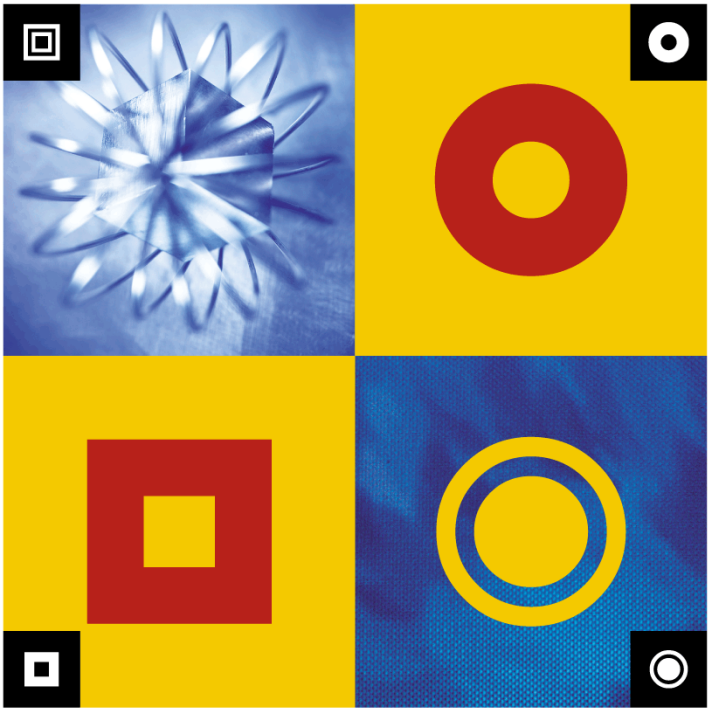
MERCURY INTERACTIVE

WinRunner

TERMINAL EMULATOR ADD-IN



G U I D E

WinRunner Terminal Emulator Add-in Guide

Version 7.6

WinRunner Terminal Emulator Add-in Guide, Version 7.6

Mercury Interactive Corporation
1325 Borregas Avenue
Sunnyvale, CA 94089 USA
Tel: (408) 822-5200
Toll Free: (800) TEST-911, (866) TOPAZ-4U
Fax: (408) 822-5300

If you have any comments or suggestions regarding this document, please send them via e-mail to documentation@merc-int.com.

WRTEGD7.6/01

Table of Contents

Welcome

Welcome to the WinRunner Terminal Emulator Add-in. The WinRunner Terminal Emulator Add-in enables you to record and run tests on mainframe, AS/400, and VAX/HP/UNIX terminal emulator applications using the 3270, 5250, and VT100 protocols respectively, including support for date manipulation testing.

You can use the WinRunner Terminal Emulator Add-in to test applications running on most terminal emulators. The Terminal Emulator Add-in recognizes your terminal emulator, and records and runs the operations you perform on the screens and fields of your terminal emulator application.

Using This Guide

This guide explains everything you need to know to install the WinRunner Terminal Emulator Add-in and to use WinRunner to test terminal emulator applications. It should be used in conjunction with the *WinRunner User's Guide* and the *TSL Online Reference* (or *TSL Reference Guide*).

This guide contains two parts:

Part I Setting Up Terminal Emulator Support

Details the process of installing the WinRunner Terminal Emulator Add-in and configuring your terminal emulator to work with WinRunner, including:

- Before You Install
- Installing and Setting Up the Terminal Emulator Add-in

Part II Working with the Terminal Emulator Add-in

Explains how to use the WinRunner Terminal Emulator Add-in to test terminal emulator applications, including:

- Testing Terminal Emulator Applications
- Synchronizing the Test Run
- Checking Screens and Fields
- Checking Text
- Testing VT100 and Text Applications
- Learning the Application Using BMS Files

Typographical Conventions

This book uses the following typographical conventions:

1, 2, 3	Bold numbers indicate steps in a procedure.
➤	Bullets indicate options and features.
>	The greater than sign separates menu levels (for example, **File** > **Open**).
Stone Sans	The **Stone Sans** font indicates names of interface elements (for example, the **Run** button) and other items that require emphasis.
Bold	**Bold** text indicates method or function names.
Italics	*Italic* text indicates method or function arguments, file names in syntax descriptions, and book titles.
<>	Angle brackets enclose a part of a file path or URL address that may vary from user to user (for example, **<MyProduct installation folder>\bin**).
Arial	The Arial font is used for examples and text that is to be typed literally.
Arial bold	The **Arial bold** font is used in syntax descriptions for text that should be typed literally.
SMALL CAPS	The SMALL CAPS font is used to indicate keyboard keys.
...	In a line of syntax, an ellipsis indicates that more items of the same format may be included. In a programming example, an ellipsis is used to indicate lines of a program that were intentionally omitted.
[ ]	Square brackets enclose optional arguments.
|	A vertical bar indicates that one of the options separated by the bar should be selected.

Part I

Setting Up Terminal Emulator Support

1

Before You Install

This guide describes how to install and use WinRunner with support for terminal emulators.

Before you begin to install the WinRunner Terminal Emulator Add-in, please review the system requirements and additional information in this chapter.

This chapter describes:

- System Requirements
- Preparing for the Terminal Emulator Add-in Installation
- Understanding Add-in Conflicts and Dependencies

System Requirements

To work successfully with the WinRunner Terminal Emulator Add-in, your system configuration should meet the requirements as specified for WinRunner 7.6 (in the WinRunner 7.6 Read Me file), plus the following add-in-specific requirements:

Prerequisites	WinRunner version 7.6
Disk Space	11 MB of free disk space (in addition to the WinRunner installation)
Operating System	Windows® 98, Windows NT® 4.0, Windows® 2000, or Windows® XP

Note: For information on emulators, versions, and protocols supported by the WinRunner Terminal Emulator Add-in, refer to the *WinRunner Terminal Emulator Add-in Read Me*.

Preparing for the Terminal Emulator Add-in Installation

Before you install the WinRunner Terminal Emulator Add-in, you must have a WinRunner 7.6 standalone installation on your computer. For information on installing WinRunner, refer to the *WinRunner Installation Guide*.

Before starting the installation, make sure you know the following:

- The emulator name, for example, RUMBA
- The emulator version, for example 7.0
- The protocol used by the emulator, for example, 3270

If you are not sure of these details, consult your terminal emulator documentation or the vendor of your terminal emulator.

Understanding Add-in Conflicts and Dependencies

You can install all the WinRunner add-ins on a single WinRunner computer. Each time you start WinRunner, you can choose which add-in(s) to load. Loading an add-in may affect the support of other add-ins.

For information on the conflicts that can occur when you load multiple add-ins, refer to the Compatibility Issues help (located in: **<WinRunner Installation path>\dat\conflict.hlp**).

2

Installing and Setting Up the Terminal Emulator Add-in

This chapter explains how to install and set up the WinRunner Terminal Emulator Add-in.

Installing and setting up the Terminal Emulator Add-in includes the following steps:

- Running the Setup Program
- Configuring Your Terminal Emulator
- Setting Your Terminal Emulator to Work with WinRunner
- Licensing the Terminal Emulator Add-in
- Activating WinRunner with Terminal Emulator Support

Running the Setup Program

The setup program installs the WinRunner Terminal Emulator Add-in in your WinRunner installation folder. It also enables you to specify the emulator that you want to use with WinRunner, and to define its settings.

Before you start to install the Terminal Emulator Add-in, be sure to read Chapter 1, “Before You Install.”

Note: You must be logged on with Administrator privileges (if applicable on your operating system) to install the WinRunner Terminal Emulator Add-in.

To run the Terminal Emulator Add-in setup program:

1 Close any instances of WinRunner.

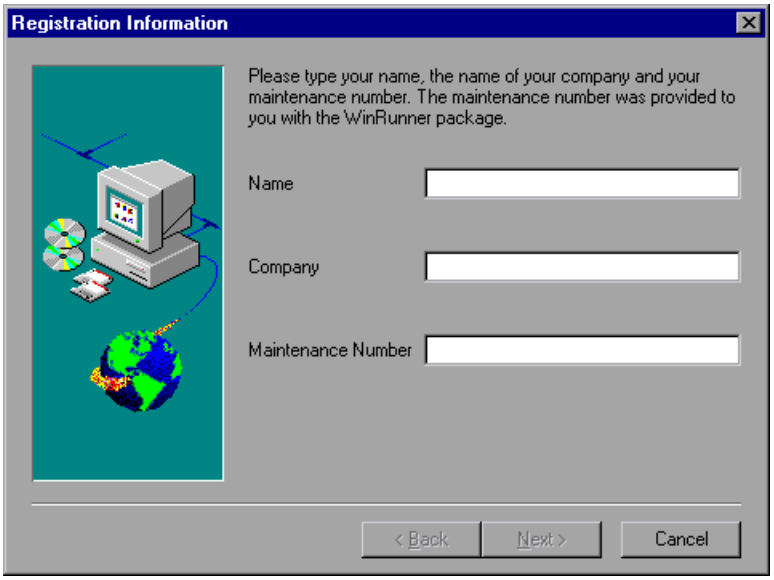
2 Insert the CD-ROM into the CD-ROM drive. If the CD-ROM drive is on your local machine, the Setup window opens. If you are installing from a network drive, connect (map) and browse to it. Double-click **autorun.exe** in the root folder of the CD-ROM.

If you want to view the Readme file, click **Readme**. If you want to browse the CD-ROM, click **Browse**.

3 Click **Install**. The Setup progress bar opens.

4 In the Registration Information screen, type your name, the name of your company, and a WinRunner maintenance number. This number can be found in the Maintenance Pack Number envelope or on the bill of lading you received when you purchased WinRunner.

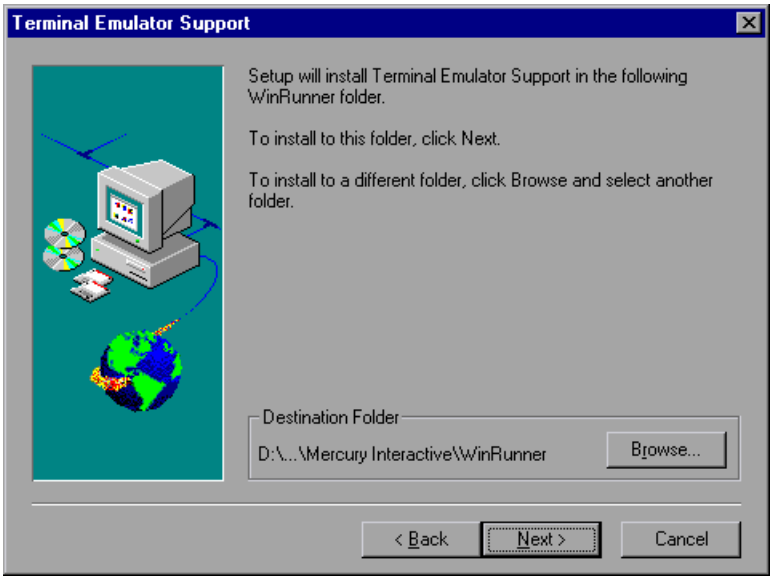
Note: Each team of users is assigned a single maintenance number. The maintenance number replaces what was formerly a serial number for each purchased copy of WinRunner. The maintenance number identifies the customer and determines how many licenses are generated.



Click **Next**.

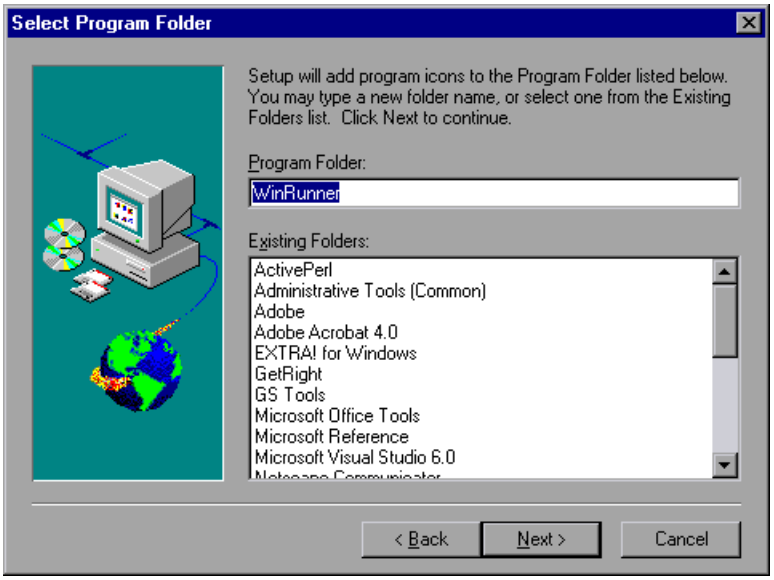
5 Click **Yes** to confirm the registration information.

6 In the Terminal Emulator Support screen, specify the folder in which to install the Terminal Emulator Add-in. The destination folder must be the WinRunner installation folder. If the installation folder displayed is not the WinRunner installation folder, click **Browse** to find the correct destination folder.



Click **Next**.

7 Select the program folder for the Terminal Emulator Add-in program icons.



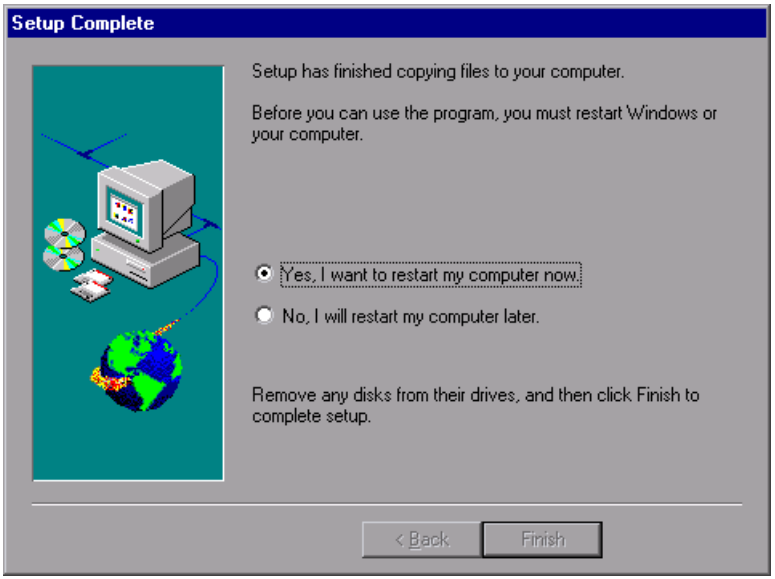
Click **Next**. The setup program begins installing files. To pause or quit the installation, click **Cancel**.

After the WinRunner Terminal Emulator Add-in files have been installed, the Terminal Emulator Configuration utility starts.

8 Follow the procedure for selecting the terminal emulator that you want to use with WinRunner and for defining the emulator settings, as described in “Configuring Your Terminal Emulator” on page 11.

When the Terminal Emulator Configuration utility closes, the Setup Complete screen opens.

The Setup Complete screen prompts you to restart Windows or your computer.



You can choose to restart your computer at a later time, but you must restart your computer before you can use WinRunner with the Terminal Emulator Add-in.

9 Click **Finish** to complete the installation and setup process.

After you have installed the WinRunner Terminal Emulator Add-in:

➤ To read what's new in the WinRunner Terminal Emulator Add-in and any last minute information, select **Start** > **Programs** > **WinRunner** > **Terminal Emulator Add-in** > **Terminal Emulator Add-in Read Me**.

➤ To read about the demo application for date operations support for terminal emulator applications, select **Start** > **Programs** > **WinRunner** > **Sample Applications** > **Date Operations Demo Server Read Me**.

➤ To use the demo application, select **Start** > **Programs** > **WinRunner** > **Sample Applications** > **Date Operations Demo Server**.

Configuring Your Terminal Emulator

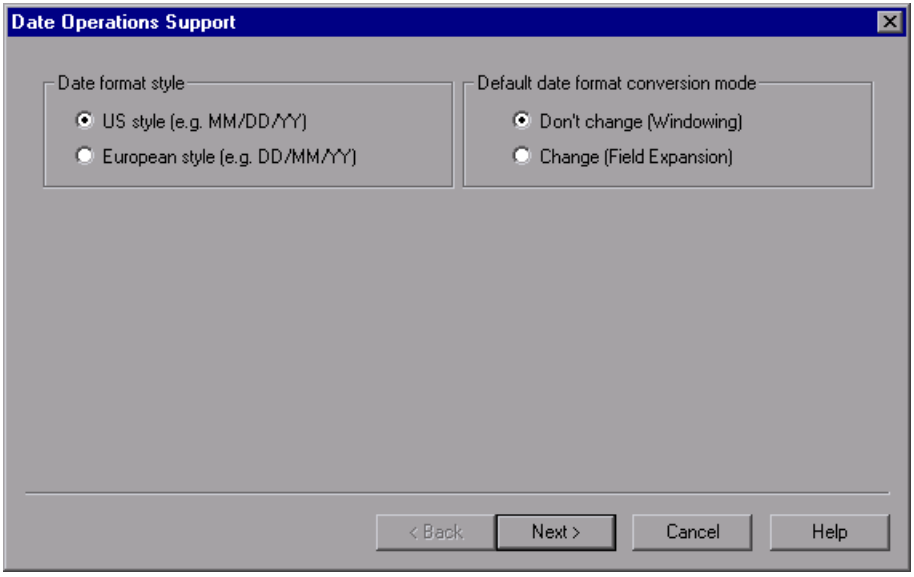
During the installation process, the Terminal Emulator Configuration utility opens, enabling you to specify the terminal emulator that you want to use with WinRunner, and to define its settings.

You can also use the Terminal Emulator Configuration utility after installation, to modify the current settings or select a different terminal emulator. To run the Terminal Emulator Configuration utility, select **Start** > **Programs** > **WinRunner** > **Terminal Emulator Add-in** > **Terminal Emulator Configuration**.

The Terminal Emulator Configuration utility opens with the Date Operations Support screen.

Date Operations Support Screen

WinRunner supports date operations testing for terminal emulator applications. In the Date Operations Support screen, you define the required date format.



Select the date format style:

- **US style**, for example, 11/22/03.
- **European style**, for example, 22/11/03.

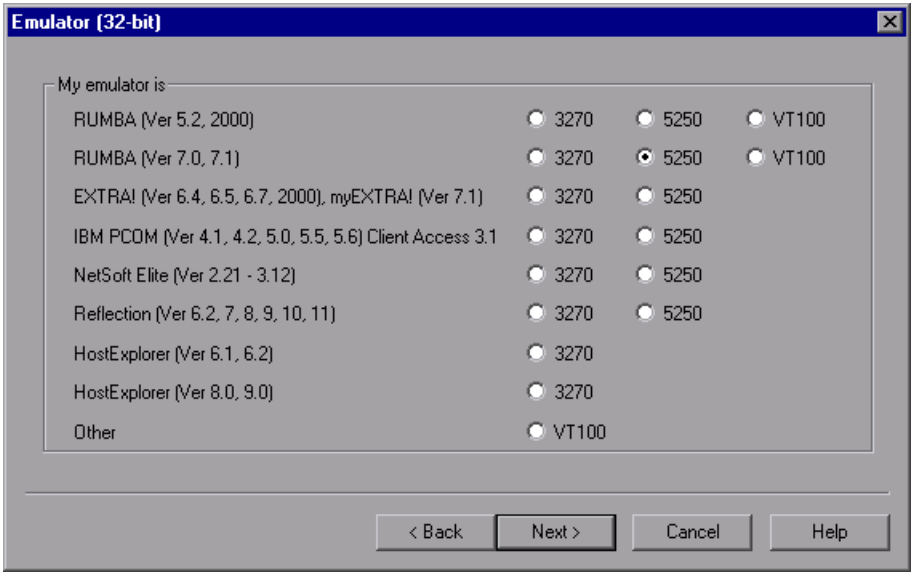
Select the default date format conversion mode used by the application you want to test:

- **Windowing**—The converted year field in the date remains two digits in length, for example, 03.
- **Field Expansion**—The converted year field in the date is expanded to four digits, for example, 2003.

When you click **Next**, the Emulator (32-bit) screen opens.

Emulator (32-bit) Screen

In the Emulator (32-bit) screen, you select the terminal emulator and protocol that you want to use with WinRunner.



Select your emulator and protocol as follows:

- If you are using a mainframe display, select **3270**.
- If you are using an AS/400 display, select **5250**.
- If you are using a UNIX, VAX or HP display, select **VT100**.
- If the current version of your terminal emulator is not displayed, select **Other VT100**.

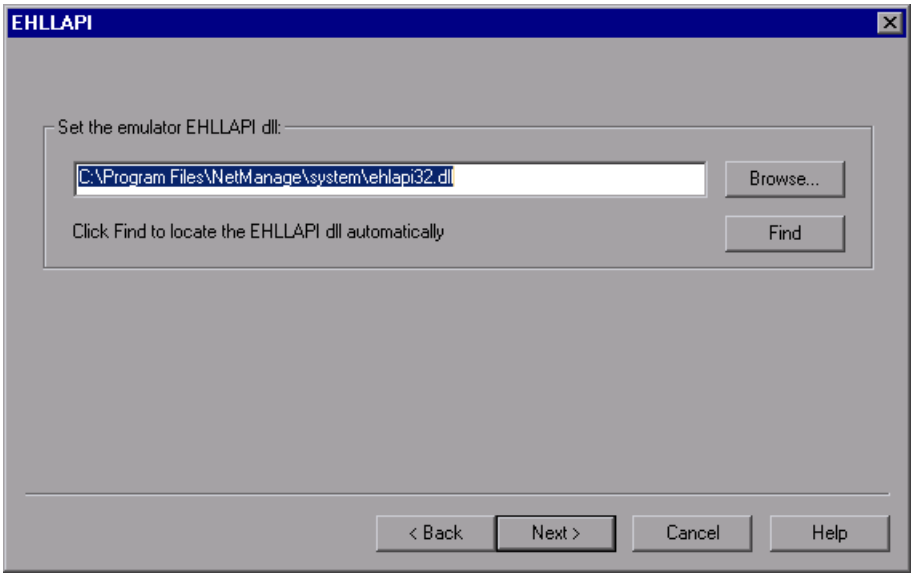
Note: The WinRunner Terminal Emulator Add-in does not support testing on 16-bit terminal emulators. For information on emulators, versions, and protocols supported by the WinRunner Terminal Emulator Add-in, refer to the *WinRunner Terminal Emulator Add-in Read Me*.

When you click **Next**:

- If the selected emulator supports the EHLLAPI protocol, the EHLLAPI screen opens. For more information, see "EHLLAPI Screen" on page 14.
- If you selected **Other VT100**, the Classes screen opens. For more information, see "Classes Screen" on page 15.

EHLLAPI Screen

If you selected an emulator that supports EHLLAPI, you specify the EHLLAPI DLL for your emulator in the EHLLAPI screen.



The table below lists the file names used for the EHLLAPI DLL by the supported terminal emulators.

Emulator	32-Bit EHLLAPI DLL
Attachmate EXTRA!	ehlapi32.dll
Hummingbird HostExplorer	ehllap32.dll
IBM Personal Communications (PCOM)	pcshll32.dll
NetSoft Elite	whllapi.dll
NetManage RUMBA	ehlapi32.dll
WRQ Reflection	hllapi32.dll

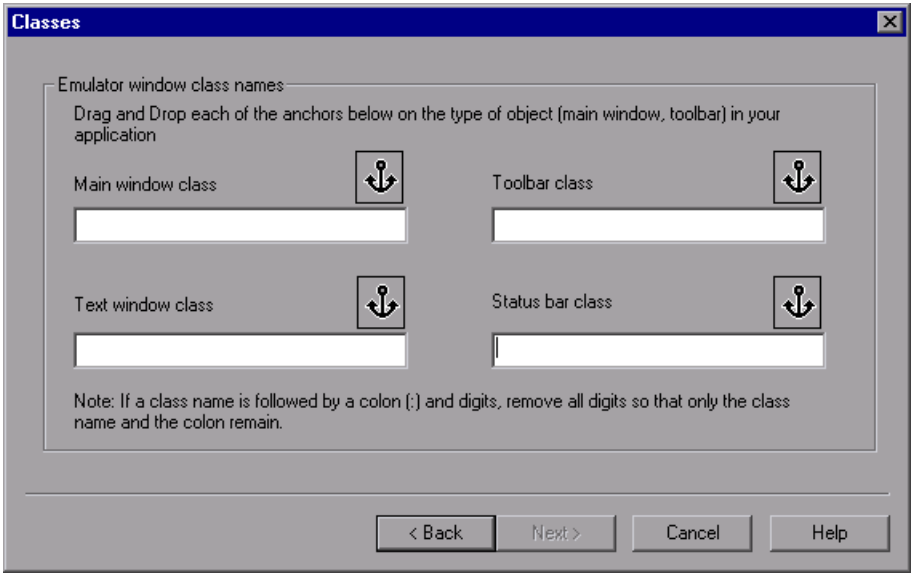
Specify the EHLLAPI DLL for your emulator in one of the following ways:

- Type in the path of the EHLLAPI DLL.
- Click **Browse** and use the Open dialog to locate the EHLLAPI DLL.
- Click **Find**. The Terminal Emulator Configuration utility searches for the appropriate EHLLAPI DLL on your computer.

When you click **Next**, the Miscellaneous screen opens. For more information, see "Miscellaneous Screen" on page 17.

Classes Screen

If you selected **Other VT100** as your terminal emulator type, you specify the classes for your emulator in the Classes screen. This enables WinRunner to identify the components of the terminal emulator window.



You must specify the class for the main window. The other classes are optional.

You can enter the class names automatically or type each class name in the relevant box.

To enter a class name automatically, open your terminal emulator, click an anchor button in the Classes screen, drag the anchor to the appropriate object class, and release the mouse.

Note: If a class name is followed by a colon (:) and digits, remove everything after the colon so that only the class name and the colon remain.
For example, if the class name appears as Afx:00400000:8:00010011: 00000000:069201DF, only Afx: should remain in the class name box.

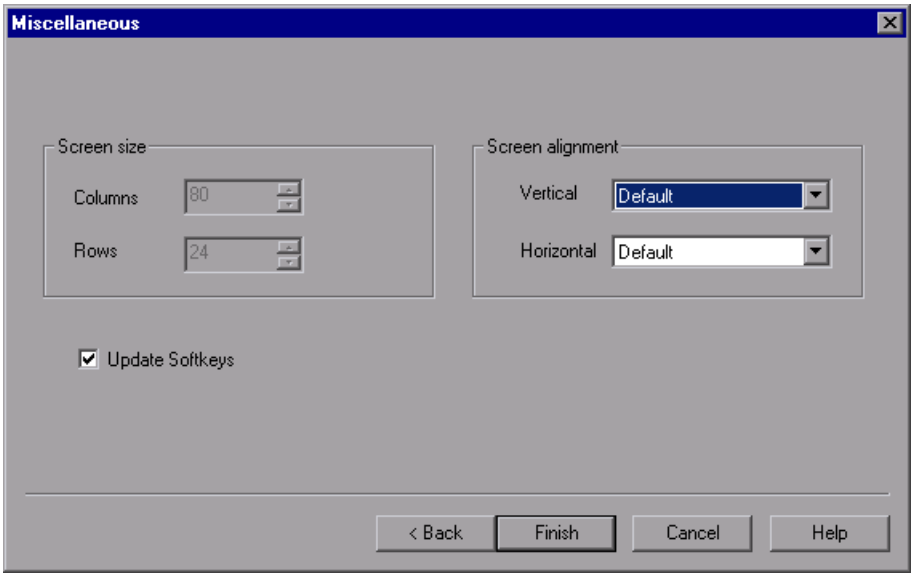
The classes are as follows:

- **Main window class**—The class of the terminal emulator's main window. You must set a value for this option.
- **Text window class**—The class of the terminal emulator's text window (client area). This is the area where text is displayed, and can be the same as the main window.
- **Toolbar class**—The class of the terminal emulator's toolbar (if applicable).
- **Status bar class**—The class of the terminal emulator's status bar (if applicable).

When you click **Next**, the Miscellaneous screen opens.

Miscellaneous Screen

In the Miscellaneous screen, you can define the alignment settings of the screen. You can also determine whether to update the softkey configuration.



Define the vertical and horizontal **Screen alignment** of your terminal emulator, if necessary.

If you selected **Other VT100** as your terminal emulator type, you can also specify the **Screen size**, in terms of number of columns and rows.

Note: The screen size for some emulators is constant (80 x 24) and cannot be changed during installation. To change the size of the screen for these emulators, open the **wrun.ini** file and change the value for the columns and rows manually. This file is located in the Windows system folder.

When the **Update Softkeys** checkbox is selected, the Terminal Emulator Configuration utility updates the current WinRunner softkey configuration with the default softkeys for the selected emulator. These settings can be changed manually if required.

To leave the current WinRunner softkey configuration unchanged, clear the **Update Softkeys** checkbox.

For more information about softkeys and for lists of the default softkeys for different emulators, see "Using Softkeys" on page 38.

When you click **Finish**, the Terminal Emulator Configuration utility closes.

Setting Your Terminal Emulator to Work with WinRunner

Before you can use WinRunner with the Terminal Emulator Add-in, you need to set your terminal emulator to work with WinRunner.

The WinRunner Terminal Emulator Add-in supports the following emulators:

- Attachmate EXTRA!
- Hummingbird HostExplorer
- IBM Personal Communications (PCOM)
- NetManage RUMBA
- NetSoft Elite
- WRQ Reflection

Note: For more information on supported emulators, versions and protocols, refer to the *WinRunner Terminal Emulator Add-in ReadMe*.

Attachmate EXTRA!

To connect your EXTRA! terminal emulator to WinRunner:

1. Load EXTRA!.
2. In EXTRA!, choose **Options** > **Global Preferences**. The Global Preferences dialog box opens.
3. Click the **Advanced** tab.
4. Choose a **Short Name** and click **Browse** to select a previously saved session profile.
5. Click **Open**. Click **OK**.
6. Save the profile when you exit EXTRA!.

Hummingbird HostExplorer

To connect your HostExplorer terminal emulator to WinRunner:

1. Load HostExplorer.
2. From the HostExplorer main menu, choose **File** > **Save Session Profile**.
3. The Save Profile dialog box opens. Set the **HLLAPI Short Name** to any uppercase letter.
4. From the main menu, choose **Options** > **API Settings**.
5. The API Global Settings dialog box opens. Check the **Update screen after PS update** and **Auto sync** options.
6. Click **OK**.

IBM Personal Communications (PCOM)

The preconfigured settings in the WinRunner Terminal Emulator Add-in enable WinRunner to work with IBM PCOM terminal emulators without the need for a specific connection.

NetManage RUMBA

To connect your RUMBA terminal emulator to WinRunner:

1. Load RUMBA.
2. In RUMBA, choose **Options** > **API**. The API Options dialog box opens.
3. Click the **Identification** tab.
4. In the Short Name field, type the uppercase letter **A** for the first session. For subsequent sessions, use consecutive uppercase letters in the alphabet.
5. Click **OK**.
6. Save the profile when you exit RUMBA.

NetSoft Elite

To connect your Elite terminal emulator to WinRunner:

1. Load NS/Administrator.
2. Select **Mainframe Workspace for 3270** or **Midrange Workspace for AS400/5250**.
3. Right-click the appropriate display icon and choose **Properties** in the menu.
4. In the **Short Name** field, type any uppercase letter.
5. Click **OK**.
6. Exit NS/Administrator.
7. Load the NetSoft Display session.
8. In the **View** menu, select **Auto Font Size** and **Auto Window Size**.
9. Choose **Tools** > **Keyboard** to remap the default keys for **New Line** (CTRL RIGHT) and **Reset** (CTRL LEFT). These keys conflict with WinRunner softkeys.

WRQ Reflection

To connect your Reflection terminal emulator to WinRunner:

1 Load Host-Mainframe or AS400.

2 Choose **Setup** > **Terminal**.

3 In the **Short Name** field, type any uppercase letter.

4 Click **OK**.

Licensing the Terminal Emulator Add-in

WinRunner external add-ins (purchased separately) require a separate seat or concurrent license code. For seat licenses, you enter the license details by clicking the **Add-in License** button in the Add-in Manager dialog box. For concurrent licenses, you install the add-in license on the Mercury Functional Testing Concurrent License Server computer.

Note: The WinRunner Add-in license is valid for all WinRunner external add-ins. This means that after you have installed the add-in license for one external add-in, you do not need to install it again if you install additional external add-ins on the same computer.

To access the add-in license installation wizard:

1 Select **Start** > **Programs** > **WinRunner** > **WinRunner**. The WinRunner Add-in Manager dialog box opens.

(If the Add-in Manager dialog box does not open, see the note on displaying the Add-in Manager, on page 25).

Note: If the add-in license has not yet been installed, the add-in is displayed as **(No License)** in the Add-in Manager dialog box.

If the words **(No License)** are not displayed, then the add-in license has already been installed, and you do not need to install it again.

---

2 Click **Add-in License**. The WinRunner License Installation - Welcome window opens.

Note: The **Add-in License** button is displayed only when a WinRunner seat license is installed.

---

3 Install the license. The procedure for installing an add-in license is the same as the procedure for installing a WinRunner license. Refer to the *WinRunner Installation Guide* for further information.

Activating WinRunner with Terminal Emulator Support

You use the Add-in Manager to activate the Terminal Emulator Add-in.

To activate the Terminal Emulator Add-in:

1 Select **Start** > **Programs** > **WinRunner** > **WinRunner**. The WinRunner Add-in Manager dialog box opens.

(If the Add-in Manager dialog box does not open, see the note on displaying the Add-in Manager, on page 25).

2 Select **Terminal Emulator**.

3 Click **OK**. WinRunner opens with the Terminal Emulator Add-in loaded.

Note:

If the Add-In Manager dialog box does not open:

1 Start WinRunner.

2 In the **Tools** > **General Options** > **General** > **Startup** category, select **Display Add-in Manager on startup**. In the **Hide Add-in Manager after ___ seconds** box, enter the number of seconds for which the Add-in Manager is displayed. (The default value is **10** seconds.)

3 Click **OK**.

4 Close WinRunner. A WinRunner message box opens asking whether you want to keep the changes you made. Click **Yes**.

For more information on the Add-in Manager, refer to the *WinRunner User's Guide*.

Disabling Terminal Emulator Support

If you decide to use WinRunner without terminal emulator support, you can disable the Terminal Emulator Add-in.

To disable Terminal Emulator Add-in support:

1 Select **Start** > **Programs** > **WinRunner** > **WinRunner**. The WinRunner Add-in Manager dialog box opens. (If the Add-in Manager dialog box does not open, see the note on displaying the Add-in Manager, on page 25).

2 Clear the **Terminal Emulator** check box and click **OK**. WinRunner opens with the add-in support disabled.

Part II

Working with the Terminal Emulator Add-in

3

Testing Terminal Emulator Applications

This chapter explains how to use WinRunner to record and run tests on terminal emulator applications, using WinRunner's Context Sensitive testing features. For general information on Context Sensitive testing with WinRunner, refer to the *WinRunner User's Guide*.

This chapter describes:

- About Testing Terminal Emulator Applications
- How WinRunner Identifies Terminal Emulator Objects
- Terminal Emulator Object Properties
- Changing How Operations are Recorded
- Using Softkeys

About Testing Terminal Emulator Applications

To create a test for a terminal emulator application, you use WinRunner to record the operations you perform on the application. As you work with objects in the application, WinRunner generates a test script consisting of statements coded in Mercury Interactive's C-like test script language (TSL).

These statements are generated automatically when you record, in response to input to the application. You can program statements manually, or mix recorded and programmed statements in the same test script.

By default, WinRunner records in Context Sensitive mode, meaning that the script reflects the objects on which you operate (such as screens and fields), and the type of operation you perform (such as pressing PF keys or typing in fields). Each object has a defined set of properties that determine its behavior and appearance. WinRunner learns these properties and uses them to identify and locate objects during a test run.

Context Sensitive testing ensures that non-essential changes in your application do not affect the test run. WinRunner can handle changes in the positioning of fields in an application screen.

WinRunner learns an accurate description of each object you are testing and uniquely identifies each screen and field. For more information, see "How WinRunner Identifies Terminal Emulator Objects" on page 31.

Softkeys can be used to activate certain WinRunner commands. For more information, see "Using Softkeys" on page 38.

If you have access to the BMS files of your application, WinRunner can learn your application by reading these files directly. For more information, see Chapter 8, "Learning the Application Using BMS Files."

Before you begin testing your terminal emulator application, make sure that you have installed all the necessary files and made any necessary configuration changes. You configure the terminal emulator settings for WinRunner when you install the WinRunner Terminal Emulator Add-in. You can modify these settings or select a different terminal emulator to work with WinRunner, as required. For more information, see "Configuring Your Terminal Emulator" on page 11.

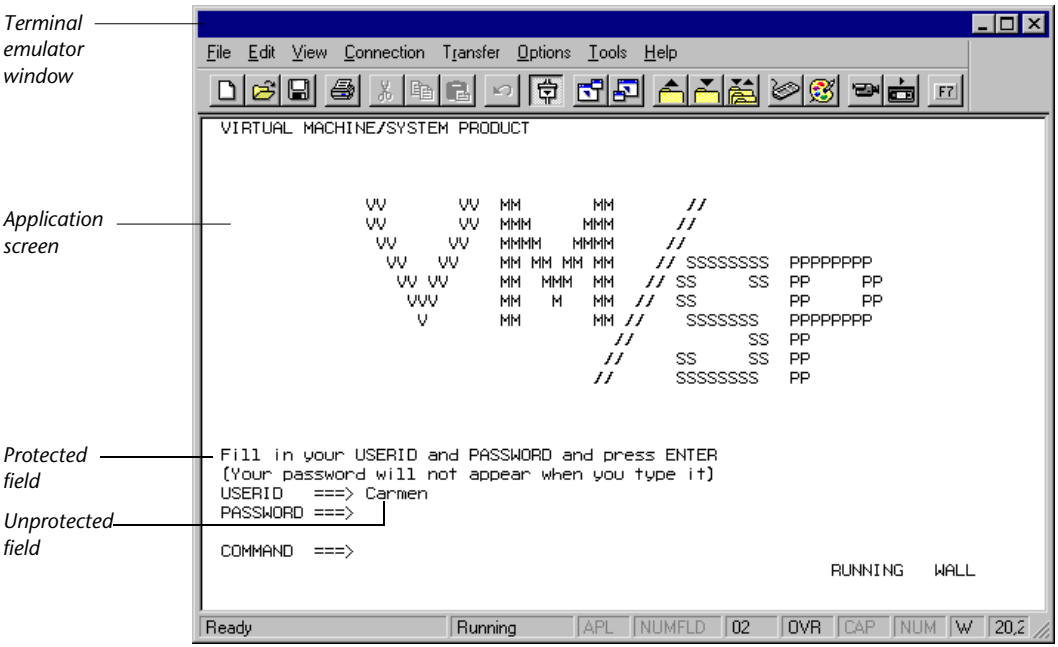
How WinRunner Identifies Terminal Emulator Objects

The WinRunner Terminal Emulator Add-in distinguishes between the window of the terminal emulator and the screens in the host application. For the purposes of testing, the terminal emulator window consists of the frame and menus of the terminal emulator itself. The terminal emulator window remains constant throughout each terminal emulator session.

The WinRunner Terminal Emulator Add-in identifies two types of objects that are specific to terminal emulator applications: *screens* and *fields*.

The *screen* refers to the area of the window in which the application is displayed. It can contain one or more *fields*. Each time the host responds to user input to the application, the screen can change.

A *field* can be either *protected* (containing fixed text) or *unprotected* (able to receive input).



WinRunner identifies the window of the terminal emulator, and its menus, buttons, and status bar, as standard Windows objects. You can perform any standard operations on these objects. For more information about standard operations, refer to the *TSL Online Reference*.

WinRunner learns the description of each object when you record, or when you use the RapidTest Script wizard or the GUI Map Editor. For general information on these options, refer to the *WinRunner User's Guide*.

The description of each screen or field (called the *physical description)* contains a detailed list of properties and values. WinRunner places this list in a GUI map file. In the test script, WinRunner uses a *logical name* for each screen or field as it is displayed in the application.

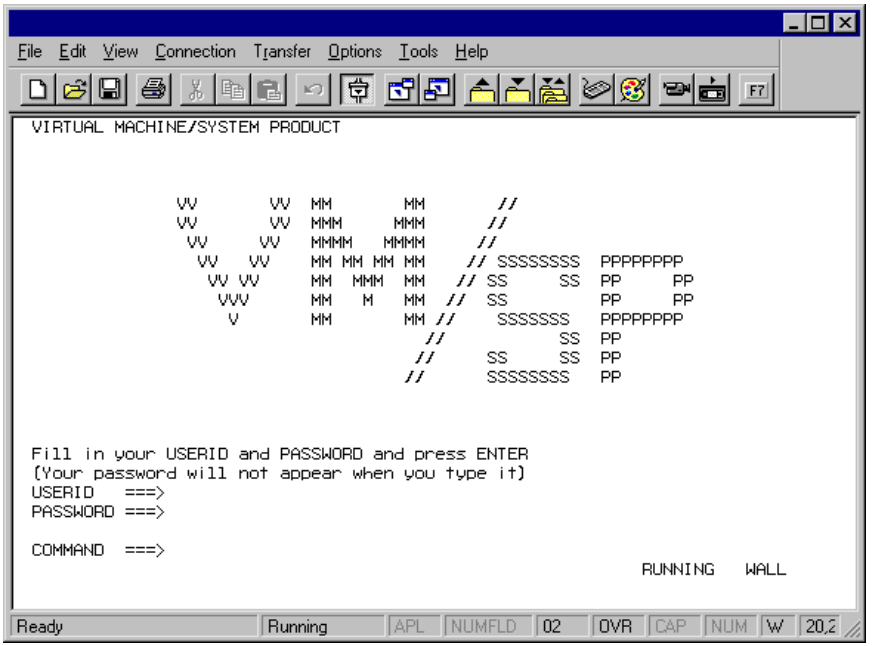
The logical name and the physical description together ensure that each object in your terminal emulator application has its own unique identification.

The physical description of an object contains a list of property–value pairs, as follows:

```
{
property1:value1,
property2:value2,
property3:value3,
...
}
```

The following example illustrates the connection between the logical name and the physical description.

Assume that you record a test in which you type your user ID in the Logon screen of your terminal emulator application.



WinRunner learns the actual description, or list of properties, of both the screen and field on which you performed the operation.

WinRunner learns the following properties and values for the screen:

Screen Property	Value	Comments
class	mic_if_win	Indicates that this is a window or an application screen.
label	VIRTUAL MACHINE/ SYSTEM PRODUCT	The logical name assigned to the screen.
mic_if_handles_ windows	1	Internal property used by WinRunner.

WinRunner learns the following properties and values for the field in which you entered your user ID:

Field Property	Value	Comments
class	field	Indicates that this is a field.
attached_text	USERID	The logical name assigned to the field.

In the test script, WinRunner inserts intuitive logical names for the objects. The logical name is the name WinRunner uses for objects in the test script. Once the name is assigned, you can modify it in the GUI map file. For more information on the GUI map file, refer to the *WinRunner User's Guide*.

The logical name assigned to an object depends on the class of the object. For example, the logical name of a screen is the value of its **label** property. The logical name of a field is the value of its **attached_text** property.

If you start recording and type the user name **Carmen**, the script segment might look like this:

```
set_window ("VIRTUAL MACHINE/SYSTEM PRODUCT");
TE_edit_field("USERID","Carmen");
```

When the test runs, WinRunner reads the logical name of each object from the script and refers to its physical description in the GUI map file. It uses this description to find the object in the terminal emulator application.

WinRunner always learns the *class* property. This indicates the type of the GUI object, such as the terminal emulator window, application screen, or field. For each class, WinRunner learns a set of default properties.

For more information on properties that are unique to terminal emulator objects, see "Terminal Emulator Object Properties" on page 36. For information on other object properties used by WinRunner, refer to the *WinRunner User's Guide*.

Note that WinRunner learns the physical description of an object in the context of the window in which it is displayed. This creates a unique physical description for each object.

The following is a sample of a WinRunner test script recorded on a terminal emulator application. The user presses the ENTER key in the first screen of a terminal emulator application. WinRunner waits for the host to respond and to be ready to receive input to the next screen (Logon). The user types the name **Michael** in the appropriate field in the Logon screen.

The recorded statements show how WinRunner ensures that input is directed to the correct screen. The comment lines (starting with #) describe the statements.

```
# Activate the Terminal Emulator window with the logical name
“RUMBA - DEMO”
win_activate("RUMBA - DEMO");

# Press the Enter key
TE_send_key(TE_ENTER);

# Wait for the host to be ready to receive input
TE_wait_sync();

# Direct input to the Logon screen
set_window("LOGON");

# Type in the user id (“Michael”)
TE_edit_field("USERID","Michael");
```

For more information on TSL functions, refer to the *TSL Online Reference* (**Help** > **TSL Reference**).

Terminal Emulator Object Properties

The following table shows the object properties for application screens and fields. For a full list of properties for all standard Windows objects, refer to the *WinRunner User's Guide*.

Screens

A screen can have the following properties:

Screen Property	Description
class	The prime property that WinRunner uses to identify the type of GUI object. All screens belong to the class **mic_if_win**.
label	The title of the screen. If there is no title, WinRunner assigns a unique number.
protected_fields_number	The number of protected fields in this screen.
input_fields_number	The number of unprotected fields in this screen.
id	A number that WinRunner uses to identify the screen.
mic_if_handles_windows	An internal property that WinRunner uses. The value of this property is always 1.

Fields

A field can have the following properties:

Field Property	Description
class	The prime property that WinRunner uses to identify the type of GUI object. All fields belong to the class **field**.
attached_text	The text that is closest to the field.
protected	A value that indicates whether the field is protected. This value is Yes if the field is protected, or No if unprotected.
visible	A value that indicates whether the contents of the field can be seen: 1 if they are visible, or 0 if not visible.

Field Property	Description
numeric_only	A value that indicates whether the field is numeric. This value is Yes if the field is numeric or No if it is not numeric.
id	A number that WinRunner uses to identify the field.
x	The x-coordinate of the top left corner of a field, relative to the screen origin.
y	The y-coordinate of the top left corner of a field, relative to the screen origin.
length	The length of the field, in characters.
color	A value indicating the color of the field. This can be 0, 1, 2, or 3, depending on the terminal emulator's color definitions.

Changing How Operations are Recorded

When working with 3270 and 5250 protocol terminal emulators that support the EHLLAPI protocol, WinRunner records operations using the *field* or *position* method.

The *field* method (the default) enables WinRunner to record screens, fields, and PF keys using functions such as **TE_edit_field** and **TE_send_key**.

When the *position* method is used, WinRunner records keyboard and mouse input only. The operations on objects in your application are recorded as **win_type**, **obj_type**, **win_mouse_click**, and **win_mouse_drag** statements.

Note: The *method* (field or position) used for recording on terminal emulator applications is not the same as the standard WinRunner *record modes* (Context Sensitive or Analog). You must always use the WinRunner Context Sensitive record mode for recording on terminal emulator applications.

For information on testing terminal emulators that do not support the EHLLAPI protocol, see Chapter 7, "Testing VT100 and Text Applications."

You use the **TE_set_record_method** function to change the emulator recording method. This function has the following syntax:

TE_set_record_method (*method* **);**

The *method* can be one of the following:

- FIELD_METHOD, or (2)—Enables full Context Sensitive recording (default).
- POSITION_METHOD, or (1)—Only keyboard and mouse input is recorded.

The current emulator recording method remains valid until you change it, even after you exit WinRunner and start it again. For more information on TSL functions and syntax, refer to the *TSL Online Reference*.

Using Softkeys

Some WinRunner commands can be activated using softkeys. WinRunner reads input from softkeys even when the WinRunner window is not the active window on your screen, or when it is minimized.

By default, the current softkey configuration is updated when an emulator is selected and configured, during the installation of the Terminal Emulator Add-in or using the Terminal Emulator Configuration utility. For more information, see "Configuring Your Terminal Emulator" on page 11.

Softkey assignments are configurable. If the application you are testing uses one of the default softkeys preconfigured for WinRunner, you can redefine the WinRunner softkey using the WinRunner Softkey Configuration utility. To use this utility, select **Start** > **Programs** > **WinRunner** > **Softkey Configuration**. For more information, refer to the *WinRunner User's Guide*.

The tables in this section show the softkey configurations available for the terminal emulators and protocols supported by the WinRunner Terminal Emulator Add-in.

WinRunner Terminal Emulator Softkeys

The following tables show the softkeys available for testing terminal emulator applications.

Terminal Emulator Command	Softkey for Elite (3270 & 5250)	Softkey for EXTRA! (3270 & 5250)	Softkey for HostExplorer (3270)
CHECK DATE	Right Alt + PgDn	Left Ctrl + F3	Left Ctrl + F3
CHECK PARTIAL DATE	Left Alt + PgUp	Left Alt + F3	Left Alt + F3
CHECK PARTIAL TEXT	Left Alt + PgDn	Left Ctrl + F8	Left Ctrl + F8
CHECK TEXT	Left Alt + F1	Left Ctrl + F2	Left Ctrl + F2
EXCLUDE FILTER	Left Alt + F3	Left Ctrl + F7	Left Ctrl + F7
GET TEXT	Left Alt + F2	Left Alt + F2	Left Alt + F2
INCLUDE FILTER	Left Alt + F7	Left Alt + F7	Left Alt + F7
WAIT STRING	Left Alt + F5	Left Ctrl + F5	Left Ctrl + F5

Terminal Emulator Command	Softkey for PCOM & RUMBA (3270)	Softkey for PCOM & RUMBA (5250 & VT100)	Softkey for Reflection (3270 & 5250)
CHECK DATE	Left Ctrl + PgDn	Left Ctrl + F2	Right Ctrl + F8
CHECK PARTIAL DATE	Left Alt + End	Left Ctrl + F8	Left Ctrl + F8
CHECK PARTIAL TEXT	PgDn	Left Ctrl + F3	Left Ctrl + F11
CHECK TEXT	Left Alt + PgUp	Left Ctrl + F1	Left Ctrl + F2
EXCLUDE FILTER	Left Alt + PgDn	Right Ctrl + F7	Right Ctrl + F7
GET TEXT	Left Ctrl + End	Left Ctrl + F5	Left Ctrl + F5
INCLUDE FILTER	Right Alt + PgDn	Left Ctrl + F7	Left Ctrl + F7
WAIT STRING	Right Ctrl + End	Left Ctrl + F12	Right Ctrl + F12

Softkeys for Standard WinRunner Commands

The following tables show the default softkeys for standard WinRunner functions when working with the specified emulators. Note that in some cases the softkey for a selected emulator may be different than the default WinRunner softkey for the command.

Standard WinRunner Command	Softkey for Elite (3270 & 5250)	Softkey for EXTRA! (3270 & 5250)	Softkey for HostExplorer (3270)
Check Bitmap of Window	Left Alt + F11	Left Ctrl + PgUp	Left Ctrl + PgUp
Check Bitmap of Screen Area	Right Alt + 2	Left Ctrl + 2	Left Ctrl + 2
Check Gui for Object/ Window	Left Alt + 2	Right Ctrl + 2	Right Ctrl + 2
Get Text from Screen Area	Right Alt + F10	Left Ctrl + F10	Left Ctrl + F10
Get Text from Window Area	Right Alt + 1	Left Ctrl + 1	Left Ctrl + 1
Get Text from Object/ Window	Right Alt + 9	Left Ctrl + 9	Left Ctrl + 9
Check Gui for Multiple Objects	Right Alt + End	Right Ctrl + F12	Right Ctrl + F12
Insert Function for Object/ Window	Left Alt + 8	Left Alt + 8	Left Alt + 8
Insert Function From Function Generator	Left Alt + 7	Left Alt + 7	Left Alt + 7
Move Locator	Left Alt + 6	Left Alt + 6	Left Alt + 6
Pause	Pause	Pause	Pause
Record	Scroll Lock	Scroll Lock	Scroll Lock
Run from Arrow	Right Alt + 7	Left Ctrl + 7	Left Ctrl + 7

Standard WinRunner Command	Softkey for Elite (3270 & 5250)	Softkey for EXTRA! (3270 & 5250)	Softkey for HostExplorer (3270)
Run from Top	Right Alt + 5	Left Ctrl + 5	Left Ctrl + 5
Step	Right Alt + 6	Left Ctrl + 6	Left Ctrl + 6
Step Into	Right Alt + 8	Left Ctrl + 8	Left Ctrl + 8
Step to Cursor	Left Alt + F10	Left Alt + 9	Left Alt + 9
Stop	Right Alt + 3	Left Ctrl + 3	Left Ctrl + 3
Synchronize Bitmap of Window	Right Alt + 0	Left Ctrl + 0	Left Ctrl + 0
Synchronize Bitmap of Screen Area	Right Alt + 4	Left Ctrl + 4	Left Ctrl + 4
Synchronize Bitmap & Location of Window	Left Alt + F9	Left Alt + F9	Left Alt + F9

Standard WinRunner Command	Softkey for PCOM & RUMBA (3270)	Softkey for PCOM & RUMBA (5250 & VT100)	Softkey for Reflection (3270 & 5250)
Check Bitmap of Window	Left Ctrl + PgUp	Right Ctrl + 0	Left Ctrl + PgUp
Check Bitmap of Screen Area	Left Ctrl + 2	Left Ctrl + 2	Left Ctrl + 2
Check GUI for Object/ Window	Right Ctrl + 2	Right Ctrl + 2	Right Ctrl + 2
Get Text from Screen Area	Left Ctrl + F10	Left Ctrl + F10	Left Ctrl + F10
Get Text from Window Area	Left Ctrl + 1	Left Ctrl + 1	Left Ctrl + 1

Standard WinRunner Command	Softkey for PCOM & RUMBA (3270)	Softkey for PCOM & RUMBA (5250 & VT100)	Softkey for Reflection (3270 & 5250)
GET TEXT FROM OBJECT/ WINDOW	Left Ctrl + 9	Left Ctrl + 9	Left Ctrl + 9
CHECK GUI FOR MULTIPLE OBJECTS	Right Ctrl + F12	Right Ctrl + F12	Right Ctrl + F12
INSERT FUNCTION FOR OBJECT/ WINDOW	Left Alt + 8	Left Alt + 8	Left Alt + 8
INSERT FUNCTION FROM FUNCTION GENERATOR	Left Alt + 7	Left Alt + 7	Left Alt + 7
MOVE LOCATOR	Right Ctrl + 6	Right Ctrl + 6	Right Ctrl + 6
PAUSE	Pause	Pause	Pause
RECORD	Scroll Lock	Left Alt + 2 (5250) Scroll Lock (VT100)	Left Alt + 2
RUN FROM ARROW	Left Ctrl + 7	Left Ctrl + 7	Left Ctrl + 7
RUN FROM TOP	Left Ctrl + 5	Left Ctrl + 5	Left Ctrl + 5
STEP	Left Ctrl + 6	Left Ctrl + 6	Left Ctrl + 6
STEP INTO	Left Ctrl + 8	Left Ctrl + 8	Left Ctrl + 8
STEP TO CURSOR	Left Ctrl + F9	Left Ctrl + F9	Left Ctrl + F9
STOP	Left Ctrl + 3	Left Ctrl + 3	Left Ctrl + 3
SYNCHRONIZE BITMAP OF WINDOW	Left Ctrl + 0	Left Ctrl + 0	Left Ctrl + 0
SYNCHRONIZE BITMAP OF SCREEN AREA	Left Ctrl + 4	Left Ctrl + 4	Left Ctrl + 4
SYNCHRONIZE BITMAP & LOCATION OF WINDOW	Left Alt + F9	Left Alt + F9	Left Alt + F9

4

Synchronizing the Test Run

WinRunner enables you to synchronize between the host and the application you are testing during the test run. Synchronization ensures that the test run is delayed until the application is ready to receive new input. This prevents incidental differences in host response time from affecting test runs.

This chapter describes:

- About Synchronizing the Test Run
- Waiting for a Response from the Host
- Waiting for a Specific String
- Waiting for a Specific Field
- Synchronizing Using Time Factors
- Synchronizing Screen Changes

About Synchronizing the Test Run

When using a terminal emulator, many factors can affect the speed of operation and therefore interfere with the test run. Host response time varies depending on the system load. The screen refresh rate of your terminal emulator can also vary. WinRunner provides different types of synchronization points to pace the test run with the system. These points can be inserted into the test script automatically, using a softkey, or by programming.

Waiting for a Response from the Host

While recording on an emulator that supports the EHLLAPI protocol, WinRunner automatically generates the following statement each time the terminal emulator waits for a response from the host:

```
TE_wait_sync ( );
```

During a test run, this statement ensures that the test run is delayed until the host responds and the application is ready to receive input.

Note: The **TE_wait_sync** function is available only for 3270 and 5250 terminal emulators that support the EHLLAPI protocol.

Waiting for a Specific String

Using the **TE_wait_string** function, you can instruct WinRunner to wait for a specific string to appear on the screen before continuing the test run. You can specify an area of the screen, or WinRunner can search the entire screen for the string.

To record a TE_wait_string statement in your test script:

1. While recording, press the WAIT STRING softkey for your emulator. WinRunner is minimized and a help window displays instructions for capturing the string.
2. Click and drag to draw the rectangle within which you want WinRunner to search for the text.

3 Right-click to capture the string. WinRunner is restored and a **TE_wait_string** statement with the following syntax is inserted into your test script:

TE_wait_string (*string* [, *start_column*, *start_row*, *end_column*, *end_row*, [, *timeout*]] **);**

The *string* parameter is the text enclosed in the rectangle. If the text you captured exceeds one line, the *string* parameter includes the first line only.

The *start_column* and *start_row* parameters indicate the column/row at which the captured text starts. The *end_column* and *end_row* parameters represent the column and row, respectively, at which the text ends.

The *timeout* parameter is the number of seconds that WinRunner waits for the specified string to appear before continuing the test run.

The following example shows the statement recorded when the text Open the mail is captured using the **Wait String** softkey:

```
TE_wait_string("Open the mail", 8, 4, 20, 4, 60);
```

The value of the first parameter, Open the mail, is the string that WinRunner searches for on the screen; WinRunner looks for this string in row 4, columns 8 through 20. If this string does not appear in the defined location within 60 seconds (the specified timeout period), the test run continues.

When you program this statement, you can omit the coordinates. In this case, WinRunner searches the entire screen for the specified string.

If no *timeout* parameter is specified, WinRunner uses the currently defined timeout period. You can retrieve the value of the currently defined timeout using **TE_get_timeout**, and you can change it using **TE_set_timeout**. For more information, see “Setting and Retrieving the Timeout” on page 47.

Waiting for a Specific Field

Using the **TE_wait_field** function, you can instruct WinRunner to wait for a specific field to appear on the screen before continuing the test run. When the field is displayed, WinRunner resumes the test run. The syntax for this function is:

TE_wait_field (*field_logical_name*, *content* [, *timeout*] **);**

The *field_logical_name* parameter is the name of the field.

The *content* parameter is the string contained in the field.

The *timeout* parameter is the number of seconds that WinRunner waits for the specified field to appear before continuing the test run.

If no *timeout* parameter is specified, WinRunner uses the currently defined timeout period. You can retrieve the value of the currently defined timeout using **TE_get_timeout**, and you can change it using **TE_set_timeout**. For more information, see "Setting and Retrieving the Timeout" on page 47.

Note: The **TE_wait_field** function is available only for 3270 and 5250 terminal emulators that support the EHLLAPI protocol.

Synchronizing Using Time Factors

Two factors that can affect whether the test runs correctly are the response time of the host and the screen refresh rate of your terminal. The following functions enable you to configure WinRunner to handle these variations.

Setting and Retrieving the Screen Refresh Time

The refresh time defines the maximum amount of time (in seconds) that WinRunner waits for the screen to refresh after the host has responded, before continuing the test run.

By default, WinRunner waits a maximum of one second for the screen to refresh. You can change this if needed, using the **TE_set_refresh_time** function, to ensure that WinRunner waits until the screen is completely refreshed before continuing the test run. The time that you set for this function remains in effect until WinRunner closes. Each time you restart WinRunner, the refresh time is reset to the default value of one second.

The syntax for this function is:

TE_set_refresh_time (*time* **);**

To retrieve the value of the currently defined maximum refresh time, use the **TE_get_refresh_time** function. The syntax for this function is:

TE_get_refresh_time ( );

Note: The **TE_set_refresh_time** function is available only for 3270 and 5250 terminal emulators that support the EHLLAPI protocol.

Setting and Retrieving the Timeout

The timeout defines the maximum time (in seconds) that WinRunner waits for a response from the host before continuing the test run.

By default, WinRunner waits a maximum of 60 seconds for the host to respond. You can change this as needed, using the **TE_set_timeout** function. The time that you set for this function remains in effect until WinRunner closes. Each time you restart WinRunner, the timeout value is reset to the default value of 60 seconds.

This statement has the following syntax:

TE_set_timeout (*timeout* **);**

To retrieve the value of the currently defined timeout, use the **TE_get_timeout** function. The syntax for this function is:

TE_get_timeout ( );

Setting and Retrieving the Synchronization Time

The synchronization time defines the minimum amount of time (in seconds) that WinRunner waits for the host to respond. WinRunner always waits for at least this amount of time before checking whether a response has been received from the host.

By default, WinRunner waits one second before checking whether a response has been received from the host. You can use the **TE_set_sync_time** function to change this amount of time.

This statement has the following syntax:

TE_set_sync_time (*time* **);**

If no response is received from the host within the defined synchronization time, WinRunner continues to wait until a response is received or until the timeout has expired.

Note: The synchronization time is included in the timeout period. For example, if the synchronization time is 5 seconds and the timeout period is 60 seconds, the maximum time that WinRunner waits is 60 seconds and not 65 seconds. For more information about the timeout, see "Setting and Retrieving the Timeout" on page 47.

To retrieve the value of the currently specified synchronization time, use the **TE_get_sync_time** function.

This statement has the following syntax:

TE_get_sync_time ( );

Note: The **TE_set_sync_time** and **TE_get_sync_time** functions are available only for 3270 and 5250 terminal emulators that support the EHLLAPI protocol.

Synchronizing Screen Changes

In some AS/400 applications, typing a key in a specific field causes the screen to change. In such a case, WinRunner does not recognize that the screen has changed and does not generate the **TE_wait_sync** function.

In such cases, you can add a **TE_force_send_key** function to your startup test. This function causes WinRunner to recognize that the specified key changes the terminal emulator screen and to generate the **TE_wait_sync** function automatically for that key.

This function has the following syntax:

TE_force_send_key (*in_screen*, *in_field* [, *in_key*] **);**

The *in_screen* parameter defines the screen in which the field exists.

The *in_field* parameter defines the field.

The *in_key* parameter defines the input key that causes the screen to change (optional). You can use a key mnemonic (such as @E for ENTER) or the WinRunner macros (such as TE_Enter for ENTER).

Note: You can include several such statements in your startup test, to specify different fields with special keys. Each **TE_force_send_key** statement should refer to a different field. If you include two **TE_force_send_key** statements with the same screen and field names, the second statement overrides the first.

You can use the **TE_reset_all_force_send_key** to cancel the effect of all **TE_force_send_key** statements until the startup script runs again.

For more information about these functions, refer to the *TSL Online Reference*.

5

Checking Screens and Fields

WinRunner verifies the behavior of your terminal emulator application by comparing the expected results to the actual results that appear when you run the test. By adding GUI checkpoints to your test, you can capture information about screens and fields, and store the information as a basis for comparison.

This chapter describes:

- About Checking Screens and Fields
- Checking the Properties of a Single Field or a Screen
- Checking the Properties of Two or More Fields
- Checking the Default Properties for All Fields in a Screen
- Screen and Field Property Checks
- Checking Dates

About Checking Screens and Fields

You can use GUI checkpoints to check the property values of the objects (screens and fields) in your terminal emulator application. For example, you can check the number of protected or input fields in a screen. You can also check the content of a specific field and whether it is protected or visible. WinRunner captures the current values of the selected properties and saves this information as expected results.

When you run the test, WinRunner compares the actual values of the selected properties to the expected results. If they do not match, the checkpoint fails.

You can use date checkpoints to check how your terminal emulator application performs date operations. For additional information on checking dates, see "Checking Dates" on page 57.

You can also add text checkpoints to your test, to check the text in the screen of your terminal emulator application. For more information, see Chapter 6, "Checking Text."

The information in this chapter applies specifically to GUI checks on terminal emulator applications. For general information on GUI checkpoints, refer to the *WinRunner User's Guide*.

Checking the Properties of a Single Field or a Screen

You can check the properties of a single field or screen by pointing to it and specifying the type of checks you want to perform.

To check the properties of a single field or a screen:

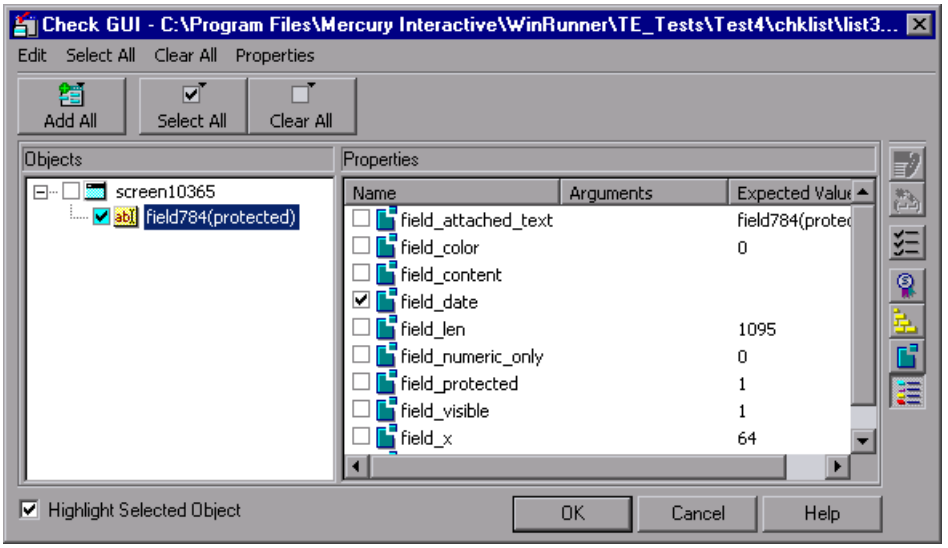
1 Choose **Insert** > **GUI Checkpoint** > **For Object/Window**, or click the **GUI Checkpoint for Object/Window** button on the User toolbar.

2 Double-click on the field or screen you want to check. (To check the screen, double-click on an empty area of the screen.)

Tip: Alternatively, you can single-click the object to perform WinRunner's default checks. The default check for a field is **Date**. The default checks for a screen are **Number of protected fields** and **Number of input fields**.

---

3 The Check GUI dialog box opens.



4 Select the checks you want to perform. For more information, see "Screen and Field Property Checks" on page 56.

5 Click **OK**. WinRunner captures the screen or field information, stores it in the expected results folder for the test, and adds an **obj_check_gui** or a **win_check_gui** statement to your test script.

Checking the Properties of Two or More Fields

You create a checklist to check the properties of two or more fields by clicking on the fields you want to check.

To check the properties of two or more fields in a screen:

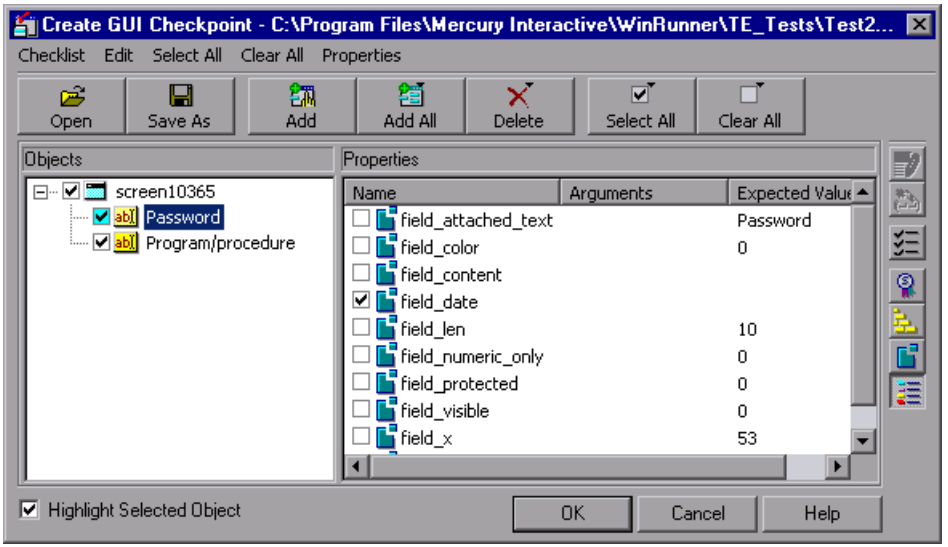
1 Choose **Insert** > **GUI Checkpoint** > **For Multiple Objects**, or click the **GUI Checkpoint for Multiple Objects** button on the User toolbar. The Create GUI Checkpoint window opens.

2 Click the **Add** button.

3 Click each field you want to check.

4 Right-click to stop the selection process. The Create GUI Checkpoint dialog box reopens.

The **Objects** pane lists the name of the screen and the fields you clicked. The **Properties** pane lists the properties for the selected field.



5 To modify a check, select the field in the **Objects** pane and select the properties to be checked in the **Properties** pane. For more information, see "Screen and Field Property Checks" on page 56.

6 Click **OK**. The checklist is saved and the Create GUI Checkpoint dialog box closes. WinRunner captures the information about the fields and stores it in the expected results folder for the test. A **win_check_gui** statement is inserted into the test script.

Checking the Default Properties for All Fields in a Screen

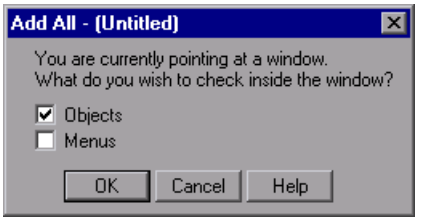
You can add a GUI checkpoint to check all fields in a screen. WinRunner creates a checklist containing the default check (**Date**) for all fields in the screen.

To check the default properties for all the fields in a screen:

1 Choose **Insert** > **GUI Checkpoint** > **For Object/Window**, or click the **GUI Checkpoint for Object/Window** button on the User toolbar.

2 Click on an empty area of the screen or in the title bar of the terminal emulator window.

The Add All dialog box opens.



3 Select **Objects**, **Menus**, or both, to indicate the types of objects to include in the checklist. When you select only **Objects** (the default setting), all objects in the window except for menus are included in the checklist. To include menus in the checklist, select **Menus**.

4 Click **OK** to close the dialog box.

WinRunner captures the information about the fields and stores it in the expected results folder for the test. (This may take several seconds.) The WinRunner window is restored and a **win_check_gui** statement is inserted into the test script.

Screen and Field Property Checks

When you create a GUI checkpoint, you can determine the types of checks to perform on screens and fields in your application.

Screen Checks

For a screen, you can check the following properties:

- **screen_input_num—**Checks the number of unprotected fields in the screen (default check).
- **screen_label**—Checks the label (title) of the screen.
- **screen_prot_num—**Checks the number of protected fields in the screen (default check).

Field Checks

For a field, you can check the following properties:

- **field_attached_text—**Checks the attached text of the field.
- **field_color—**Checks the color of the field.
- **field_content—**Checks the content of the field.
- **field_date—**Checks the date of the field (default check).
- **field_len—**Checks the length of the field, in characters.
- **field_numeric_only—**Checks whether the field is numeric only.
- **field_protected—**Checks whether the field is protected.
- **field_visible—**Checks whether the field is visible.
- **field_x**—Checks the x-coordinate of the top left corner of the field, relative to the screen origin.
- **field_y**—Checks the y-coordinate of the top left corner of the field, relative to the screen origin.

Checking Dates

You can check how your terminal emulator application performs date operations. For example, if your application is used by European and North American customers, you may want to check how your application responds to the different date formats used by these customers.

For terminal emulators, you can:

- choose how WinRunner identifies dates
- choose how WinRunner identifies date fields
- check all or some dates in the current terminal emulator screen
- automatically create date checkpoints whenever the terminal emulator screen changes

Setting How WinRunner Identifies Dates

You can use the **TE_date_set_capture_mode** function to determine how WinRunner captures dates in terminal emulator applications. WinRunner can capture dates either by date field or by position on the screen. This function has the following syntax:

TE_date_set_capture_mode (*mode* **);**

The *mode* is the date capture mode—one of the following:

- 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 their row and column numbers on the screen.

  If you use the default FIELD_METHOD mode, WinRunner can capture only those dates that start at the beginning of a field. To capture dates that do not appear at the beginning of a field (for example, Tuesday 03/31/99), you must use the POSITION_METHOD mode. This mode captures dates according to their row and column numbers on the screen.

Setting How WinRunner Identifies Date Fields

You can use the **TE_date_set_attr** function to set the record configuration mode for a field. This determines whether WinRunner identifies a date field by its index or by its attached text. This function has the following syntax:

TE_date_set_attr (*mode* **);**

The *mode* is the record configuration mode—either INDEX or ATTACHED TEXT.

Checking All Dates in a Terminal Emulator Screen

You can use the **TE_date_check** function to create a date checkpoint for dates in all or part of the current screen of a terminal emulator application. This function has the following syntax:

TE_date_check (*filename* [, *start_column*, *start_row*, *end_column*, *end_row*] **);**

The *filename* is the file containing the expected results of the date checkpoint.

The *start_column/row* is the column/row at which the captured date begins; the *end_column/row* is the column/row at which the captured date ends.

Automatically Creating Date Checkpoints

You can use the **TE_set_auto_date_verify** function to automatically generate a date checkpoint (by inserting a **TE_date_check** statement) whenever the screen changes in a terminal emulator application. This function has the following syntax:

TE_set_auto_date_verify (ON|OFF **);**

If this function is set ON, WinRunner automatically generates a date checkpoint to capture all date information in the current screen.

For additional information on the date-related functions and examples of usage, refer to the *TSL Online Reference*. For additional information on date checkpoints, refer to the "Checking Dates" chapter in the *WinRunner User's Guide*.

6

Checking Text

You can use WinRunner to check the text in the screen of your terminal emulator application, by inserting text checkpoints. Text checkpoints compare on-screen text according to its physical location on the screen. WinRunner can capture the entire screen of the active terminal emulator window, or only the portion of the screen that you specify.

This chapter describes:

- About Checking Text
- Checking Text Automatically
- Checking Text Using Softkeys
- Using Filters when Checking Text
- Reading Text from the Screen
- Searching for Text

About Checking Text

WinRunner provides different methods of checking the text in your terminal emulator application screen. You can:

- capture all or part of the screen contents while recording a test
- instruct WinRunner to automatically capture all or part of the screen contents of the active terminal emulator window

While creating a test, you indicate the text that you want to check. WinRunner inserts a checkpoint into the script, captures the specified text, and stores it in the expected results folder for the test.

When you run the test, WinRunner recaptures the text and compares it to the expected text captured earlier. You can view both the expected and the actual test results. In the case of a mismatch, you can also view any differences between them.

You can also use WinRunner to read text from a selected portion of the screen and store it in a variable. The screen coordinates of the text you indicated are inserted into the test script. You could use this feature, for example, to change the logical flow of a test run according to the text found in the indicated area.

Checking Text in a Terminal Emulator Screen

The **TE_check_text** function statement captures and compares the text in a terminal emulator screen. This function has the following syntax:

TE_check_text (*file_name* [, *start_column*, *start_row*, *end_column*, *end_row*] **);**

The *file_name* parameter is a string expression provided by WinRunner that identifies the captured window.

The *start_column/row* is the column/row at which the captured text begins; the *end_column*/*row* is the column/row at which the captured text ends.

Checking Text Automatically

You can instruct WinRunner to capture the contents of the active terminal emulator window automatically each time a new screen is displayed. The three main options for automatic text checkpoints are:

- check full screen
- check partial screen
- check partial screen using the coordinates specified for a previous checkpoint

---

Note: The automatic text checking options use the **TE_set_auto_verify** function. This function is available only for 3270 and 5250 terminal emulators that support the EHLLAPI protocol.

Checking Full Screens

When automatic full screen text checking is active, all the text in the active window is captured each time a new screen is displayed.

To activate automatic full screen text checking, include the following statement in your test script:

```
TE_set_auto_verify ( ON );
```

To deactivate automatic full screen text checking, include the following statement:

```
TE_set_auto_verify ( OFF );
```

Each time a new screen is displayed in the window, a **TE_check_text** statement, WinRunner automatically inserts a statement similar to the following into the test script.

```
TE_check_text ( "Trm1" );
```

The default name that WinRunner assigns to the first incidence of a full screen text checkpoint in a test script is called **Trm1**. The text is stored as an ASCII file in the expected results folder for the test.

When you run the test, WinRunner compares the text currently displayed on the screen with the expected text captured earlier (the contents of the file **Trm1**, stored in the expected results folder).

In the event of a mismatch, WinRunner captures the actual text and generates a file that shows the discrepancy between the expected and the actual results. Both files are stored in the current verification results folder.

Checking Partial Screens

When automatic partial screen text checking is active, the text in the specified area of the active window is captured each time a new screen is displayed.

To activate automatic partial screen text checking, include a statement in your test script with the following syntax:

```
TE_set_auto_verify ( ON, start_column, start_row, end_column, end_row );
```

The ON parameter activates the automatic check.

The *start_column*/*row* is the column/row at which the captured text starts; the *end_column*/*row* is the column/row at which the text ends.

The example below shows a statement to automatically check the text in columns 22 through 31, rows 10 through 14.

```
TE_set_auto_verify (ON, 22, 10, 31, 14);
```

Each time a new screen is displayed in the window, a **TE_check_text** statement similar to the following is automatically inserted into the test script:

```
TE_check_text ("Prt1", 22, 10, 31, 14);
```

The default file name **Prt1** indicates the first incidence of captured partial text in any test script.

To deactivate automatic partial text checking, include the following statement:

```
TE_set_auto_verify (OFF);
```

Checking Partial Screens Using Previous Coordinates

When you choose the first/last partial text option, the coordinates for the automatic partial screen text check are taken from a previous **TE_check_text** statement in the test run.

To activate automatic first/last partial screen text checking, include a statement in your test script with the following syntax:

TE_set_auto_verify (ON, *FIRST|LAST* **);**

The ON parameter activates the automatic check.

If you specify FIRST, the coordinates for the automatic partial screen text check are taken from the first **TE_check_text** statement in the test run.

If you specify LAST, the coordinates are taken from the **TE_check_text** statement immediately before the current statement in the test run. The coordinates are updated during the test run each time a **TE_check_text** statement is executed.

If there is no **TE_check_text** statement in the test script prior to the **set_auto_verify** statement, then the entire screen is captured.

To deactivate automatic first/last partial screen text checking, include the following statement in your test script:

```
TE_set_auto_verify (OFF);
```

Checking Text Using Softkeys

While recording, you can use softkeys to check text. For a list of the softkeys for different terminal emulators, see "WinRunner Terminal Emulator Softkeys" on page 39.

Using softkeys, you can check the entire contents of the terminal emulator screen or a specific portion of the screen. All captured text is stored as ASCII text.

Checking a Full Screen Using a Softkey

You can use a softkey to capture the entire contents of the active terminal emulator screen.

To capture the entire contents of the screen:

1. While recording, make sure that the terminal emulator window you want to check is active.
2. Press the CHECK TEXT softkey. A **TE_check_text** statement is generated in your test script.

   The entire contents of the active terminal emulator screen are captured (even if not all the text is visible in the window). A **TE_check_text** statement similar to the following is inserted into the test script:

   ```
   TE_check_text ("Trm1");
   ```

   The default name that WinRunner assigns to the first incidence of a full screen text checkpoint in a test script is **Trm1**. The text is stored as an ASCII file in the expected results folder for the test.

   When you run the test, WinRunner compares the text currently displayed on the screen with the expected text captured earlier (the contents of the file **Trm1**, stored in the expected results folder).

   In the event of a mismatch, WinRunner captures the actual text and generates a file that shows the discrepancy between the expected and the actual results. Both files are stored in the current verification results folder.

Checking a Partial Screen Using a Softkey

You can use a softkey to create a partial text checkpoint when you want to capture only part of the text on the screen.

To capture text in an area of the screen:

1. Press the CHECK PARTIAL TEXT softkey. WinRunner is minimized to an icon and a dialog box displays instructions for capturing the text.
2. Click and hold down the left mouse button and drag the mouse to create a rectangle that encloses the area to be filtered.
3. Right-click inside the defined area. WinRunner is restored and a **TE_check_text** statement similar to the following is inserted into the test script:

```
TE_check_text ("Prt1", 51, 13,60, 13);
```

The above example shows the statement recorded when the text in line 13, columns 51 through 60, is captured.

The default file name **Prt1** indicates the first incidence of captured partial text in any test script.

For more information on the **TE_check_text** function, refer to the *TSL Online Reference*.

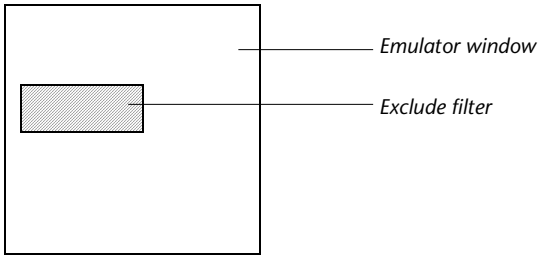
Using Filters when Checking Text

WinRunner lets you use filters to include or exclude regions of a terminal emulator window when checking text. In cases where you do not want to check an entire window, you can define parts of the window to be filtered during the comparison. You can use two types of filters: *exclude* and *include*.

Note: You can also use filters to check dates in specific areas of the screen. For more information on adding date checkpoints, see "Checking Dates" on page 57.

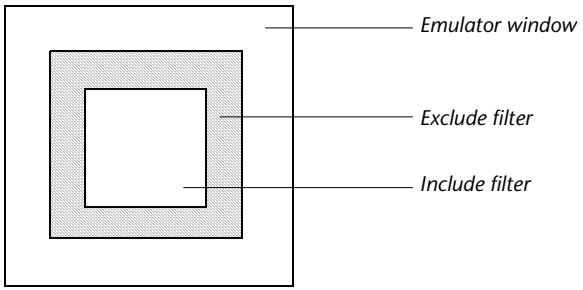
Exclude and Include Filters

An exclude filter defines the area to be ignored during the comparison. For example, you can create an exclude filter on a region of a window containing the current date and time.



Emulator window with exclude filter

An include filter is used in combination with an exclude filter. In the diagram below, the white areas are included in the comparison and the shaded area is excluded. This is achieved by defining an exclude filter and then defining a smaller include filter on top of it. The result is a "ring" that is excluded from comparison.



Emulator window with exclude filter and include filter

Note that when you combine exclude and include filters, the order in which the filters are activated in the test script determines the actual area of interest. For example, if an exclude filter that fully or partially overlaps an include filter is activated after the include filter, the overlapped region is excluded from the area of interest.

Creating Filters

You can use the EXCLUDE FILTER and INCLUDE FILTER softkeys to create a filter while recording. For a list of the softkeys for different terminal emulators, see "WinRunner Terminal Emulator Softkeys" on page 39.

If you know the exact coordinates, you can also insert **TE_set_filter** statements manually into your test script.

To create a filter while recording:

1. While recording, press the appropriate softkey (FILTER EXCLUDE or FILTER INCLUDE). WinRunner is minimized and a dialog box displays instructions for defining the filter area.
2. Click and hold down the left mouse button and drag the mouse to create a rectangle that encloses the area to be filtered.
3. Right-click to save the filter. WinRunner is restored. The filter is added to the **db** folder for the test and a **TE_set_filter** statement is inserted into your test script.

   The following example shows what WinRunner records when an exclude filter is defined for row 23, columns 1 through 30, of all the screens in the terminal emulator application.

```
TE_set_filter ("Filter0",1,23, 30, 23, EXCLUDE, "ALL_SCREENS");
```

   You can modify the statement and specify the name of a specific screen.

   When a **TE_set_filter** statement is executed during a test run, the filter is activated. For more information on the **TE_set_filter** function, refer to the *TSL Online Reference*.

Note: You can create up to a maximum of 256 filters using **TE_set_filter**.

Deactivating and Deleting Filters

When you deactivate an existing filter, it remains in the **db** folder for the test but is inactive for the test. To deactivate a filter, include a statement in your test script with the following syntax:

TE_reset_filter (*filter_name* **);**

If you do not know the filter name, you can define the filter to be deactivated using its coordinates and type, by including a statement with the following syntax:

TE_reset_filter (*start_column*, *start_row*, *end_column*, *end_row*, *EXCLUDE* | *INCLUDE*, *screen_name* **);**

To deactivate all active filters, include a statement in your test script with the following syntax:

TE_reset_all_filters( );

To delete a filter from the test database, include a statement with the following syntax:

TE_delete_filter (*filter_name* **);**

Creating and Activating Filters Separately

In some cases, you may wish to create a filter and store it in the test's **db** folder for later use. You can do this by using the **TE_create_filter** function to create a filter, and then activate it by executing a **TE_set_filter** statement containing only the name of the filter.

To create a filter, include a statement in your test script with the following syntax:

TE_create_filter (*filter_name*, *start_column*, *start_row*, *end_column*, *end_row*, *EXCLUDE* | *INCLUDE*, *screen_name* **);**

The *filter_name* can contain a maximum of 16 characters.

To activate a filter, include a statement in the script with the following syntax:

TE_set_filter (*filter_name* **);**

The *filter_name* must be the name of an existing filter for the current test.

Reading Text from the Screen

Using the **TE_get_text** function, you can instruct WinRunner to read the text in a specified area of the screen and store it in a variable. You can use a softkey while recording to define the area of the screen to be read. You can also enter **TE_get_text** statements manually.

To read text from the screen:

1. Make sure that you are in recording mode and that the terminal emulator window from which you want to capture text is in focus.
2. Press the GET TEXT softkey. For a list of the softkeys for different terminal emulators, see "WinRunner Terminal Emulator Softkeys" on page 39.

   WinRunner is minimized and a dialog box displays instructions for capturing the string.
3. Click and hold down the left mouse button and drag the mouse to create a the rectangle that encloses the desired area.
4. Right-click to capture the text. A **TE_get_text** statement is inserted into the test script. This statement has the following syntax:

   TE_get_text (x_1, y_1, x_2, y_2 **);**

   For more information on the **TE_get_text** function, refer to the *TSL Online Reference*.

   Each new line of text that is captured is preceded by the characters **\n** in the variable. The following example shows how two lines of text appear in the variable *t*:

```
t = "Fill in your User ID and press Enter \n(Your password will not appear
    when you type it)"
```

Searching for Text

You can search for text in a terminal emulator screen, using the **TE_find_text** function. This function looks for a specified text string and returns its location on the screen as an x-coordinate and a y-coordinate. Using an optional parameter, you can restrict the search to a rectangular area of the screen that you define using pairs of x-, y-coordinates.

The **TE_find_text** function has the following syntax:

TE_find_text (, *string*, *out_x_location*, *out_y_location* [x_1, y_1, x_2, y_2] **);**

The *string* parameter is the text that you want to locate.

The *out_x_location* parameter is the output variable that stores the x-coordinate of the test string. The *out_y_location* parameter is the output variable that stores the y-coordinate of the text string.

The optional x_1, y_1, x_2, y_2 parameters define a rectangle that specifies the limits of the search area.

For more information on the **TE_find_text** function, refer to the *TSL Online Reference*.

7

Testing VT100 and Text Applications

You can use WinRunner to test applications on terminal emulators that do not support the EHLLAPI protocol, such as VT100, VAX, UNIX, HP, and also to test text applications.

This chapter describes:

- About Testing VT100 and Text Applications
- Creating Test Scripts
- Synchronizing Tests
- Checking Text for VT100 and Text Applications
- TSL Functions

About Testing VT100 and Text Applications

When working with VT100 terminal emulators, the *text* recording method is used. The text method is similar to the *position* recording method, which can be used in 3270 and 5250 terminal emulators that support the EHLLAPI protocol. In both methods, WinRunner records keyboard and mouse input only. The operations on objects in your application are recorded as **win_type, obj_type**, **win_mouse_click**, and **win_mouse_drag** statements.

However, unlike the position method, the text method does not insert synchronization statements into your test script automatically. You need to insert synchronization statements using softkeys or by entering them into the script manually.

Creating Test Scripts

When using VT100 terminal emulator applications, WinRunner records keyboard and mouse input only. The operations on objects in your application are recorded as **win_type**, **obj_type**, **win_mouse_click**, and **win_mouse_drag** statements.

The following is a sample of a WinRunner test script recorded on a VT100 terminal emulator application. The comment lines (starting with #) describe the statements.

```
# Activate the terminal emulator window
win_activate ("RUMBA - DEMO");

# Direct input to the screen
set_window ("RUMBA - DEMO", 1);

# Type in the user id “Michael”
obj_type ("AfxWnd40","Michael");

# Press the Enter key
obj_type ("AfxWnd40","<kReturn>");

# Wait for a string to appear on the next screen.
TE_wait_string(" MENU ", 1, 1, 53, 1, 60);

# Type a menu option.
obj_type ("AfxWnd40","90");
```

In the above example, the user clicks on the terminal emulator window to activate it. WinRunner records this action to ensure that the input is directed to the correct window. Then the user types the user name **Michael** in the appropriate field and presses the ENTER key.

A wait statement is added to ensure that WinRunner waits for the string to appear on the next screen. The user types an option.

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

Synchronizing Tests

When using VT100 terminal emulator applications, you can insert synchronization points into your test script to pace the test run with the system.

Using the **TE_wait_string** function, you can instruct WinRunner to wait for a specific string to appear on the screen before continuing the test run. You can specify an area of the screen, or WinRunner can search the entire screen for the string. For more information, see "Waiting for a Specific String" on page 44.

The **TE_set_timeout** function determines the maximum amount of time (in seconds) that WinRunner waits for a response from the host before continuing the test run. The **TE_get_timeout** function returns the maximum amount of time (in seconds) that WinRunner waits for a response from the host before continuing the test run. For more information, see "Setting and Retrieving the Timeout" on page 47.

Setting Synchronization Keys

Using the **TE_define_sync_keys** function, you can set keys that enable automatic synchronization in **type**, **win_type**, and **obj_type** functions.

When WinRunner executes a **type**, **win_type**, or **obj_type** statement that includes a synchronization key, WinRunner waits for a specified string to either appear on the screen or disappear from the screen. This function has the following syntax:

TE_define_sync_keys (*keys*, *string*, *mode* [, x_1, y_1, x_2, y_2] **);**

The *keys* parameter contains a list of the keys that enable automatic synchronization. Represent each key by an identifier consisting of the letter k followed by the key name, bracketed with greater than/less than signs (< >). Use a comma (,) as the delimiter between key identifiers. For example, "<kReturn>, <kCtrl>, ...". For key sequences in which more than one key is pressed simultaneously, use a hyphen (-) to join them, for example, "<kCtrl-kC>, ...".

The *string* parameter defines the string that WinRunner waits for before continuing with the test run.

The *mode* parameter can be one of the following:

- SYNC_WHILE—Waits until the string disappears
- SYNC_UNTIL—Waits until the string appears
- SYNC_DEFAULT—Waits the default synchronization time

The parameters x_1, y_1, x_2, y_2 define a rectangle in which to search for the string (optional). If these parameters are not defined, WinRunner searches the entire screen.

Checking Text for VT100 and Text Applications

While creating a test for VT100 and text applications, you indicate the text that you want to check from a selected portion of the screen by defining the screen coordinates of the text.

Using the **TE_get_text** function, you can instruct WinRunner to read the text in a specified area of the screen and store it in a variable. For more information, see "Reading Text from the Screen" on page 69.

You can create filters to include or exclude regions of a terminal emulator window when checking text. In cases where you do not want to check an entire window, you can define parts of the window that will be filtered during the comparison. For more information, see "Using Filters when Checking Text" on page 65.

The **TE_check_text** function captures and compares the text in a terminal emulator window. For more information, see "Checking Text in a Terminal Emulator Screen" on page 60.

You can search for text in a terminal emulator screen using the **TE_find_text** function. This function looks for a specified text string and returns its location on the screen as an x-coordinate and a y-coordinate. Using an optional parameter, you can restrict the search to a rectangular area of the screen that you define using pairs of x- and y-coordinates.

For more information on **TE_find_text**, see "Searching for Text" on page 70.

TSL Functions

You can use the following TSL functions when testing your VT100 terminal emulator application. For more information on TSL functions, refer to the *TSL Online Reference*.

Synchronization Functions

You can insert synchronization points into your test script to pace the test run with the system.

➤ The **TE_define_sync_keys** function sets keys that enable automatic synchronization in **win_type** and **obj_type** commands. It has the following syntax:

TE_define_sync_keys (*keys*, *string*, *mode* [, x_1, y_1, x_2, y_2] **);**

➤ The **TE_set_timeout** function sets the maximum amount of time (in seconds) that WinRunner waits for a response from the host. It has the following syntax:

TE_set_timeout (*timeout* **)**;

➤ The **TE_get_timeout** function returns the currently defined timeout (see above). It has the following syntax:

TE_get_timeout ( );

➤ The **TE_wait_string** function waits for a string to appear on screen. It has the following syntax:

TE_wait_string (*string* [, *start_column, start_row, end_column, end_row* [, *timeout*]] **)**;

Date Operation Functions

You can set how WinRunner identifies dates and date fields in your terminal emulator application, and how it creates date checkpoints.

➤ The **TE_date_set_capture_mode** function determines how WinRunner captures dates in terminal emulator applications. It has the following syntax:

TE_date_set_capture_mode (*mode* **);**

➤ The **TE_date_set_attr** function sets the record configuration mode for a field. It has the following syntax:

TE_date_set_attr (*mode* **);**

➤ The **TE_date_check** function creates a date checkpoint for dates in all or part of the current screen of a terminal emulator application. It has the following syntax:

TE_date_check (*filename* [, *start_column*, *start_row*, *end_column*, *end_row*] **);**

➤ The **TE_set_auto_date_verify** function automatically generates a date checkpoint whenever the screen changes in a terminal emulator application. It has the following syntax:

TE_set_auto_date_verify (ON|OFF **);**

For additional information about the date-related functions, see "Checking Dates" on page 57. For examples of usage, refer to the *TSL Online Reference*.

Text Functions

You can check the text in the screen of your terminal emulator application.

➤ The **TE_check_text** function captures and compares the text in a terminal emulator window. It has the following syntax:

TE_check_text (*file_name* [, *start_column*, *start_row*, *end_column*, *end_row*] **);**

➤ The **TE_find_text** function returns the location of a specified string. It has the following syntax:

TE_find_text (*string, out_x_location, out_y_location* [, x_1, y_1, x_2, y_2] **);**

➤ The **TE_get_text** function reads text from the screen and stores it in a string. It has the following syntax:

TE_get_text (x_1, y_1, x_2, y_2 **);**

Filter Functions

You can create filters to check text and dates on your terminal emulator screen.

➤ The **TE_create_filter** function creates a filter in the test database. It has the following syntax:

TE_create_filter (*filter_name, start_column, start_row, end_column, end_row,* EXCLUDE | INCLUDE*, screen_name* **);**

➤ The **TE_delete_filter** deletes a specified filter from the test database. It has the following syntax:

TE_delete_filter (*filter_name* **)**;

➤ The **TE_get_active_filter** function returns the coordinates of a specified active filter. It has the following syntax:

TE_get_active_filter (*filter_num* [, *out_start_column, out_start_row, out_end_column, out_end_row, screen_name*] **);**

➤ The **TE_get_auto_reset_filter** function indicates whether or not filters are automatically deactivated at the end of a test run. It has the following syntax:

TE_get_auto_reset_filters ( );

➤ The **TE_get_filter** function returns the properties of a specified filter. It has the following syntax:

TE_get_filter (*filter_name* [, *out_start_column, out_start_row, out_end_column, out_end_row, out_type, out_active, screen_name*] **);**

➤ The **TE_reset_all_filters** function deactivates all filters in a test. It has the following syntax:

TE_reset_all_filters ( );

➤ The **TE_reset_filter** function deactivates a specified filter. It has the following syntax:

TE_reset_filter (*filter* **);**

➤ The **TE_set_filter** function creates and activates a filter. It has the following syntax:

TE_set_filter (*filter_name* [, *start_column, start_row, end_column, end_row,* EXCLUDE | INCLUDE, *screen_name*] **);**

➤ The **TE_set_auto_reset_filters** function deactivates the automatic reset of filters when a test run is completed. It has the following syntax:

TE_set_auto_reset_filters (ON|OFF **);**

➤ The **TE_set_filter_mode** function specifies whether to assign filters to all screens or to the current screen. It has the following syntax:

TE_set_filter_mode (*mode* **);**

8

Learning the Application Using BMS Files

The Learn BMS Files feature can teach WinRunner your 3270 mainframe application by inserting information about screens and fields directly into a GUI map file. This chapter describes:

- About Learning the Application Using BMS Files
- Learning the Application the First Time
- Relearning the Application

About Learning the Application Using BMS Files

If you have access to the BMS file of your 3270 mainframe application, you can use the Learn BMS Files feature. This feature enables WinRunner to learn your application directly from a BMS file containing descriptions of the screens and fields in your application.

When you use Learn BMS Files, WinRunner learns these descriptions and inserts them into a GUI map file. You can change the names or descriptions as desired, as with any other GUI map file. You use the TSL function **TE_bms2gui** to learn the BMS file.

The RELEARN option lets you update the GUI map file you created earlier as your application changes during the development cycle. An interactive user interface guides you through the process. It helps you retain desired modifications to the descriptions in the GUI map file while changing others as needed.

It is recommended that you read Chapter 3, "Testing Terminal Emulator Applications" in this guide, as well as the "Understanding the GUI Map" section in the *WinRunner User's Guide,* before you use the Learn BMS Files feature.

Learning the Application the First Time

You use the **TE_bms2gui** function to learn (and to relearn) your BMS file. This function has the following syntax:

TE_bms2gui (*bms_filename*, *gui_filename*, LEARN|RELEARN **);**

The *bms_filename* parameter is the full path of the BMS file of your application.

The *gui_filename* parameter is the full path of the GUI map file in which WinRunner inserts the descriptions of the objects into your application. If no parameter is specified, the temporary GUI map file is used.

The LEARN|RELEARN parameter determines how WinRunner handles the BMS file:

- Use the LEARN option the first time you learn a BMS file. Do not perform LEARN twice for the same GUI map file.
- Use RELEARN when you have made changes to your application and updated the BMS file. When RELEARN is specified, WinRunner compares the descriptions in the current BMS file with those in the specified GUI map file. It notifies you of any inconsistencies and enables you to make changes as required.

To learn the BMS files, run the **TE_bms2gui** function in a WinRunner script.

In the following example, **TE_bms2gui** is used to teach WinRunner object descriptions from a BMS file called **Mail_app.txt**, and place them into a GUI map file called **Mail_1.gui**:

```
TE_bms2gui ("Mail_app.txt", "Mail_1.gui", LEARN);
```

You can edit names or descriptions in the GUI map file created by **TE_bms2gui** and make any other required changes, using the GUI Map Editor. For more information on the GUI Map Editor, refer to the *WinRunner User's Guide*.

Relearning the Application

You use the RELEARN option each time you want to update the GUI map file to reflect changes in your application. RELEARN enables you to add new screens and fields to the GUI map file while maintaining or changing the names and descriptions that appear in the existing GUI map file, as required.

To relearn a BMS file, you execute the **TE_bms2gui** function using RELEARN as the *learn_mode* parameter. For example, to relearn a BMS file called **Mail_app.txt** into an existing GUI map file called **Mail_1.gui**, run the following statement:

```
TE_bms2gui ("Mail_app.txt", "Mail_1.gui", RELEARN);
```

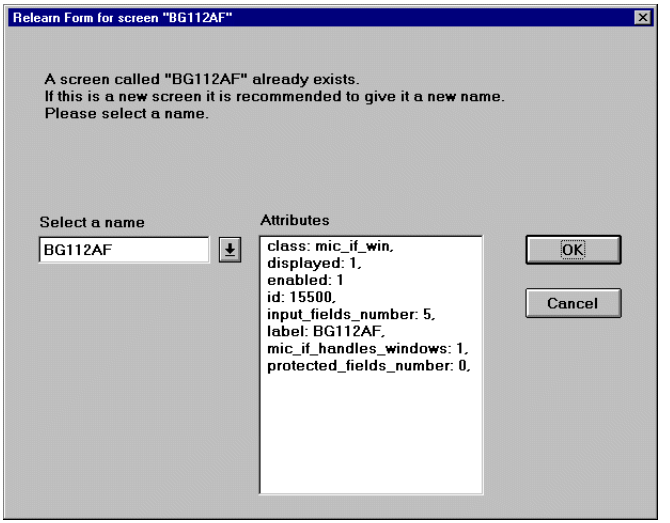
As WinRunner converts the GUI map file from the BMS file, it looks for discrepancies between the existing GUI map file and the current file on which RELEARN is performed. Each time it finds a mismatch, a dialog box prompts you to specify how to proceed.

In most cases, accepting the default option ensures that the intentional changes made to your application are reflected accurately in the GUI map file. However, WinRunner also gives you the option of changing the name of the relevant screen or field.

The following examples describe the different RELEARN dialog boxes that may be displayed during the RELEARN process and the options they provide.

Object Exists in the GUI Map File with Different Properties

In this example, WinRunner found a screen in the BMS file with the same name as a screen in the existing GUI map file, but with different properties.



The current name of the screen is displayed in the list on the left. The list on the right shows all the properties of the selected object, according to the new BMS file. By default, WinRunner updates the GUI map to include the new properties.

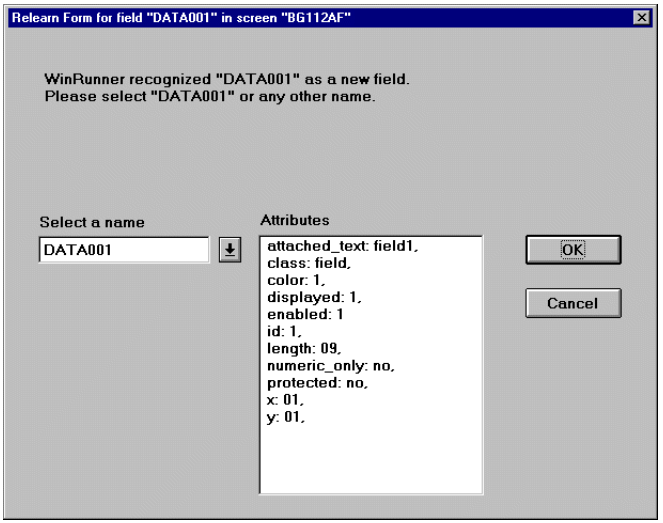
To use a different name, select it from the list or type in another name.

To save your changes, click **OK**. The RELEARN dialog box closes and a confirmation message is displayed. Click **OK**.

To continue the RELEARN operation without making changes in the GUI map for the specified screen object, click **Cancel**.

Object is not in the Original GUI Map File

In this example, WinRunner found a field that it recognizes as a new one. No other field with the same name or properties exists in the GUI map file.



The name of the field is displayed in the list on the left. The list on the right shows all the properties of the selected field, according to the new BMS file.

By default, WinRunner adds the object to the GUI map file with the name specified.

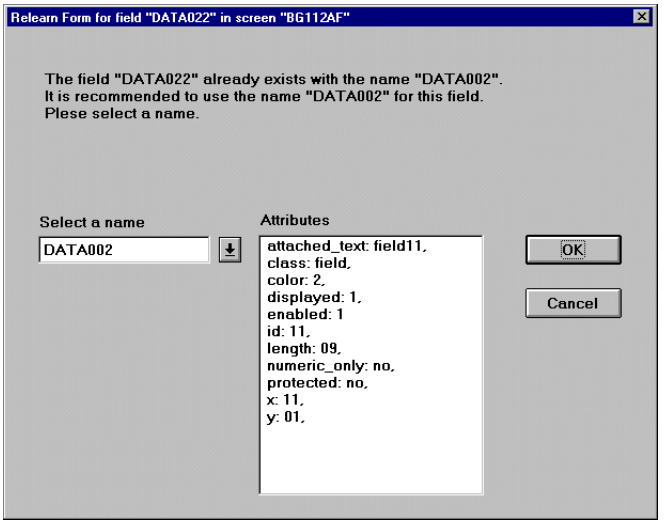
To specify a new name for the object, type it in or select the name of another screen from the list.

To save your changes, click **OK**. The RELEARN dialog box closes and a confirmation message is displayed. Click **OK**.

To continue the RELEARN operation without making changes to the GUI map file, click **Cancel**.

Object has a Different Name in the GUI Map File

In this example, WinRunner found a field with the same properties as an existing field, but with a different name.



By default, WinRunner retains the original name for the field as it is displayed in the GUI map. This ensures that you can rerun existing tests containing the original name for the field without changing them.

To use the name in the new BMS file or to select a new name, select it from the list or type it in.

Click **OK** to save your changes, or **Cancel** to continue the RELEARN process without making changes to the GUI map.

Index

T

V

W

Mercury Interactive Corporation
1325 Borregas Avenue
Sunnyvale, CA 94089 USA

Main Telephone: (408) 822-5200
Sales & Information: (800) TEST-911, (866) TOPAZ-4U
Customer Support: (877) TEST-HLP
Fax: (408) 822-5300

Home Page: www.mercuryinteractive.com
Customer Support: support.mercuryinteractive.com