_mode ( out_mode );
```

<i>out_mode</i>	The run mode in use. If the mode is FALSE (the default) the test runs by mouse operations. If TRUE is specified, the test runs by events.
-----------------	---

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_get_timeout

Context Sensitive • Web

returns the maximum time that WinRunner waits for response from the web.

```
web_get_timeout ( out_timeout );
```

<i>out_timeout</i>	The maximum response interval in seconds.
--------------------	---

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_image_click

Context Sensitive • Web

clicks a hypergraphic link or an image.

web_image_click (*image*, *x*, *y*);

<i>image</i>	The logical name of the image.
<i>x,y</i>	The x- and y-coordinates of the mouse pointer when clicked on a hypergraphic link or an image. The coordinates are relative to the upper left corner of the image.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_label_click

Context Sensitive • Web

clicks the specified label.

web_label_click (*label*);

<i>label</i>	The name of the label.
--------------	------------------------

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_link_click

Context Sensitive • Web

clicks a hypertext link.

web_link_click (*link*);

link The name of link.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_link_valid

Context Sensitive • Web

checks whether a URL name of a link is valid (not broken).

web_link_valid (*name*, *valid*);

name The logical name of a link.

valid The status of the link may be valid (TRUE) or invalid (FALSE)

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_obj_click

Context Sensitive • Web

clicks an object in a frame.

web_obj_click (*object*, *x*, *y*);

<i>object</i>	The name of an object.
<i>x,y</i>	The x- and y-coordinates of the mouse pointer when clicked on an object. The coordinates are relative to the upper left corner of the object.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

This function is available only when using Microsoft Internet Explorer.

web_obj_get_child_item

Context Sensitive • Web

returns the description of the children in an object.

web_obj_get_child_item (*object*, *table_row*, *table_column*, *object_type*, *index*,
out_object);

<i>object</i>	The name of object.
<i>table_row</i>	The row number in the table.
<i>table_column</i>	The column number in the table.
<i>object_type</i>	Specifies the object type.
<i>index</i>	Unique number assigned to the object.
<i>out_object</i>	The output variable that stores the description.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_obj_get_child_item_count

Context Sensitive • Web

function returns the count of the children in an object.

```
web_obj_get_child_item_count ( object, table_row, table_column, object_type,  
                               object_count );
```

<i>object</i>	The name of object.
<i>table_row</i>	The row number in the table.
<i>table_column</i>	The column number in the table.
<i>object_type</i>	Specifies the object type.
<i>object_count</i>	The output variable that stores the object count number.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_obj_get_info

Context Sensitive • Web

returns the value of an object property.

web_obj_get_info (*object*, *property_name*, *property_value*);

<i>object</i>	The name of the object.
<i>property_name</i>	The name of the property (PARENT, SCR, TEXT, TYPE, URL).
<i>property_value</i>	The output variable that stores the value of the property.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_obj_get_text

Context Sensitive • Web

returns a text string from an object.

web_obj_get_text (*object*, *table_row*, *table_column*, *out_text* [, *text_before*, *text_after*, *index*]);

<i>object</i>	The name of the object.
<i>table_row</i>	If the object is a table, it specifies the location of the row within a table. The string is preceded by the # character.
<i>table_column</i>	If the object is a table, it specifies the location of the column within a table. The string is preceded by the # character.
<i>out_text</i>	The output variable that stores the text string.
<i>text_before</i>	Defines the start of the search area for a particular text string.

<i>text_after</i>	Defines the end of the search area for a particular text string.
<i>index</i>	The occurrence number to locate. (The default parameter number is numbered 1).

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_obj_get_text_count

Context Sensitive • Web

returns the number of occurrences of a regular expression in an object.

web_obj_get_text_count (*object*, *table_row*, *table_column*, *regex_text_to_find*, *count*);

<i>object</i>	The name of the object.
<i>table_row</i>	If the object is a table, it specifies the location of the row within a table. The string is preceded by the character #.
<i>table_column</i>	If the object is a table, it specifies the location of the column within a table. The string is preceded by the character #.
<i>regex_text_to_find</i>	The specified regular expression to locate.
<i>count</i>	The output variable that stores the count number.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_obj_text_exists

Context Sensitive • Web

returns a text value if it is found in an object.

```
web_obj_text_exists ( object, table_row, table_column, text_to_find [ , text_before,
                    text_after ] );
```

<i>object</i>	The name of the object to search.
<i>table_row</i>	If the object is a table, it specifies the location of the row within a table. The string is preceded by the character #.
<i>table_column</i>	If the object is a table, it specifies the location of the column within a table. The string is preceded by the character #.
<i>text_to_find</i>	The string that is searched for.
<i>text_before</i>	Defines the start of the search area for a particular text string.
<i>text_after</i>	Defines the end of the search area for a particular text string.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_password_encrypt

Context Sensitive • Web

encrypts a password on a Web page.

```
web_password_encrypt ( password );
```

<i>password</i>	The password on the Web page.
-----------------	-------------------------------

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_refresh

Context Sensitive • Web

resets WinRunner's connection to the specified frame.

web_refresh (*frame*);

frame The logical name of the frame.

Tip: Call this function when the frame changes dynamically and WinRunner does not capture the change.

Note: This function is not recordable.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is available for WebTest only.

web_restore_event_default

Context Sensitive • Web

resets all events to their default settings.

web_restore_event_default ();

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is available for WebTest only.

web_set_event

Context Sensitive • Web

sets the event status.

```
web_set_event ( class, event_name, event_type, event_status );
```

<i>class</i>	The MSW class of the object.
<i>event_name</i>	The name of an event handler. Use one of the following: blur: An event occurs when an object loses focus, or when a window or a frame loses focus. change: An event occurs when a value of an object has been modified. click: An event occurs when an object is clicked. focus: An event occurs when an object receives focus by clicking the mouse or by tabbing with the keyboard. mousedown: An event occurs when the mouse button is clicked down. mouseout: An event occurs when the mouse pointer leaves an object from inside that object. mouseover: An event occurs when the mouse pointer moves over an object from outside that object. mouseup: An event occurs when the mouse button is released.
<i>event_type</i>	The name of an event type. Use one of the following: ANYCASE: Connects to the event in any case. BEHAVIOR: Connects to an event only when the behavior is associated with the object class. HANDLER: Connects to an event only when the handler exists in the HTML script. BEHAVIOR_OR_HANDLER: Connects to an event only when the handler exists in the HTML script, or when the behavior is associated with the object class.

event_status

The name of an event status. Use one of the following:

ENABLE: The event is recordable.

DISABLE: Disables the recordable event for an object class.

DELETE: Deletes and disables an event from the map of recordable events.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_set_run_event_mode

Context Sensitive • Web

sets the event run mode.

web_set_run_event_mode (*mode*);

mode

The event run mode can be set to TRUE or FALSE. If set to FALSE, the test runs by mouse operations. If set to TRUE, the test runs by events.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_set_timeout

Context Sensitive • Web

sets the maximum time WinRunner waits for a response from the Web.

web_set_timeout (*timeout*);

timeout The maximum interval in seconds.

The **web_set_timeout** function sets the maximum time WinRunner waits for a response from the web.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_set_tooltip_color

Context Sensitive • Web

sets the colors of the WebTest ToolTip.

web_set_tooltip_color (*fg_color*, *bg_color*);

fg_color A hexadecimal number denoting a color value of the foreground color. Default color is set to black.

bg_color A hexadecimal number denoting a color value of the background color. Default color is set to aqua.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_sync

Context Sensitive • Web

waits for the navigation of a frame to be completed.

web_sync (*timeout*);

time The maximum interval in seconds.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_tbl_get_cell_data

Context Sensitive • Web

retrieves the contents of the specified cell from a Web table, starting from the specified character.

web_tbl_get_cell_data (*table, row, column, starting_pos, out_text, out_actual_text_length*);

table The logical name of the table.

row By location: # <row_location> The location of the row within the table, specified by a string preceded by the character #, such as "#2".

The row can also be in the following format:

column By location: # <column_location> The location of the column within the table, specified by a string preceded by the character #, such as "#2".

starting_pos The index of the character in the cell from which WinRunner starts retrieving the text string.

<i>out_text</i>	The output variable that stores the string found in the specified cell.
<i>out_actual_text_length</i>	The actual length of the text string in the table. Note that this length cannot exceed 1023 characters.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

web_url_valid

Context Sensitive • Web

checks whether a URL is valid.

web_url_valid (*URL*, *valid*);

<i>URL</i>	Address of a link.
<i>valid</i>	The status of the link may be valid (TRUE) or invalid (FALSE).

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WebTest only.

win_activate

Context Sensitive • Window Object

activates a window.

win_activate (*window*);

window The logical name of the window.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available in WinRunner only.

win_check_bitmap

Context Sensitive • Window Object

compares a window bitmap to an expected bitmap.

win_check_bitmap (*window, bitmap, time* [, *x, y, width, height*]);

window The logical name of the window.

bitmap A string expression that identifies the captured bitmap.

time The interval marking the maximum delay between the previous input event and the capture of the current bitmap, in seconds. This interval is added to the *timeout_msec* testing option.

x, y For an area bitmap: the coordinates of the upper left corner, relative to the window in which the selected area is located.

width, height For an area bitmap: the size of the selected area, in pixels.

The analog version of **win_check_bitmap** is **check_window**.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_check_gui**Context Sensitive • Window Object**

compares current GUI data to expected GUI data for a window.

win_check_gui (*window*, *checklist*, *expected_results_file*, *time*);

<i>window</i>	The logical name of the window.
<i>checklist</i>	The name of the checklist specifying the checks to perform.
<i>expected_results_file</i>	The name of the file storing the expected GUI data.
<i>time</i>	The <i>time</i> is added to the <i>timeout_msec</i> testing option to give the maximum interval between the previous input even and the screen capture.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_check_info

Context Sensitive • Window Object

checks the requested window property.

win_check_info (*window*, *property*, *property_value*);

<i>window</i>	The logical name of the window.
<i>property</i>	The property to check.
<i>property_value</i>	The expected value of the property.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_click_help

Context Sensitive • Window Object

clicks the help button in a window title bar.

win_click_help (*window*);

<i>window</i>	The logical name of the window.
---------------	---------------------------------

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_click_on_text

Context Sensitive • Window Object

searches for text in a window.

```
win_click_on_text (window, string [ ,search_area [ , string_def [ , mouse_button ] ] ] );
```

<i>window</i>	The logical name of the window.
<i>string</i>	The text to locate. To specify a literal, case sensitive string, enclose the string in quotation marks. Alternatively, you can specify the name of a string variable. The value of the string variable can include a regular expression (the regular expression need not begin with an exclamation mark).
<i>search_area</i>	The region of the object to search, relative to the window. This area is defined as a pair of coordinates, with <i>x1,y1,x2,y2</i> specifying any two diagonally opposite corners of the rectangular search region. If this parameter is not defined, then the entire window specified is considered the search area.
<i>string_def</i>	Defines how the text search is performed. If no <i>string_def</i> is specified, (0 or FALSE, the default parameter), the interpreter searches for a complete word only. If 1, or TRUE, is specified, the search is not restricted to a single, complete word.
<i>mouse_button</i>	Specifies the mouse button that clicks on the text string. The value can be LEFT, MIDDLE, or RIGHT. If no button is specified, the default is the left button.

The analog version of this function is **click_on_text**.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_close

Context Sensitive • Window Object

closes a window.

win_close (*window*);

window The logical name of the window.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_drag

Context Sensitive • Window Object

drags an object from a source window.

win_drag (*source_window*, *x*, *y* [, *mouse_button*]);

source_window The logical name of the window.

x,y The coordinates of the mouse pointer when clicked on the source window, relative to the upper left corner of the client area of the source window expressed in pixels.

mouse_button A constant that specifies the mouse button to hold down while dragging. The value can be LEFT, MIDDLE, or RIGHT. If no button is specified, the default is the button that performs the select function.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_drop

Context Sensitive • Window Object

drops an object onto a target window.

win_drop (*target_window*, *x*, *y*);

<i>target_window</i>	The logical name of the window.
<i>x,y</i>	The coordinates of the mouse pointer when released over the target window, relative to the upper left corner of the client area of the target window, expressed in pixels.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_exists

Context Sensitive • Window Object

checks whether a window is displayed on the screen.

win_exists (*window* [, *time*]);

<i>window</i>	The logical name of the window.
<i>time</i>	The amount of time (in seconds) that is added to the default timeout setting (specified with the <i>timeout_msec</i> testing option), yielding a new maximum wait time before the subsequent statement is executed.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_find_text

Context Sensitive • Window Object

returns the location of a string within a window.

```
win_find_text ( window, string, result_array [ , search_area [ , string_def ] ] );
```

<i>window</i>	The logical name of the window to search.
<i>string</i>	The text to locate. To specify a literal, case sensitive string, enclose the string in quotation marks. Alternatively, you can specify the name of a string variable. The value of the string variable can include a regular expression. The regular expression should not include an exclamation mark (!), however, which is treated as a literal character.
<i>result_array</i>	The name of the output variable that stores the location of the string as a four-element array.
<i>search_area</i>	The region of the object to search, relative to the window. This area is defined as a pair of coordinates, with <i>x1,y1,x2,y2</i> specifying any two diagonally opposite corners of the rectangular search region. If this parameter is not defined, then the entire <i>window</i> is considered the search area.
<i>string_def</i>	Defines how the text search is performed. If no <i>string_def</i> is specified, (0 or FALSE, the default parameter), the interpreter searches for a complete word only. If 1, or TRUE, is specified, the search is not restricted to a single, complete word.

The Analog version of this function is **find_text**.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_get_desc

Context Sensitive • Window Object

returns the physical description of a window.

win_get_desc (*window*, *obligatory*, *optional*, *selector*, *out_desc*);

<i>window</i>	The logical name of the window.
<i>obligatory</i>	The list of obligatory properties (separated by spaces).
<i>optional</i>	The list of optional properties (separated by spaces).
<i>selector</i>	The type of selector used for this object class (location or index).
<i>out_desc</i>	The output variable that stores the description of the window.

Return Values

This function returns the value 0 if it succeeds and -1 if it fails. If obligatory, optional, and selector are null strings, **win_get_desc** returns the current learning configuration for the object

Availability

This function is always available.

win_get_info

Context Sensitive • Window Object

returns the value of a window property.

win_get_info (*window*, *property*, *out_value*);

<i>window</i>	The logical name of the window.
<i>property</i>	Any of the properties listed in the User's Guide.
<i>out_value</i>	The variable that stores the value of the specified property.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

win_get_text**Context Sensitive • Window Object**

reads text from the indicated area of a window.

win_get_text (*window*, *out_text* [, *x1*, *y1*, *x2*, *y2*]);

<i>window</i>	The window from which text is read.
<i>out_text</i>	The output variable that holds the captured text.
<i>x1,y1,x2,y2</i>	An optional parameter that defines the location from which to read text relative to the specified window in pixels. The coordinate pairs can designate any two diagonally opposite corners of a rectangle. The interpreter searches for the text in the area defined by the rectangle.

The Analog version of this function is **get_text**.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_highlight

Context Sensitive • Window Object

highlights the specified window.

win_highlight (*window* [, *flashes*]);

window The logical name of the window.

flashes The number of times the window flashes on screen.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_max

Context Sensitive • Window Object

maximizes a window to fill the entire screen.

win_max (*window*);

window The logical name of the window.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner and LoadRunner GUI Vusers running on PC platforms only.

win_min

Context Sensitive • Window Object

minimizes a window to an icon.

win_min (*window*);

window the logical name of the window.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is available for WinRunner and LoadRunner GUI Vusers running on PC platforms only.

win_mouse_click

Context Sensitive • Window Object

performs a mouse click within a window.

win_mouse_click (*window*, *x*, *y* [, *mouse_button* [, *modifier*]]);

window The logical name of the window.

x, *y* The position of the mouse click expressed as *x* and *y* (pixel) coordinates. Coordinates are relative to the upper left corner of the client area of the window, and not to the screen.

mouse_button A constant specifying the mouse button to click. The value can be LEFT, MIDDLE, or RIGHT. If no *mouse_button* is specified, the default is the button performing the select function.

modifier A constant specifying the modifier key used with the mouse button. The value can be CONTROL, SHIFT, or CONTROL_SHIFT.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_mouse_dbl_click**Context Sensitive • Window Object**

performs a double-click within a window.

win_mouse_dbl_click (*window*, *x*, *y* [, *mouse_button* [, *modifier*]]);

<i>window</i>	The logical name of the window.
<i>x</i> , <i>y</i>	The position of the double-click expressed as <i>x</i> and <i>y</i> (pixel) coordinates. Coordinates are relative to the upper left corner of the client area of the window, and not to the screen.
<i>mouse_button</i>	A constant specifying the mouse button to click. The value can be LEFT, MIDDLE, or RIGHT. If no <i>mouse_button</i> is specified, the default is the button performing the select function.
<i>modifier</i>	A constant specifying the modifier key used with the mouse button. The value can be CONTROL, SHIFT, or CONTROL_SHIFT.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_mouse_drag

Context Sensitive • Window Object

performs a mouse drag within a window.

```
win_mouse_drag ( window, start_x, start_y, end_x, end_y [ , mouse_button  
                [ ,modifier ] ] );
```

<i>window</i>	The logical name of the window.
<i>start_x, start_y</i>	The x- and y-coordinates of the start point of the mouse drag in pixels. Coordinates are relative to the upper left corner of the client area of the window, and not to the screen.
<i>end_x, end_y</i>	The x- and y-coordinates of the end point of the mouse drag in pixels. Coordinates are relative to the upper left corner of the client area of the window, and not to the screen.
<i>mouse_button</i>	A constant specifying the mouse button to click (LEFT, MIDDLE, RIGHT). If no <i>mouse_button</i> is specified, the default is the one performing the selection.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_mouse_move

Context Sensitive • Window Object

moves the mouse pointer to the designated position within a window.

win_mouse_move (*window*, *x*, *y*);

<i>window</i>	The logical name of the window.
<i>x</i> , <i>y</i>	The position of the mouse pointer, expressed as <i>x</i> and <i>y</i> (pixel) coordinates. The coordinates are relative to the upper left corner of the client area of the window, and not to the screen.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_move

Context Sensitive • Window Object

moves a window to a new absolute location.

win_move (*window*, *x*, *y*);

<i>window</i>	The logical name of the window.
<i>x</i> , <i>y</i>	The <i>x</i> and <i>y</i> coordinates are relative to the upper left corner of the screen.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_move_locator_text

Context Sensitive • Window Object

moves the mouse pointer to a string in a window.

```
win_move_locator_text ( window, string [ , search_area [ , string_def ] ] );
```

<i>window</i>	The logical name of the window.
<i>string</i>	The text to locate. To specify a literal, case sensitive string, enclose the string in quotation marks. Alternatively, you can specify the name of a string variable. The value of the string variable can include a regular expression (the regular expression need not begin with an exclamation mark).
<i>search_area</i>	The region of the object to search, relative to the window. This area is defined as a pair of coordinates, with x1,y1,x2,y2 specifying any two diagonally opposite corners of the rectangular search region. If this parameter is not defined, then the entire window specified is considered the search area.
<i>string_def</i>	Defines how the text search is performed. If no <i>string_def</i> is specified, (0 or FALSE, the default parameter), the interpreter searches for a complete word only. If 1, or TRUE, is specified, the search is not restricted to a single, complete word.

The Analog version of this function is **move_locator_text**.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_open

Context Sensitive • Window Object

opens an application window.

win_open (*window*);

window the logical name of the window.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_resize

Context Sensitive • Window Object

resizes a window.

win_resize (*window, width, height*);

window The logical name of the window.

width The new width of the window, in pixels.

height The new height of the window, in pixels.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_restore

Context Sensitive • Window Object

restores a window to its previous size.

win_restore (*window*);

window The logical name of the window.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_type

Context Sensitive • Window Object

sends keyboard input to a window.

win_type (*window*, *keyboard_input*);

window The logical name of the window.

keyboard_input A string expression that represents keystrokes.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_wait_bitmap

Context Sensitive • Window Object

waits for a window bitmap.

Note: This function is provided for backward compatibility only. You should use the **win_check_bitmap** and **obj_check_bitmap** functions instead of this function.

win_wait_bitmap (*window*, *bitmap*, *time* [, *x*, *y*, *width*, *height*]);

<i>window</i>	The logical name of the window.
<i>bitmap</i>	A string expression identifying the captured bitmap.
<i>time</i>	The <i>time</i> is added to the <i>timeout_msec</i> testing option to give the maximum interval between the previous input even and the screen capture.
<i>x</i> , <i>y</i>	For an area bitmap: the coordinates of the upper left corner, relative to the window in which the selected region is located in pixels.
<i>width</i> , <i>height</i>	For an area bitmap: the size of the selected region, in pixels.

For an Analog version of the **win_wait_bitmap**, see **wait_window**.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is always available.

win_wait_info

Context Sensitive • Window Object

waits for the value of a window property.

win_wait_info (*window*, *property*, *value*, *time*);

<i>window</i>	The logical name of the window.
<i>property</i>	Any of the properties listed in the <i>User's Guide</i> .
<i>value</i>	The property value for which the function waits.
<i>time</i>	The interval, in seconds, before the next statement is executed.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is always available.

Y2K_age_string

Context Sensitive • Year 2000

ages a date string and returns the aged date.

Y2K_age_string (*date*, *years*, *month*, *days*, *new_date*);

<i>date</i>	The date to age.
<i>years</i>	The number of years to age the date.
<i>month</i>	The number of months to age the date.
<i>days</i>	The number of days to age the date.
<i>new_date</i>	The new date after the date string is aged the specified number of years, months, and days.

Return Values

This function returns 0 if it succeeds; -1 if it fails.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_align_day**Context Sensitive • Year 2000**

ages dates to a specified day of the week or type of day.

Y2K_align_day (*align_mode*, *day_in_week*);

align_mode

You can select one of the following modes:

Mode	Description
NO_CHANGE	No change is made to the aged dates.
BUSINESSDAY_BACKWARD	Ages dates to the closest business day before the actual aged date. For example, if the aged date falls on Saturday, WinRunner changes the date so that it falls on Friday.
BUSINESSDAY_FORWARD	Ages dates to the closest business day after the actual aged date. For example, if the aged date falls on a Saturday, WinRunner changes the date so that it falls on a Monday.
DAYOFWEEK_BACKWARD	Ages dates to the closet week day before the actual aged date. For example, if the aged date falls on a Sunday, WinRunner changes the date so that it falls on a Friday.
DAYOFWEEK_FORWARD	Ages dates to the closest week day after the actual aged date. For example, if the aged date falls on a Sunday, WinRunner changes the date so that it falls on a Monday.
SAMEDAY_BACKWARD	Ages dates to the same day of the week, occurring before the actual aged date. For example, if the original date falls on a Thursday, and the aged date falls on a Friday, WinRunner changes the date so that it falls on the Thursday before the Friday.
SAMEDAY_FORWARD	Ages dates to the same day of the week, occurring after the actual aged date. For example, if the original date falls on a Thursday, and the aged date falls on a Friday, WinRunner changes the date so that it falls on the Thursday after the Friday.

day_in_week A day of the week (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, or Sunday.) This parameter is only necessary when the DAYSOFWEEK_BACKWARD or DAYSOFWEEK_FORWARD option is specified for *align_mode*.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_calc_days_in_field

Context Sensitive • Year 2000

calculates the number of days between two date fields.

Y2K_calc_days_in_field (*field_name₁*, *field_name₂*);

field_name₁ The name of the 1st date field.

field_name₂ The name of the 2nd date field.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_calc_days_in_string

Context Sensitive • Year 2000

calculates the number of days between two numeric strings.

Y2K_calc_days_in_string (*string*₁, *string*₂);

*string*₁ The name of the 1st string.

*string*₂ The name of the 2nd string.

Return Values

This function returns 0 if it succeeds; -1 if it fails.

Availability

This function is supported for WinRunner 2000 only.

Y2K_change_field_aging

Context Sensitive • Year 2000

overrides the aging on a specified date object.

Y2K_change_field_aging (*field_name*, *aging_type*, *days*, *months*, *years*);

field_name The name of the date object.

aging_type The type of aging to apply to the date object:
INCREMENTAL: Ages the date a specified number of days, months, and years.
STATIC: Ages the date to a specific date, for example, "9, 2, 2005" (February 9, 2005). Note that the year must be in YYYY format.

DEFAULT_AGING: Ages the date using the default aging applied to the entire test, and ignores the days, months, and years parameters.

days The number of days to increment the test script.

months The number of months to age the test script.

years The number of years to age the test script.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_change_original_new_formats**Context Sensitive • Year 2000**

overrides the automatic date format for an object.

Y2K_change_original_new_formats (*object_name*, *original_format*, *new_format* [, TRUE|FALSE]);

<i>object_name</i>	The name of the object.
<i>original_format</i>	The original date format used to identify the object.
<i>new_format</i>	The new date format used to identify the object.
TRUE FALSE	TRUE tells WinRunner to use the original date format. FALSE (default) tells WinRunner to use the new date format. This parameter is optional.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_check_date

Context Sensitive • Year 2000

checks all dates in the current screen of a terminal emulator application.

Y2K_check_date (*filename*);

filename The file containing the expected results of the date checkpoint.

Return Values

This function return 0 if it succeeds or 1 if it fails.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_disable_format

Context Sensitive • Year 2000

disables a date format.

Y2K_disable_format (*format*);

format The name of a date format or "ALL" to choose all formats.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_enable_format

Context Sensitive • Year 2000

enables a date format.

Y2K_enable_format (*format*);

format The name of a date format.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_field_to_Julian

Context Sensitive • Year 2000

translates a date field to a Julian number.

Y2K_field_to_Julian (*date_field*);

date_field The name of the date field.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_is_date_field

Context Sensitive • Year 2000

determines whether a field contains a valid date.

Y2K_is_date_field (*field_name*, *min_year*, *max_year*);

<i>field_name</i>	The name of the field containing the date.
<i>min_year</i>	Determines the minimum year allowed.
<i>max_year</i>	Determines the maximum year allowed.

Return Values

This function returns 1 if the field contains a valid date and 0 if the field does not contain a valid date.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_is_date_string

Context Sensitive • Year 2000

determines whether a string contains a valid date.

Y2K_is_date_string (*string*, *min_year*, *max_year*);

<i>string</i>	The numeric string containing the date.
<i>min_year</i>	Determines the minimum year allowed.
<i>max_year</i>	Determines the maximum year allowed.

Return Values

This function returns 1 if the string contains a valid date and 0 if the string does not contain a valid date.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_is_leap_year

Context Sensitive • Year 2000

determines whether a year is a leap year.

Y2K_is_leap_year (*year*);

year A year, for example "1998".

Return Values

This function returns 1 if a year is a leap year, or 0 if it is not.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_leading_zero

Context Sensitive • Year 2000

determines whether to add a zero before single-digit numbers when aging and translating dates.

Y2K_leading_zero (*mode*);

mode One of two modes can be specified: ON or OFF.

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_month_language

Context Sensitive • Year 2000

sets the language used for month names.

Y2K_month_language (*language*);

language The language used for month names.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_set_aging

Context Sensitive • Year 2000

sets aging in the test script.

Y2K_set_aging (*format, type, days, months, years*);

format The date format to which aging is applied (default is ALL).

aging_type The type of aging to apply to the test script:
INCREMENTAL: Ages the test script a specified number of days, months, and years.

STATIC: Ages the test script to a specific date, for example, "9, 2, 2005" (February 9, 2005).

DEFAULT_AGING: Ages the test script using the default aging applied to the entire test, and ignores the days, months, and years parameters.

days The number of days to increment the test script.

months The number of months to age the test script.

years The number of years to age the test script.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_set_attr

Context Sensitive • Year 2000

sets the record configuration mode for a field.

Y2K_set_attr (*index*);

<i>index</i>	The record configuration mode (INDEX or ATTACHED TEXT).
--------------	---

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_set_auto_date_verify

Context Sensitive • Year 2000

automatically generates a date checkpoint for the current screen in a terminal emulator application.

Y2K_set_auto_date_verify (ON|OFF);

ON OFF	If ON, WinRunner automatically generates a date checkpoint for the current screen.
--------	--

The **Y2K_set_auto_date_verify** function automatically captures all date information in a screen of a terminal emulator window and inserts a date checkpoint in the test script.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_set_capture_mode**Context Sensitive • Year 2000**

determines how WinRunner 2000 captures dates in terminal emulator applications.

Y2K_set_capture_mode (*mode*);

mode

The date capture mode. Use one of the following modes:

FIELD_METHOD: Captures dates in the context of the screens and fields in your terminal emulator application (Context Sensitive). This is the default mode.

POSITION_METHOD: Identifies and captures dates according to the unformulated view of the screen.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_set_replay_mode

Context Sensitive • Year 2000

sets the Year 2000 run mode in the test script.

Y2K_set_replay_mode (*mode*);

<i>mode</i>	The Year 2000 run mode. Use one of the following modes: NO_CHANGE: No change is made to objects containing dates during the test run. AGE: Performs aging during the test run. TRANSLATE: Translates dates to the new date format. TRANSLATE_AND_AGE: Translates date formats and performs aging.
-------------	---

Return Values

This function returns one of a list of return values. For more information, see "General Return Values," on page 118.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_set_system_date

Context Sensitive • Year 2000

sets the system date and time.

Y2K_set_system_date (*year*, *month*, *day* [, *hour*, *minute*, *second*]);

<i>year</i>	The year, for example, "2005".
<i>month</i>	The month, for example, "8" (August).
<i>day</i>	The day, for example, "15".
<i>hour</i>	The hour, for example, "2". (optional)
<i>minute</i>	The minute, for example, "15". (optional)
<i>second</i>	The second, for example, "30". (optional)

Return Values

This function always returns 0.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_set_year_limits**Context Sensitive • Year 2000**

sets the minimum and maximum years valid for date verification and aging.

Y2K_set_year_limits (*min_year*, *max_year*);

<i>min_year</i>	The minimum year to be used during date verification and aging.
<i>max_year</i>	The maximum year to be used during date verification and aging.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_set_year_threshold**Context Sensitive • Year 2000**

sets the year threshold.

Y2K_set_year_threshold (*number*);

<i>number</i>	The threshold number.
---------------	-----------------------

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner 2000 only.

Y2K_string_to_Julian

Context Sensitive • Year 2000

translates a string to a Julian number.

Y2K_string_to_Julian (*string*);

string The numeric date string.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Y2K_type_mode

Context Sensitive • Year 2000

disables overriding of automatic date recognition for all date objects in a GUI application.

Y2K_type_mode (*mode*);

mode The type mode. Use one of the following modes:
DISABLE_OVERRIDE: Disables all overrides on date objects.
ENABLE_OVERRIDE: Enables all overrides on date objects.

Return Values

This function returns one of a list of return values. For more information, see “General Return Values,” on page 118.

Availability

This function is supported for WinRunner with Year 2000 Add-in support only.

Index

Symbols

! operator 14
!= operator 13
&& operator 14
< operator 13
<= operator 13
== operator 13
> operator 13
>= operator 13
|| operator 14

A

Acrobat Reader vi
ActiveBar functions 60
ActiveBar_combo_select_item function 126
ActiveBar_dump function 127
ActiveBar_select_menu function 128
ActiveBar_select_tool function 129
ActiveX functions 60
ActiveX_activate_method function 130
ActiveX_get_info function 131
ActiveX_set_info function 132
add_cust_record_class function 134
add_dlp_obj function 135
add_record_attr function 136
add_record_message function 136
ampersand character 13
Analog functions
 by category 56–58
 coordinate conventions 3
 numbering conventions 3
 overview 3
and operator 14
arithmetic functions 101

arithmetical operators 12
 applying to string 9
 assignment 15
array operator 26
arrays 22–28
 declaration 23
 for loop 27
 functions 28, 101
 initializing 25
 multidimensional 25
 operator 26
ascii function 137
assignment operators 15
associativity 16
atan2 function 137
auto 51
auto variables 10

B

bitmap checkpoint functions
 Analog 56
 Context Sensitive 61
break statement 21
built-in functions 29–30
 return value 29
 syntax 29
button object functions 61
button_check_enabled 51
button_check_info function 137
button_check_state function 138
button_get_info function 138
button_get_state function 139
button_get_value 51
button_press function 139
button_set function 140
button_wait_info function 140

C

calendar function 62
calendar_activate_date function 141
calendar_get_selected function 142
calendar_get_status function 142
calendar_get_valid_range function 143
calendar_select_date function 144
calendar_select_range function 144
calendar_select_time function 145
calendar_set_status function 145
call statement 146
call statements 102
call_chain_get_attr statement 147
call_chain_get_depth statement 147
call_close statement 148
call_ex statement 149
case 51
char 51
check_file 51
check_wid 51
check_window function 150
click function 151
click_on_text function 151
comments 29
compare_text function 152
compiled module functions 102
concatenation operator 13
conditional operator 14
const 51
constant declarations 11
constants 7–11
Context Sensitive functions
 by category 58–97
 numbering conventions 4
 object naming conventions 4
 overview 3
continue 51
continue statement 22
control flow 18–22
 break statement 21
 continue statement 22
 do statement 21
 for statement 20
 if-else statement 18
 loop statement 21
 switch statement 19

 while statement 20
coordinate conventions, Analog functions 3
cos function 153
create_browse_file_dialog function 153
create_custom_dialog function 154
create_input_dialog function 155
create_list_dialog function 155
create_password_dialog function 156
custom record functions 98
custom user interface functions 98
Customization functions
 by category 97–99
 overview 5

D

Data Junction 63
data objects 7
database functions 63
 for working with Data Junction 63
 return values for 122
data-driven test functions 64
datawindow_text_click function 157
datawindow_text_dbl_click function 158
db_check function 158
db_connect function 159
db_disconnect function 160
db_dj_convert function 160
db_execute_query function 161
db_get_field_value function 161
db_get_headers function 162
db_get_last_error function 162
db_get_row function 163
db_record_check function 164
db_write_records function 165
dbl_click function 165
ddt_close function 166
ddt_export function 167
ddt_get_current_row function 168
ddt_get_parameters function 168
ddt_get_row_count function 169
ddt_is_parameter function 170
ddt_next_row function 170
ddt_open function 171
ddt_report_row function 171
ddt_save function 172

ddt_set_row function 172
 ddt_set_val function 173
 ddt_set_val_by_row function 174
 ddt_show function 175
 ddt_update_from_db function 175
 ddt_val function 176
 ddt_val_by_row function 177
 declare_rendezvous function 177
 declare_transaction function 178
 default 51
 define_object_exception function 178
 define_popup_exception function 179
 define_tsl_exception function 180
 delete function 28, 180
 delete_record_attr function 181
 Delphi functions 65
 display_date_result 51
 display_euro_result 51
 dlph_button_panel_press function 184
 dlph_edit_set function 182
 dlph_list_select_item function 182
 dlph_obj_get_info function 183
 dlph_obj_set_info function 183
 do statement 21
 dos_system function 185
 double 51

E

edit object functions 65
 edit_activate function 186
 edit_check_content 51
 edit_check_format 51
 edit_check_info function 186
 edit_check_selection function 187
 edit_check_text function 187
 edit_delete function 188
 edit_delete_block function 188
 edit_get_block function 189
 edit_get_info function 190
 edit_get_row_length function 190
 edit_get_rows_count function 191
 edit_get_selection function 191
 edit_get_selection_pos function 192
 edit_get_text function 193

edit_insert function 193
 edit_insert_block function 194
 edit_replace function 194
 edit_replace_block function 195
 edit_set function 195
 edit_set_focus function 196
 edit_set_insert_pos function 196
 edit_set_selection function 197
 edit_type function 197
 edit_wait_info function 198
 else 51
 end_transaction function 198
 endif 51
 equal to (relational) operator 13
 error_message function 199
 EURO functions 67
 EURO_check_currency function 199
 EURO_compare_columns function 200
 EURO_compare_fields function 201
 EURO_compare_numbers function 202
 EURO_convert_currency function 204
 EURO_override_field function 205
 EURO_set_auto_currency_verify function
 206
 EURO_set_capture_mode function 207
 EURO_set_conversion_mode function 207
 EURO_set_conversion_rate function 208
 EURO_set_cross_rate function 209
 EURO_set_currency_threshold function 210
 EURO_set_decimals_precision function 210
 EURO_set_original_new_currencies function
 211
 EURO_set_regional_symbols function 212
 EURO_set_triangulation_decimals function
 212
 EURO_type_mode function 213
 eval function 213
 exception handling functions 103
 exception_off function 214
 exception_off_all function 214
 exception_on function 215
 exception_on_print 51
 exit 51
 exp function 215
 expressions 12–16

extern 51
 declarations 32–34
 variables 10

F

file_close function 216
file_compare function 216
file_getline function 217
file_open function 217
file_printf function 218
find_text_function 220
float 51
for statement 20
function 51
Function Generator functions 99
function types, overview 2

G

general return values 118–122
generator_add_category function 221
generator_add_function function 221
generator_add_function_to_category
 function 223
generator_add_subcategory function 223
generator_set_default_function function 224
get_aut_var function 224
get_class_map function 225
get_host_name function 225
get_lang 51
get_master_host_name function 226
get_obj_record_method 51
get_record_attr function 226
get_record_method function 227
get_runner_str 51
get_text function 228
get_time function 229
get_x function 229
get_y function 229
getenv function 230
getline 51
getvar function 230
grab 51
greater than operator 13
greater than or equal to operator 13

gsub 51
GUI checkpoint functions
 Context Sensitive 69
 Customization 99
GUI map configuration functions 69
GUI map editor functions 70
GUI_add function 231
GUI_buf_get_data 51
GUI_buf_get_data_attr 51
GUI_buf_get_desc function 231
GUI_buf_get_desc_attr function 232
GUI_buf_get_logical_name function 233
GUI_buf_new function 233
GUI_buf_set_data_attr 52
GUI_buf_set_desc_attr function 234
GUI_close function 234
GUI_close_all function 235
GUI_data_get_attr 52
GUI_data_set_attr 52
GUI_delete function 235
GUI_desc_compare function 236
GUI_desc_get_attr function 236
GUI_desc_set_attr function 237
GUI_get_name function 237
GUI_get_window function 238
GUI_list_buf_windows function 239
GUI_list_buffers function 239
GUI_list_data_attrs 52
GUI_list_desc_attrs function 240
GUI_list_map_buffers function 240
GUI_list_win_objects function 241
GUI_load function 242
GUI_map_get_desc function 243
GUI_map_get_logical_name function 243
GUI_mark 52
GUI_open function 244
GUI_point_to 52
GUI_replay_wizard 52
GUI_save function 244
GUI_save_as function 245
GUI_set_window function 245
GUI_unload function 246
GUI_unload_all function 246
gui_ver_add_check function 247
gui_ver_add_check_to_class function 247

gui_ver_add_class function 248
 gui_ver_set_default_checks function 248

I

i/o functions 103
 icon object functions 72
 icon_move function 249
 icon_select function 249
 if 52
 if/else statement 18
 in 52
 index function 250
 inout 52
 input device functions 57
 input/output functions 28
 input_to_description_int 52
 int function 250
 invoke_application function 250

J

Java function 72
 java_activate_method function 252
 jco_create function 253
 jco_free function 254
 jco_free_all function 254
 jdc_aut_connect function 255

L

length function 256
 less than operator 13
 less than or equal to operator 13
 list object functions 73
 list_activate_item function 256
 list_check_info function 257
 list_check_item function 257
 list_check_multi_selection 52
 list_check_row_num 52
 list_check_selected function 258
 list_check_selection 52
 list_collapse_item function 258
 list_deselect_item function 259
 list_deselect_range function 259
 list_drag_item function 260
 list_drop_on_item function 260

list_expand_item function 261
 list_extend_item function 262
 list_extend_multi_items function 262
 list_extend_range function 263
 list_get_checked_items function 264
 list_get_column_header function 264
 list_get_info function 265
 list_get_item function 265
 list_get_item_info function 267
 list_get_item_num function 267
 list_get_items_count 52
 list_get_multi_selected 52
 list_get_selected function 268
 list_get_subitem function 269
 list_rename_item function 269
 list_select_item function 270
 list_select_multi_items function 271
 list_select_range function 271
 list_set_item_state function 272
 list_wait_info function 273
 load function 273
 load testing functions 104
 load_16_dll function 274
 load_dll function 275
 log function 275
 logical operators 14
 long 52
 loop modification statements 21
 looping statements 20
 lov_get_item function 276
 lov_select_item function 276
 lr_whoami function 277

M

match function 277
 menu object functions 74
 menu_get_desc function 278
 menu_get_info function 278
 menu_get_item function 279
 menu_get_item_num function 280
 menu_get_items_count 52
 menu_select_item function 280
 menu_verify 52
 menu_wait_info function 281
 method_wizard function 281

- miscellaneous functions 105
- move_locator_abs function 282
- move_locator_rel function 282
- move_locator_text function 283
- move_locator_track function 283
- move_mouse_abs 52
- move_mouse_rel 52
- move_window 52
- mtype function 284
- multidimensional arrays 25

N

- next 52
- not (unary) operator 14
- not equal to (relational) operator 13
- numbering conventions
 - Analog functions 3
 - Context Sensitive functions 4

O

- obj_check_attr 52
- obj_check_bitmap function 285
- obj_check_enabled 52
- obj_check_focused 52
- obj_check_gui function 285
- obj_check_info function 286
- obj_check_label 52
- obj_check_pos 52
- obj_check_size 52
- obj_check_style 52
- obj_click_on_text function 287
- obj_drag function 288
- obj_drop function 288
- obj_exists function 289
- obj_find_text function 290
- obj_get_desc function 291
- obj_get_info function 291
- obj_get_text function 292
- obj_highlight function 293
- obj_mouse_click function 293
- obj_mouse_dbl_click function 294
- obj_mouse_drag function 295
- obj_mouse_move function 296
- obj_move_locator_text function 297

- obj_set_focus 52
- obj_set_info function 298
- obj_type function 299
- obj_verify 52
- obj_wait_bitmap function 299
- obj_wait_info function 300
- object functions 75
- object naming conventions, Context Sensitive functions 4
- online help vii
- online resources vi
- operating system functions 105
- operators 12–16
 - arithmetical 12
 - assignment 15
 - conditional 14
 - precedence and associativity 16
 - relational 13
 - string 13
- or (logical) operator 14
- oracle functions 76
- out 52
- output_message function 301

P

- password functions 106
- password_edit_set function 301
- password_encrypt function 302
- pause function 302
- pause_test 52
- phone functions 89
- phone_append_text function 303
- phone_edit_set function 303
- phone_get_name function 304
- phone_GUI_load function 304
- phone_key_click function 305
- phone_navigate function 305
- phone_sync function 306
- PowerBuilder functions 76
 - return values for 123
- precedence 16
- printf 52
- process_return_value 52
- prvars 52

public 52
public variables 10

Q

qt_force_send_key function 307
qt_reset_all_force_send_key function 307
quad_click 52
QuickTest 2000 functions 106

R

rand function 308
Readme file vi
relational operators 9, 13
reload function 308
rendezvous function 309
report_event 52
report_msg function 310
report_param_msg 52
reserved words 51–53
reset_filter 52
reset_internals 52
return 52
return statement 32, 310
return values 117–124
 for database functions 122
 for PowerBuilder functions 123
 for table functions 123
 for terminal emulator functions 124
 general 118–122

S

sample tests vii
save_report_info 52
scroll object functions 77
scroll_check_info function 312
scroll_check_pos function 312
scroll_drag function 313
scroll_drag_from_min function 313
scroll_get_info function 314
scroll_get_max function 314
scroll_get_min function 315
scroll_get_pos function 315
scroll_get_selected function 316
scroll_get_value 52

scroll_line function 316
scroll_max function 317
scroll_min function 317
scroll_page function 318
scroll_wait_info function 318
set_aut_var function 319
set_class_map function 319
set_filter 52
set_obj_record_method 52
set_record_attr function 320
set_record_method function 321
set_window function 322
setvar function 323
short 52
siebel_click_history function 323
siebel_connect_repository function 324
siebel_get_active_applet function 324
siebel_get_active_buscomp function 325
siebel_get_active_busobj function 326
siebel_get_active_control function 326
siebel_get_active_view function 327
siebel_get_chart_data function 328
siebel_get_control_value function 329
siebel_goto_record function 329
siebel_navigate_view function 330
siebel_obj_get_info function 330
siebel_obj_get_properties function 331
siebel_select_alpha function 332
siebel_set_active_applet function 333
siebel_set_active_control function 333
signed 52
sin function 335
spin object functions 80
spin_down function 335
spin_get_info function 336
spin_get_pos function 336
spin_get_range function 337
spin_max function 337
spin_min function 338
spin_next function 338
spin_prev function 339
spin_set function 339
spin_up function 340
spin_wait_info function 340
split function 28, 341
sprintf function 341

- sqrt function 342
- srand function 342
- Standard functions
 - by category 100–115
 - overview 5
- start_transaction function 343
- statements 17
- static 53
- static text object functions 80
- static variables 10
- static_check_info function 343
- static_check_text function 344
- static_get_info function 344
- static_get_text function 345
- static_wait_info function 345
- statusbar functions 81
- statusbar_get_field_num function 346
- statusbar_get_info function 346
- statusbar_get_text function 347
- statusbar_wait_info function 347
- str_map_logical_to_visual function 348
- string 53
 - functions 107
 - operators 13
- strings 8
- sub 53
- substr function 349
- support information vii
- switch statement 19
- synchronization functions
 - Analog 57
 - Context Sensitive 81
- system function 349

T

- tab object functions 82
- tab_get_info function 350
- tab_get_item function 350
- tab_get_page 53
- tab_get_selected function 351
- tab_get_selected_page 53
- tab_select_item function 351
- tab_select_page 53
- tab_wait_info function 352

- table functions
 - Analog 57
 - Context Sensitive 82
 - for WebTest 93
 - return values for 123
- tbl_activate_cell function 353
- tbl_activate_col function 355
- tbl_activate_header function 356
- tbl_activate_row function 357
- tbl_click_cell function 357
- tbl_dbl_click_cell function 358
- tbl_deselect_col function 359
- tbl_deselect_cols_range function 360
- tbl_deselect_row function 361
- tbl_deselect_rows_range function 361
- tbl_drag function 362
- tbl_extend_col function 363
- tbl_extend_cols_range function 364
- tbl_extend_row function 365
- tbl_extend_rows_range function 366
- tbl_get_cell_coords 53
- tbl_get_cell_data function 367
- tbl_get_cols_count function 369
- tbl_get_column_name function 370
- tbl_get_column_names function 371
- tbl_get_rows_count function 372
- tbl_get_selected_cell function 373
- tbl_get_selected_row function 375
- tbl_select_cells_range function 376
- tbl_select_col_header function 377
- tbl_select_cols_range function 379
- tbl_select_rows_range function 380
- tbl_set_cell_data function 381
- tbl_set_selected_cell function 385
- tbl_set_selected_col function 387
- tbl_set_selected_row function 388
- tbl_synchronize 53

- TDAPI functions
 - defect tracking functions 110
 - design steps functions 109
 - project administration functions 113
 - project connection functions 108
 - test functions 108
 - test plan tree functions 112
 - test run functions 111
 - test set functions 110
 - test step functions 112
- TDAPI functions by category 108–113
 - database administration functions 113
 - database connection functions 108
 - defect tracking functions 110
 - design steps functions 109
 - test functions 108
 - test plan tree functions 112
 - test run functions 111
 - test set functions 110
 - test step functions 112
- tddb_get_step_value function 389
- tddb_get_test_value function 390
- tddb_get_testset_value function 390
- tddb_load_attachment function 391
- TE_add_screen_name_location function 391
- TE_bms2gui function 392
- TE_check_text function 392
- TE_create_filter function 393
- TE_define_sync_keys function 394
- TE_delete_filter function 395
- TE_edit_field function 395
- TE_edit_hidden_field function 396
- TE_edit_screen function 396
- TE_find_text function 397
- TE_force_send_key function 398
- TE_get_active_filter function 398
- TE_get_auto_reset_filters function 399
- TE_get_auto_verify function 400
- TE_get_cursor_position function 400
- TE_get_field_content function 401
- TE_get_filter function 401
- TE_get_merge_rule function 402
- TE_get_refresh_time function 403
- TE_get_screen_name_location function 403
- TE_get_sync_time function 404
- TE_get_text function 404
- TE_get_timeout function 405
- TE_merge_fields function 405
- TE_reset_all_filters function 406
- TE_reset_all_force_send_key function 406
- TE_reset_all_merged_fields function 406
- TE_reset_filter function 407
- TE_reset_screen_name_location function 407
- TE_send_key function 408
- TE_set_auto_reset_filters function 408
- TE_set_auto_transaction function 409
- TE_set_auto_verify function 409
- TE_set_BMS_name_tag function 410
- TE_set_cursor_position function 410
- TE_set_field function 411
- TE_set_filter function 411
- TE_set_filter_mode function 412
- TE_set_record_method function 413
- TE_set_refresh_time function 413
- TE_set_screen_name_location function 414
- TE_set_sync_time function 414
- TE_set_timeout function 415
- TE_set_trailing function 415
- TE_user_attr_comment function 416
- TE_user_reset_all_attr_comments function 416
- TE_wait_field function 417
- TE_wait_string function 417
- TE_wait_sync function 418
- tech 53
- technical support online vii
- terminal emulator functions 84
 - return values for 124
- TestDirector functions 114
- testing option functions 114
- tests, sample vii
- textit statement 419
- text checkpoint functions
 - Analog 58
 - Context Sensitive 88
- time_str function 419
- time-related functions 115
- tl_get_status 53
- tl_set_status 53
- tl_setvar 53

- tl_step function 420
- tl_step_once function 420
- tolower function 421
- toolbar object functions 88
- toolbar_button_press function 422
- toolbar_get_button function 422
- toolbar_get_button_info function 423
- toolbar_get_button_num function 424
- toolbar_get_buttons_count function 423, 425
- toolbar_get_info 53
- toolbar_select_item function 425
- toolbar_wait_info 53
- toupper function 426
- return 53
- return statement 426
- trpl_click 53
- TSL language 7–34
 - introduction 1–5
- TSL Online Reference vii
- tsl_set_module_mark 53
- tsl_test_is_module 53
- type (of constant or variable) 8
- type function 427
- typographical conventions viii

U

- ungrab 53
- unload function 428
- unload_16_dll function 429
- unload_dll function 429
- unsigned 53
- user_data_point function 430
- user-defined functions 30–32
 - class 30
 - declarations 31
 - parameters 30
 - return statement 32

V

- variables 7–11
 - declarations 9–11
 - names 7
 - undeclared 9

- vendor 53
- Visual Basic functions 60
- vuser_status_message 53

W

- wait function 431
- wait_stable_window 53
- wait_window function 432
- WAP functions 89
- Web functions 90
 - See also* WebTest functions
- web_browser_invoke function 433
- web_cursor_to_image function 433
- web_cursor_to_label function 434
- web_cursor_to_link function 434
- web_cursor_to_obj function 435
- web_event function 435
- web_file_browse function 436
- web_file_set function 437
- web_find_text function 437
- web_frame_get_text function 438
- web_frame_get_text_count function 439
- web_frame_text_exists function 439
- web_get_run_event_mode function 440
- web_get_timeout function 440
- web_image_click function 441
- web_label_click function 441
- web_link_click function 442
- web_link_valid function 442
- web_obj_click function 443
- web_obj_get_child_item function 443
- web_obj_get_child_item_count function 444
- web_obj_get_info function 445
- web_obj_get_text function 445
- web_obj_get_text_count function 446
- web_obj_text_exists function 447
- web_password_encrypt function 447
- web_refresh function 448
- web_restore_event_default function 448
- web_set_event function 449
- web_set_run_event_mode function 450
- web_set_timeout function 451
- web_set_tooltip_color function 451
- web_sync function 452
- web_tbl_get_cell_data function 452

- web_url_valid function 453
- WebTest functions 90
 - for tables 93
- What's New in WinRunner help vi
- while statement 20
- win_activate function 454
- win_check_attr 53
- win_check_bitmap function 454
- win_check_gui function 455
- win_check_info function 456
- win_check_label 53
- win_check_pos 53
- win_check_size 53
- win_click_help function 456
- win_click_on_text function 457
- win_close function 458
- win_drag function 458
- win_drop function 459
- win_exists function 459
- win_find_text function 460
- win_get_desc function 461
- win_get_info function 461
- win_get_text function 462
- win_highlight function 463
- win_max function 463
- win_min function 464
- win_mouse_click function 464
- win_mouse_dbl_click function 465
- win_mouse_drag function 466
- win_mouse_move function 467
- win_move function 467
- win_move_locator_text function 468
- win_open function 469
- win_press_cancel 53
- win_press_ok 53
- win_press_return 53
- win_resize function 469
- win_restore function 470
- win_set_focus 53
- win_type function 470
- win_verify 53
- win_wait_bitmap function 471
- win_wait_info function 472
- window object functions 93

- WinRunner
 - context-sensitive help vii
 - online resources vi
 - sample tests vii
- WinRunner Customization Guide vi
- WinRunner Installation Guide vi
- WinRunner Tutorial vi
- WinRunner User's Guide v, vi

Y

- Y2K_age_string function 472
- Y2K_align_day function 473
- Y2K_calc_days_in_field function 474
- Y2K_calc_days_in_string function 475
- Y2K_change_field_aging function 475
- Y2K_change_original_new_formats function 476
- Y2K_check_date function 477
- Y2K_disable_format function 477
- Y2K_enable_format function 478
- Y2K_field_to_Julian function 478
- Y2K_is_date_field function 479
- Y2K_is_date_string function 479
- Y2K_is_leap_year function 480
- Y2K_leading_zero function 480
- Y2K_month_language function 481
- Y2K_set_aging function 481
- Y2K_set_attr function 482
- Y2K_set_auto_date_verify function 482
- Y2K_set_capture_mode function 483
- Y2K_set_replay_mode function 484
- Y2K_set_system_date function 484
- Y2K_set_year_limits function 485
- Y2K_set_year_threshold function 485
- Y2K_string_to_Julian function 486
- Y2K_type_mode function 486
- Year 2000 functions 95



MERCURY INTERACTIVE

Mercury Interactive Corporation

1325 Borregas Avenue
Sunnyvale, CA 94089 USA

Main Telephone: (408) 822-5200

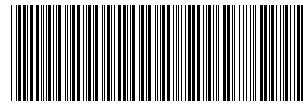
Sales & Information: (800) TEST-911

Customer Support: (877) TEST-HLP

Fax: (408) 822-5300

Home Page: www.mercuryinteractive.com

Customer Support: support.mercuryinteractive.com



* WRTSLREF 7. 0 / 01 *