*QA*Run Character-Based Testing

Getting Started Guide

Release 4.7

COMPUWARE®

BETA RELEASE

Please direct questions about *QA*Run
or comments on this document to:

*QA*Run **Technical Support**
Compuware Corporation
31440 Northwestern Highway
Farmington Hills, MI 48334-2564

1-800-538-7822

Outside the USA and Canada, please contact
your local Compuware office or agent.

U.S. GOVERNMENT RIGHTS

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in Compuware Corporation license agreement and as provided in DFARS 227.7202-1(a) and 227.7202-3(a) (1995), DFARS 252.227-7013(c)(1)(ii)(OCT 1988), FAR 12.212(a) (1995), FAR 52.227-19, or FAR 52.227-14 (ALT III), as applicable. Compuware Corporation.

THIS PRODUCT CONTAINS CONFIDENTIAL INFORMATION AND TRADE SECRETS OF COMPUWARE CORPORATION. USE, DISCLOSURE, OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF COMPUWARE CORPORATION. ACCESS IS LIMITED TO AUTHORIZED USERS. USE OF THIS PRODUCT IS SUBJECT TO THE TERMS AND CONDITIONS OF THE USER'S LICENSE AGREEMENT WITH COMPUWARE CORPORATION.

Doc. CWQUGST4F
April 04, 2000

Table of Contents

Introduction

This guide is designed for new *QA*Run users who intend to test character-based applications running through a Windows 95 or Windows NT terminal emulator. This guide describes the basic testing principles involved in using *QA*Run to test a character-based application. As you complete the exercises contained in this guide, you will become familiar with *QA*Run testing concepts through a comprehensive tutorial that is designed to assist you in building a complete test system. The exercises presented in this manual are designed to take you from the most basic testing principles through the more advanced testing concepts using *QA*Run.

This manual is divided into the following chapters:

- **"Chapter 1. Introducing Testbed":** This chapter provides an overview of the sample target application (Testbed) and introduces you to Testbed's basic menus and toolbar buttons.
- **"Chapter 2. Building a Synchronized Driver Script":** This chapter provides an introduction to the concepts of driver scripts and synchronization techniques. It also provides step-by-step procedures designed to help you create the driver script that will be used in future exercises.
- **"Chapter 3. Building Test Scripts":** This chapter provides an overview of *QA*Run's more powerful testing capabilities. As you complete the step-by-step exercises contained in this chapter, you will learn how to use checks, testdata files, and user-defined dialog boxes to build robust test scripts.
- **"Chapter 4. Using Driver and Test Scripts Together":** This chapter provides step-by-step instructions that assist you in combining the driver and test scripts that you created in previous exercises. This chapter also discusses the final test element — how to run your scripts against a new version of the target application and interpret the test results.

The concepts required to test graphic user interface (GUI) applications are covered in the *QARun GUI Testing Getting Started Guide*. You should refer to that document for introductory information related to testing GUI applications.

Who Should Read This Guide

The *QARun Character-Based Testing Getting Started Guide* is intended for new *QA*Run users who will be using *QA*Run to test character-based applications. This guide provides a comprehensive *QA*Run tutorial.

We designed and organized the information in this guide to help novice users gain a basic understanding of the test-building process using *QA*Run. This guide does not contain extensive reference or theoretical information. You can find that information in the online help facility, in the *QARun User's Guide*, and in the *QARun Language Reference Manual*.

We assume that you have some familiarity with basic Microsoft Windows navigation. If this isn't the case, familiarize yourself with the documentation accompanying your copy of Microsoft Windows before reading this guide.

Related Publications

In addition to the *QARun Character-Based Testing Getting Started Guide*, the *QA*Run documentation set includes the following other publications as well as a complete online help facility.

- *QACenter Installation and Configuration Guide* provides licensing instructions, step-by-step installation procedures, and database selection information.
- *QARun GUI Testing Getting Started Guide* provides a "hands on" tutorial designed to take you from the most basic principles of testing GUI testing principles through the more advanced testing concepts using *QA*Run.
- *QARun User's Guide* provides a complete reference to using *QA*Run. It provides a basic product and component overview and explains how to configure the system, create scripts, define test conditions (checks), and view the results of a test run.
- *QARun Language Reference Manual* provides a reference to the commands available for use in your *QA*Run scripts. It contains detailed information specific to command syntax, variants, operation, and script examples. It is intended for experienced *QA*Run users who wish to exploit the scripting language to develop robust, sophisticated test procedures.
- *QA*Run Online Help provides help information. If you press the **F1** key while using the software, you can obtain dialog-sensitive online help. If you encounter any problems that cannot be resolved, contact *QA*Run Technical Support.
- The online Bookshelf provides access to the complete *QA*Run documentation set in Adobe Acrobat PDF format. The Bookshelf is installed automatically with the *QA*Run program files. You can access the Bookshelf by selecting **Bookshelf** from the *QA*Run program group. The book opens in Adobe Acrobat Reader.

Viewing and Printing Online Books

*QA*Run's online books are provided in PDF format, so you need Adobe Acrobat Reader 3.0 or above to view them. To install the Adobe Acrobat Reader, click **Install Adobe Acrobat Reader** on the *QA*Center CD, or go to Adobe's Web site at www.adobe.com.

You can access the online books from the documentation bookshelf. For example, if you have *QA*Run installed and you wish to access the *QA*Run *User's Guide,* click the taskbar's **Start** button and choose **Programs>Compuware>QARun >QARun Books> QARun Bookshelf**. Select the *QA*Run *User's Guide* from the list.

Because PDF is based on PostScript, a PostScript printer is the most reliable way to print the online books. In most cases, you can also print PDF files to PCL printers. If you cannot print the PDF files to your printer, refer to Adobe's web site at www.adobe.com for troubleshooting information.

World Wide Web Information

To access Compuware Corporation's site on the World Wide Web, point your browser at http://www.compuware.com. The Compuware site provides a variety of product and support information.

FrontLine Support Web Site: You can access online technical support for Compuware products via our FrontLine support Web site at http://frontline.compuware.com. FrontLine provides fast access to critical information about your *QA*Center product. You can read or download documentation, frequently asked questions, and product fixes, or e-mail your questions or comments. The first time you access FrontLine, you are required to register and obtain a password.

*QA*Run Terminology

The following terms are used throughout the *QARun Character-Based Testing Getting Started Guide*.

Target Application: The system under test.

Scripts: Step-by-step instructions that specify how the target application should be tested.

Test Site: A point in the target application where testing begins.

Driver Script: A *QA*Run script that drives the target application to a test site. A driver performs no testing.

Test Script: A *QA*Run script that is called by the driver script when a test site is reached. A test script performs some actions and then executes a “check” that compares the target’s response to the actions with the expected response.

Check: Definition of the correct response of the target application to the given input. A check may be called by single or multiple test scripts.

Getting Help

At Compuware, we strive to make our products and documentation the best in the industry. Feedback from our customers helps us maintain our quality standards. If you need support services, please obtain the following information before calling Compuware’s 24-hour product support hotline:

- The name, release (version), and build number of the *QA*Run product. This information is displayed when you select the About command from your product’s Help menu. The name and release are also on the covers of the product documentation.
- Installation information, including installed options, whether the product uses local or network databases, whether it is installed in the default directories, whether it is a standalone or network installation, and whether it is a client or server installation.
- Environment information, such as the operating system and release on which the product is installed, memory, hardware/network specifications, and the names and releases of other applications that were running.
- The location of the problem in the *QA*Run product software and the actions taken before the problem occurred.
- The exact product error message, if any.
- The exact application, licensing, or operating system error messages, if any.
- Your Compuware client, office, or site number, if available.

***QA*Run Technical Support**
Compuware Corporation
31440 Northwestern Highway
Farmington Hills, MI 48334-2564
1-800-538-7822

Chapter 1. Introducing Testbed

The subsequent sections of this guide describe how to use *QA*Run to perform practical exercises that are common testing requirements. To complete these exercises, you must first create a testing scenario that includes an application-under-test (the target application), a driver script, and various test scripts.

The *QA*Run installation process loads a demonstration application called Testbed, which is used as the target application for the examples that follow in this manual. Testbed is designed to look like a “mainframe” application which is accessed via a Windows-based terminal emulator. However, Testbed is not an actual mainframe application. Its purpose is simply to demonstrate the features of *QA*Run. The Testbed application includes screens that set up communications parameters, userIDs and passwords, and buttons to connect and disconnect from the mainframe session. There is, of course, no real communication to a remote computer — Testbed just simulates the situation.

Testbed is delivered in two versions. Testbed Version 1 (the default for most exercises) is used to develop a *QA*Run test system. This system is then used to test Testbed Version 2 and to reveal any differences between the two versions of the application.

This chapter is designed to acquaint you with Testbed’s primary functions and operations. After completing this chapter, you will be ready to begin the exercises described in “Exercise 1 — Creating a Driver Script”.

Starting Testbed

Use the following procedure to start Testbed:

1. Start the target application, Testbed:

   Although *QA*Run can execute an application from anywhere in the system, the best way to launch the target is to use the **Start** button on the taskbar.

   - Click the **Start** button on the taskbar and choose **Run** from the menu. At the Run dialog box, enter Testbed's directory location. The default path is:

     ```
     C:\Program Files\Compuware\QARun\Demos\Testbed.exe
     ```

   - Click **OK**.

2. Click **OK**. Testbed appears as shown in Figure 1-1.

The Testbed Main Window

Testbed's main window is shown in Figure 1-1. The main window contains a menu bar, a toolbar, and a status bar. In later exercises, when you use *QA*Run's Identify utility to interrogate this window, the window will be recognized as MainWindow.

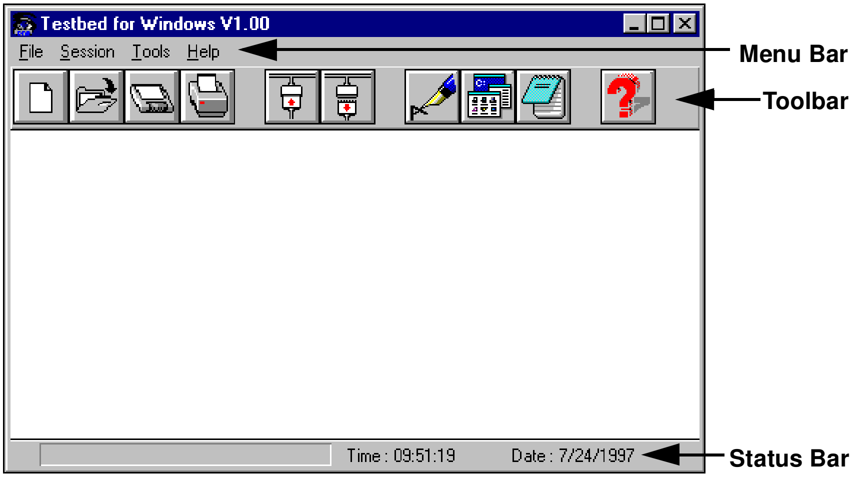


Figure 1-1. Testbed's Main Window

Menu Options

The following menus are available from Testbed's menu bar:

File menu The File menu commands are used to create a new session, open an existing session, save the current session, print the current session, and exit Testbed.

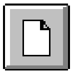
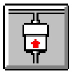
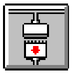
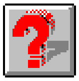
Session menu The Session menu commands are used to enter user information, connect or disconnect the current session from the mainframe, and configure the current session.

Tools menu The Tools menu commands are used to transfer a file from the current session and activate WordPad or Notepad.

Help menu The Help menu commands are used to activate Testbed's online help and to review information about Testbed.

Toolbar Options

The Testbed toolbar contains many buttons. The toolbar buttons that are necessary to complete the exercises in this guide are actually functional. If the toolbar buttons are not required to complete the exercises, they will be non-functional. The following table provides a brief overview of Testbed's toolbar buttons that you will use as you complete the exercises in this guide:

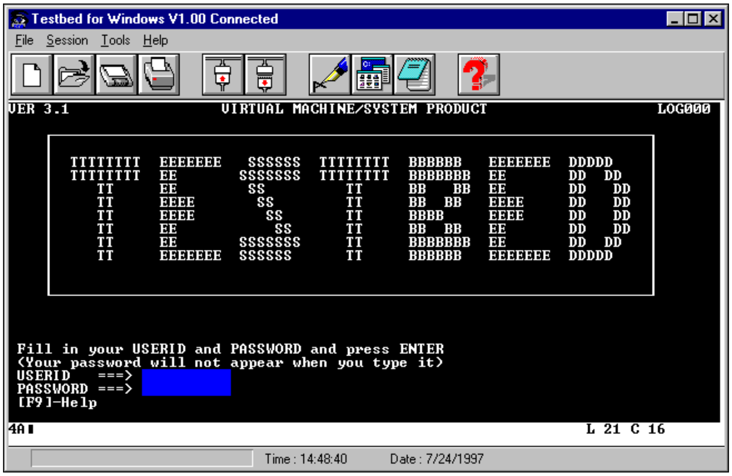
Button	Function
	**New Session** button. This button starts a new Testbed session.
	**Connect Session** button. This button connects the current session to the mainframe.
	**Disconnect Session** button. This button disconnects the current mainframe session.
	**Help Contents** button. This button activates the contents page of the Testbed help system.

Using Testbed

Before you begin testing, you need to learn the basic operations of the target application, Testbed. Make sure that Testbed Version 1 is running. If it isn't, refer to "Starting Testbed" on page 1-2.

Connecting to Sessions and Logging On

1. When Testbed's main window is open, click the **Connect** button. The VIRTUAL MACHINE/SYSTEM PRODUCT logon screen displays:



```
VER 3.1                 VIRTUAL MACHINE/SYSTEM PRODUCT                  LOG000

      TTTTTTTT  EEEEEEE   SSSSSS  TTTTTTTT  BBBBBB   EEEEEEE  DDDDD
      TTTTTTTT  EE       SSSSSSS  TTTTTTTT  BBBBBBB  EE       DD  DD
         TT     EE       SS          TT     BB   BB  EE       DD   DD
         TT     EEEE      SS         TT     BB  BB   EEEE     DD   DD
         TT     EEEE       SS        TT     BBBB     EEEE     DD   DD
         TT     EE          SS       TT     BB  BB   EE       DD   DD
         TT     EE       SSSSSSS     TT     BBBBBBB  EE       DD  DD
         TT     EEEEEEE  SSSSSS      TT     BBBBBB   EEEEEEE  DDDDD

 Fill in your USERID and PASSWORD and press ENTER
 (Your password will not appear when you type it)
 USERID   ===>
 PASSWORD ===>
 [F9]-Help
4A                                                         L 21  C 16
```

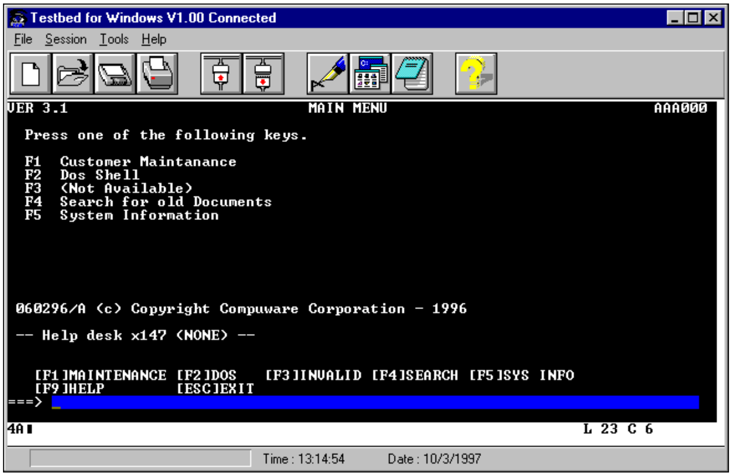
The VIRTUAL MACHINE/SYSTEM PRODUCT screen is Testbed's logon screen and is frequently referred to simply as the logon screen throughout this guide.

2. Use the following information to complete the necessary fields to logon to Testbed:

   a. Type **CW** in the USERID field and press **Tab**.

   b. Type **PASS** in the PASSWORD field and press **Enter**.

   After logging on, Testbed's MAIN MENU screen displays:



```
VER 3.1                               MAIN MENU                                AAA000

  Press one of the following keys.

  F1   Customer Maintanance
  F2   Dos Shell
  F3   (Not Available)
  F4   Search for old Documents
  F5   System Information



 060296/A (c) Copyright Compuware Corporation - 1996

 -- Help desk x147 (NONE) --

   [F1]MAINTENANCE [F2]DOS    [F3]INVALID [F4]SEARCH [F5]SYS INFO
   [F9]HELP        [ESC]EXIT
===> _
4A▮                                                                   L 23 C 6
```

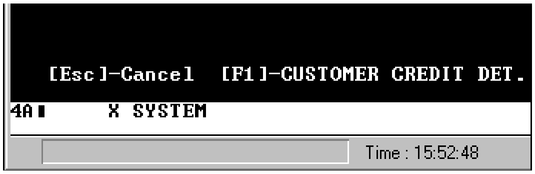
Time : 13:14:54 Date : 10/3/1997

This screen displays a list of options that can be accessed by pressing the function keys at the bottom of each screen.

3. Press the **Escape** key to return to the previous screen.

Busy Indicator

As Testbed moves between screens or processes instructions, a “busy indicator” (also called the “system light”), displays at the bottom of the screen. In Testbed, the busy indicator is “X SYSTEM.”



```
   [Esc]-Cancel   [F1]-CUSTOMER CREDIT DET.
4A▮      X SYSTEM
```

Time : 15:52:48

When the busy indicator is displayed, Testbed cannot accept further input from the keyboard. If you try to enter data while the busy indicator is showing, Testbed issues the error message “X-f” on the status line. Testbed cannot continue processing until the error message is cleared. To clear the error message, press the **F11** function key.

Disconnecting from Sessions

To properly disconnect from a Testbed session, you must return to Testbed's logon screen. Use the following procedure to disconnect from a Testbed session.

1. Each time you press the **Escape** key, Testbed exits the current screen. Press the **Escape** key as many times as necessary to return to the logon screen.

   Remember to wait for the "X SYSTEM" busy indicator to clear before you press **Escape** key again.

2. Click the **Disconnect** Button from the logon screen's toolbar.
3. When the system prompts you to confirm the disconnection, click **Yes**. Testbed clears the emulation screen.
4. From the **File** menu, choose **Exit** close Testbed.

Chapter 2. Building a Synchronized Driver Script

The following exercises teach you how to create a driver script that runs the target application, Testbed.

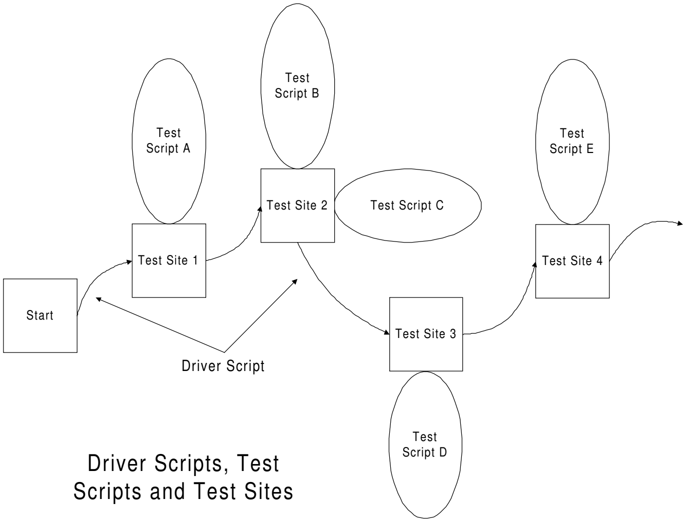
This chapter contains the following exercises that will help you understand how to build and implement a complete driver script:

- “Exercise 1 — Creating a Driver Script” demonstrates the synchronization problems that can occur when automating character-based applications. The remaining exercises explain how to overcome these problems to build a robust driver script.
- “Exercise 2 — Implementing Basic Synchronization” explains some basic synchronization techniques and demonstrates how to use Wait events and Arrived At statements as a basic means of synchronization.
- “Exercise 3 — Synchronizing Scripts Using System Variables” demonstrates how to create an external script that contains synchronization commands and system variables as a more reliable synchronization method. The exercise explains how to include this external script in each individual test script.
- “Exercise 4 — Completing the Driver Script” demonstrates how to create a new driver script that contains AutoWait logic as the synchronization technique.

Driver Scripts

A driver script merely “drives” the target application to the proper point to begin a test (called a test site) and then calls the individual test scripts that begin testing the test site. When the test script finishes, the driver script regains control and proceeds to the next test site. A driver script does not conduct any actual testing; the testing is actually done by the test scripts that are called by the driver script.

The following illustration demonstrates how a driver script interacts with test scripts.



Driver Scripts, Test Scripts and Test Sites

A well-designed test script returns the target application to the original test site before completion. This ensures that:

- Driver scripts can always pick up from where they left off.
- You can add or remove tests at a test site without modifying the driver path.

Exercise 1 — Creating a Driver Script

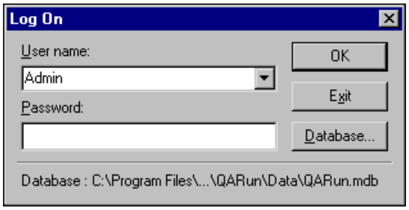
The quickest way to understand how *QA*Run learns to drive an application is to try it out. The purpose of this first exercise is to build a simple driver script. During this exercise you will generate a script that starts Testbed, connects to a session, logs on at the main screen, and selects an option from the menu screen. You will use *QA*Run's Learn function to generate the script. Before you begin learning the script, however, you'll need to change a few of *QA*Run's configuration options in order to ensure that your test scripts resemble the examples provided throughout this guide.

Starting *QA*Run

Use the following procedure to launch *QA*Run and to log on to the *QA*Run database:

1. Start *QA*Run by clicking the taskbar **Start** button and choosing **Programs>Compuware>QARun** from the **Start** menu.

   Choose the ***QA***Run icon. The *QA*Run splash screen displays followed by the *QA*Run Log On dialog box:

   Log On

   User name: Admin

   Password:

   Database : C:\Program Files\...\QARun\Data\QARun.mdb

2. If your userID has been added to the User Table, select your name from the **User name** drop-down list and enter your password in the **Password** field.

   If you are using the software for the first time, type **Admin** to log on. The default password for this user is “Admin”. You should change this password once you have logged on to prevent unauthorized access.

3. Click **OK** to log on.

Configuring *QA*Run

Before you begin creating the driver script, you need to change some of *QA*Run's configuration options. Changing these options will make your resulting driver script more closely resemble the examples in the following exercises and allow you to test other versions of Testbed. Specifically, you'll need to change these options:

- **Script Editor —** You will change the amount of time *QA*Run will wait for an event to occur, and you will change the way that the event is inserted into your script. In addition, it is necessary to change the default script's text and change the way

*QA*Run learns pauses between your actions. You will also tell *QA*Run to Learn image selections as mouse-clicks.

- **Object Map —** Because you will eventually use the test scripts created using Testbed Version 1 to test Testbed Version 2, it is important that any information found in the screen attach names that relates to the product's version is ignored. To accomplish this, you will create a significant fields mask that tells *QA*Run not to recognize the attach name's title as a significant field in the Object Map.

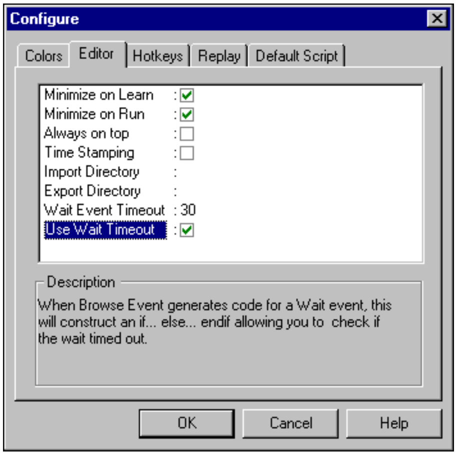
  The procedures that follow describe how to create and use a significant field mask so that each window that is learned from Testbed ignores the product's version.

You must change the specific configuration options described below in order to generate scripts that closely resemble the examples provided throughout this guide; however, if you wish, you may also change some of the Script Editor's general configuration options to suit your personal preferences. For example, you may change the fonts and colors used in the Script Editor, and you may also change the way the Script Editor behaves when it is active. If you wish to alter any of these general configuration options, refer to the section on configuring the Script Editor in the *QARun User's Guide*.

Changing Configuration Options

Use the following procedure to change the Script Editor's configuration options:

1. Click the **Editor** button on the Modules toolbar to open the Script Editor.
2. From the Script Editor's **Options** menu, choose **Configure**. The Configure dialog box displays.



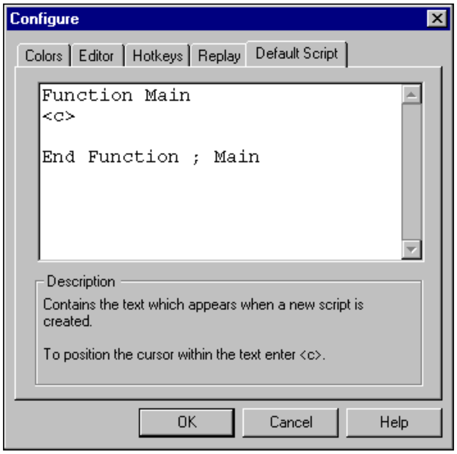
Minimize on Learn : ☑
Minimize on Run : ☑
Always on top : ☐
Time Stamping : ☐
Import Directory :
Export Directory :
Wait Event Timeout : 30
Use Wait Timeout : ☑

Description
When Browse Event generates code for a Wait event, this will construct an if... else... endif allowing you to check if the wait timed out.

3. Click the **Editor** tab and double-click the **Wait Event Timeout** option.
4. If necessary, type **30** in the **New value** field and click **OK**. The Configure dialog box re-appears.

5. Clear the **Use Wait Timeout** check box. Clearing this option inserts events into your script without requiring you to use the If...Else event logic. In later exercises, you will turn this option on and create the event logic that is typically found in more robust scripts.

6. Click the **Default Script** tab.



```
Function Main
<c>

End Function ; Main
```

Description

Contains the text which appears when a new script is created.

To position the cursor within the text enter <c>.

*QA*Run's default script allows you to specify information that is automatically placed into each subsequent script that is created. By default, *QA*Run provides error handling text in the default script, but you can change the default script to anything. In future exercises, you'll use the default script to make function calls.

7. Error handling will not be used for the exercises that follow. Remove all information from the default script except the following code:

```
Function Main
<c>
End Function ; Main
```

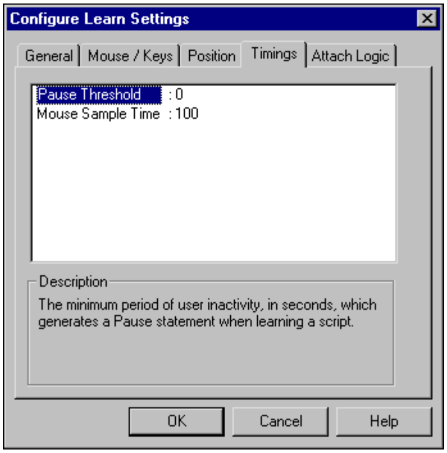
8. Click **OK**.

Changing Learn Settings

1. Click the **Editor** button on the Modules toolbar to open the Script Editor.

2. From the Script Editor's **Options** menu, choose **Learn Settings**. The Configure Learn Settings dialog box displays.

3. Click the **Timings** tab and double-click the **Pause Threshold** option.



Pause Threshold : 0
Mouse Sample Time : 100

Description
The minimum period of user inactivity, in seconds, which generates a Pause statement when learning a script.

4. If necessary, type **0** in the **New value** field to and click **OK**.

   The Pause Threshold value is the amount of time that *QA*Run will wait before learning pauses between learned actions into the script. Setting the value to 0 tells *QA*Run not to Learn your pauses, so your scripts replay as fast as possible.

5. Click the **General** tab and ensure that the **BitmapSelects** and **TypeToControl** check boxes are cleared.

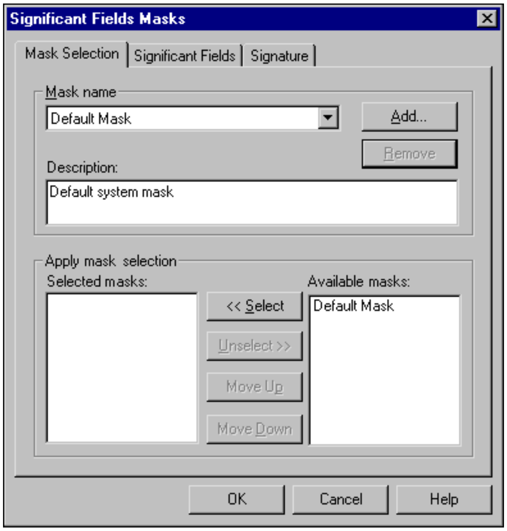
Changing the Significant Fields Mask

Because you will eventually use the test scripts created using Testbed Version 1 to test Testbed Version 2, it is important that any information found in the screen attach names that relates to the product's version is ignored. To accomplish this, you will create a significant fields mask that tells *QA*Run not to recognize the attach name's title as a significant field in the Object Map.

Use the following procedure to create and use an Object Map significant fields mask:

1. Click the **Object Map** button on the Module's toolbar. The Object Map window displays.

2. From the **Options** menu, choose **Significant Fields Masks**. The Significant Fields Masks dialog box displays:



Mask name: Default Mask

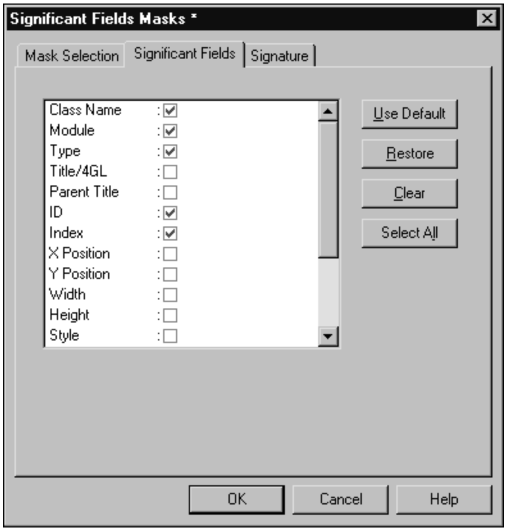
Description: Default system mask

Available masks: Default Mask

3. Click the **Add** button to add a new mask name. The *QA*Run dialog box displays.

4. In the **Add mask name** field, type the name **Testbed** and click **OK**.

5. Click the **Significant Fields** tab. The following information displays:

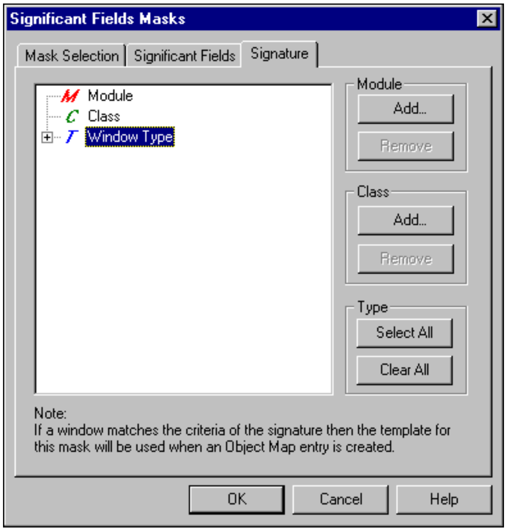


Class Name	: ☑
Module	: ☑
Type	: ☑
Title/4GL	: □
Parent Title	: □
ID	: ☑
Index	: ☑
X Position	: □
Y Position	: □
Width	: □
Height	: □
Style	: □

This dialog box allows you to specify the fields that will be considered when objects are learned using the mask you are creating.

6. Clear the **Title** and **Parent title** check boxes.

7. Click the **Signature** tab. The following information displays:



Note:
If a window matches the criteria of the signature then the template for this mask will be used when an Object Map entry is created.

This dialog box allows you to specify the modules, classes, and window types that the mask's significant fields selections will be applied to. For the purposes of this exercise, you want to apply the mask settings to Testbed's windows.

8. In the **Module** group, click the **Add** button, type **TESTBED.EXE** in the **Name of module** field, and click **OK**.

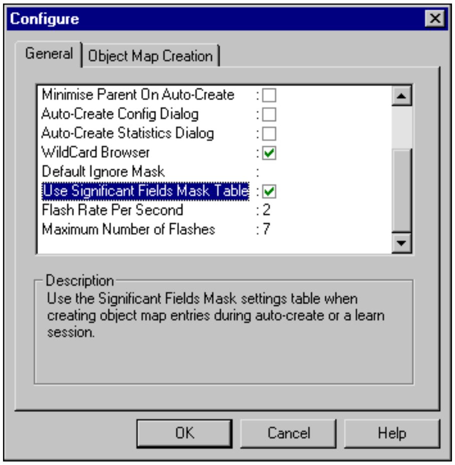
9. Click the **Mask Selection** tab to tell *QA*Run to use the Testbed mask you just created.

10. In the **Available masks** list, select **Testbed** and click the **Select** button. The **Testbed** mask should now appear in the **Selected masks** list.

    This tells *QA*Run to use the Testbed mask when learning objects from the Testbed.exe module and to use the Default Mask mask when learning any objects that do not belong to the Testbed.exe module.

11. Click **OK** to save the significant fields mask and return to the Object Map display.

12. From the Object Map's **Options** menu, choose **Configure**. The Configure dialog box displays:



Minimise Parent On Auto-Create : □
Auto-Create Config Dialog : □
Auto-Create Statistics Dialog : □
WildCard Browser : ☑
Default Ignore Mask :
Use Significant Fields Mask Table : ☑
Flash Rate Per Second : 2
Maximum Number of Flashes : 7

Description
Use the Significant Fields Mask settings table when creating object map entries during auto-create or a learn session.

13. Select the **Use Significant Fields Mask Table** check box and click **OK**.

14. Close the Object Map.

Getting Started

Now that you have changed *QA*Run's configuration options, you are ready to begin the exercise. This exercise is designed to create a simple driver script.

Exercise Prerequisites: Before beginning this exercise you must have completed the following prerequisites:

- Installed *QA*Run properly
- Changed *QA*Run's configuration options (see page 2-3)
- Closed any open *QA*Run scripts.

Testing Requirements: The following test elements are created during this exercise:

- Script named *Example 1 Script*.

You should allow 10–15 minutes to complete this exercise.

Learning the Script

Use the following procedure to create a new script for this exercise:

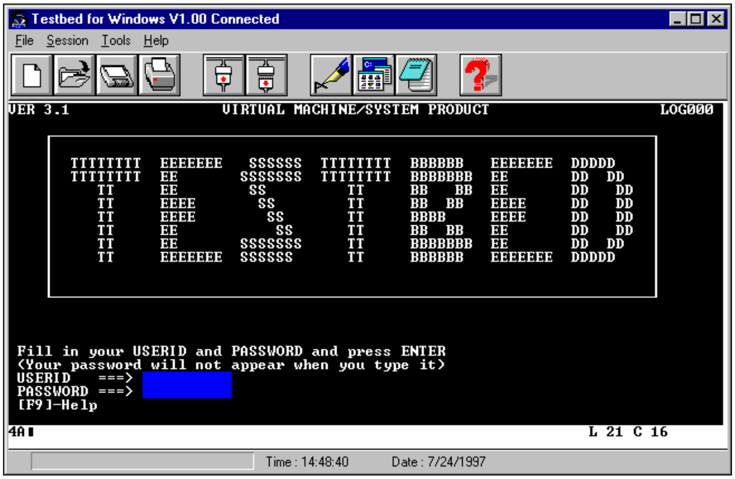
1. Start *QA*Run and click the **New** button on the toolbar. The New dialog box displays.
2. Select **Script** from the **Create new** list and click **OK**. The Script window opens and shows the default information for a new script.
3. Click the **Learn Script** button on the toolbar. *QA*Run minimizes, allowing clear access to the desktop.
4. Start the target application, Testbed Version 1:

   Although *QA*Run can execute an application from anywhere in the system, the safest way to launch the target is to use the **Start** button on the taskbar.

   - Click the **Start** button on the taskbar and choose **Run** from the menu. In the Run dialog box, enter Testbed's directory location. The default path is:

     ```
     "C:\Program Files\Compuware\QARun\Demos\TESTBED.EXE" -v1
     ```

   - Click **OK**.
5. When Testbed appears, click the **Connect** button on Testbed's toolbar. The Testbed logon screen appears:



```
VER 3.1                  VIRTUAL MACHINE/SYSTEM PRODUCT                   LOG000

      TTTTTTTT  EEEEEEE   SSSSSS  TTTTTTTT  BBBBBB   EEEEEEE  DDDDD
      TTTTTTTT  EE       SSSSSSS  TTTTTTTT  BBBBBBB  EE       DD  DD
         TT     EE       SS          TT     BB   BB  EE       DD   DD
         TT     EEEE      SS         TT     BB  BB   EEEE     DD   DD
         TT     EEEE       SS        TT     BBBB     EEEE     DD   DD
         TT     EE          SS       TT     BB  BB   EE       DD   DD
         TT     EE       SSSSSSS     TT     BBBBBBB  EE       DD  DD
         TT     EEEEEEE  SSSSSS      TT     BBBBBB   EEEEEEE  DDDDD

Fill in your USERID and PASSWORD and press ENTER
(Your password will not appear when you type it)
USERID   ===>
PASSWORD ===>
[F9]-Help
4A                                                              L 21 C 16
```

Time : 14:48:40   Date : 7/24/1997

6. Logon to Testbed:

   a. Type **CW** in the **USERID** field and press **Tab**.

   b. Type **PASS** in the **PASSWORD** field and press **Enter**.

   After logging on, Testbed's MAIN MENU screen displays.

7. Press the **F1** key to access the CUSTOMER MASTER MAINTENANCE MENU screen

8. Press the **Esc** key to return to Testbed’s MAIN MENU.

9. Click the **Disconnect** button from the logon screen’s toolbar.

10. When the system prompts you to confirm the disconnection, click **Yes**. Testbed clears the emulation screen.

11. From the **File** menu, choose **Exit** to close Testbed.

Stop Learning the Script

Use the following procedure to stop learning the driver script:

1. To stop learning, press the **Learn Hotkey, {Alt {F10}}**.

   Hotkeys allow you to access *QA*Run’s major functions (control Learn, create events, create checks, etc.) while Learn is active. The Hotkey isn’t learned in the script or by any other application.

   If the key combination assigned to the hotkey is already used by your target application, you can change the hotkey assignment by selecting Configure from the Script Editor’s Options menu.

2. After you press the **Learn Hotkey** to stop learning, *QA*Run reappears and displays the contents of the captured script in the Script Editor’s window.

The Resulting Script

After you turn Learn off, *QA*Run displays the contents of the script. Your script should resemble the following example:

```
Function Main

Attach "PopupWindow~1"
   Button "Start", 'Left SingleClick'
   PopupMenuSelect "Run..."

Attach "Run PopupWindow"
   ComboText "&Open:",
      """C:\Program Files\Compuware\QARun\Demos\TESTBED.EXE"" -v1"
   Button "OK", 'Left SingleClick'

Attach "Testbed for Windows V1.00  MainWindow"

Attach "Testbed for Windows V1.00  ChildWindow~1"
   MouseClick 186, 20, 'Left SingleClick

Attach "Testbed for Windows V1.00 Connected MainWindow"
   Type "CW{Tab}PASS{Return}"
   Type "{F1}"
   Type "{Esc}"

Attach "Testbed for Windows V1.00 Connected ChildWindow~1"
   MouseClick 239, 31, 'Left SingleClick'

Attach "Testbed for Windows PopupWindow"
   Button "&Yes",'Left SingleClick'

Attach "Testbed for Windows V1.00  MainWindow"
   MenuSelect "File~Exit"

End Function ; Main
```

Note

Your script results may vary slightly from the above example if your *QA*Run system configuration is different from the one used to Learn the example script or if you learned extra mouse clicks.

Understanding the Script

The script you just learned is comprised of seven basic sections that merit explanation:

- Including a default script header
- Attaching to the target application's window
- Running the target application
- Maximizing the target application
- Connecting to the target application
- Submitting keystrokes to the target application
- Closing the target application.

This section explains how each action was recorded in *QA*Run's script language. The following is a descriptive breakdown of the script.

The Default Script Header: A script may contain a header block. The header block contains any information that you define on the Default Script tab in the Script Editor's Configuration dialog box.

Each time you create a new script, *QA*Run automatically copies any data from the Default Script tab to the new script's header block. You can use the header block to insert descriptive notes (inserted as comments) into your scripts without affecting the script's execution. In the exercises that follow, you will eventually insert Include statements into the Default Script to call synchronization logic. This way, you won't have to manually insert the logic into each individual test script.

Attaching to a Window: The attach name is probably the most important item in *QA*Run's scripting language. By attaching to a specific window, *QA*Run ensures that its actions are directed to the correct recipient. *QA*Run's Attach command is similar to you clicking on a window to bring it into focus. Because there may be many windows on the screen when a script is replayed — including windows that weren't there when the script was originally recorded, the attach name ensures that *QA*Run finds the appropriate window and makes it the active window for the commands that follow.

Scripts that are designed to test GUI applications contain many more Attach statements because a GUI application's windows are continuously appearing and disappearing from the screen. In contrast, scripts that are intended to test character-based applications contain significantly fewer Attach statements because character-based applications are usually accessed via a Windows-based terminal emulator, and the terminal emulator's window remains on the screen throughout the session. It is the content of the window that changes, rather than the actual window.

A typical Attach statement resembles the following example:

```
Attach "PopupWindow~1"
```

This is the attach name for the Windows taskbar. This attach name was generated with object mapping turned on:

Note

The example scripts throughout these exercises contain attach names that are generated by the Object Map. These names are typically shorter and easier to read than the complete attach name.

Refer to the *QARun User's Guide* or the *QARun Language Reference Manual* for a complete description of the Attach command syntax.

Running the Target Application: After attaching to the taskbar, *QA*Run learns to click the Start button and choose Run from the Start menu.

```
   Button "Start", 'Left SingleClick'
   PopupMenuSelect "Run..."
```

The Run dialog box opens automatically. *QA*Run must first attach to this window, and then it can send keystrokes and mouse clicks to it:

```
Attach "Run PopupWindow"
   ComboText "&Open:",
      """C:\Program Files\Compuware\QARun\Demos\TESTBED.EXE"" -v1"
   Button "OK", 'Left SingleClick'
```

Connecting to the Target Application: The first line from the script excerpt shown below tells *QA*Run to attach to the Testbed toolbar. The attach name shows that the toolbar is the first "child" window of the main Testbed window.

```
Attach "Testbed for Windows V1.00  ChildWindow~1"
   MouseClick 186, 20, 'Left SingleClick'
```

The script also shows a MouseClick statement, rather than the Button command that you might expect (like the Button statement that was generated when the OK button was used). This is because Testbed's toolbar is an example of a non-standard control. In this particular case, Testbed's toolbar buttons are not real buttons. They are bitmaps (pictures) that are treated by Testbed as if they were buttons when you click them.

There are many types of non-standard controls. When *QA*Run encounters a non-standard control, it automatically switches from its object-oriented method of learning to the literal recording of mouse-clicks and keystrokes.

Sending Keystrokes to the Target Application: Before *QA*Run can begin entering the appropriate userID and password to logon to Testbed, it must first attach to the Testbed logon screen. The first line from the script excerpt shows how *QA*Run attaches to the Testbed logon screen.

```
Attach "Testbed for Windows V1.00 Connected MainWindow"
    Type "CW{Tab}PASS{Return}"
    Type "{F1}"
```

The remaining two lines from the script show how the system learns the keystrokes for entering the userID and password. *QA*Run learns the keystrokes typed to Testbed as Type commands. Keys that are not characters are enclosed in curly braces (for example, the Tab key is shown as {Tab}).

Closing the Target Application: Finally, Testbed is closed by choosing Exit from the File menu.

```
    MenuSelect "File~Exit"
```

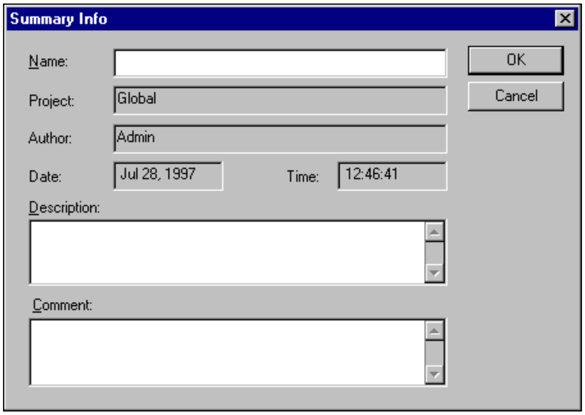
Running the Script

When you run the script, all the actions that you just learned will be replayed. Before attempting to run the script, close all active versions of Testbed. Use the following procedure to run the driver script from *QA*Run:

Caution

This script will begin to execute, but it will probably fail. The script is designed to fail for reasons described in “Analyzing the Results” on page 2-16. Later exercises will amend the script so that it runs properly.

1. Click the **Run** button on *QA*Run’s toolbar. The Summary Info dialog box displays the first time the script is run:



Summary Info

Name:

Project: Global

Author: Admin

Date: Jul 28, 1997 Time: 12:46:41

Description:

Comment:

2. Enter a file name for the script such as *Example 1 Script* (conventional Windows long file names are supported) and click **OK**. The Run Script dialog box displays.
3. Click **OK** to begin running the script. The script will begin to execute, but it will probably fail (see “Analyzing the Results” on page 2-50 for an explanation).

Analyzing the Results

Depending on the speed of your computer, your script is likely to get to Testbed’s logon screen or its MAIN MENU before Testbed beeps and locks with the “X-f” error message. The script will almost certainly fail to arrive at the CUSTOMER MASTER MAINTENANCE MENU screen.

The test script failed to run properly in this exercise, because *QA*Run replayed the script as fast as possible, but the character-based application required processing time in order to complete the requested transactions. Consequently, *QA*Run’s script statements were “out-of-synch” with the actual target application. In order to ensure proper script replay when testing character-based applications, you must implement a method of script synchronization.

Exercise 2 — Implementing Basic Synchronization, describes why the script failed to replay and introduces you to the synchronization techniques required to effectively run character-based applications.

Exercise Summary

This exercise focused on creating a basic driver script using *QA*Run’s Learn functionality. After completing this exercise, you should be able to successfully:

- Use *QA*Run’s Learn function to record a script
- Read scripts and understand script Attach statements
- Run a script.

In future exercises, we’ll discuss how to synchronize the driver script and the target application.

Exercise 2 — Implementing Basic Synchronization

The test script failed to run properly in Exercise 1 — Creating a Driver Script, because *QA*Run replayed the script as fast as possible, but the application required processing time in order to complete the requested transactions. Consequently, *QA*Run's script statements were "out-of-synch" with the target application.

In order to ensure proper script replay when testing character-based applications, you must implement a method of script synchronization. There are several different synchronization methods that you can use, and you'll find that some are more suitable than others — depending on the complexity of your script and the typical transaction times from your character-based application.

Synchronization

Let's begin by examining the reasons why the script in Exercise 1 — Creating a Driver Script failed.

Typically, whenever *QA*Run drives a GUI application, the Attach statements serve as the primary means of synchronization. Attaching to new windows as they appear helps *QA*Run keep in step with the target application.

However, in our testing environment, the target application is not a local Windows application, but a remote host-based application accessed via an emulator. When this is the case, there is only one main window. This single window remains active throughout the session; only the contents of the window change as you move from screen to screen. There are no new Attach statements to help *QA*Run synchronize with the target.

When you are testing a character-based application, you must explicitly instruct *QA*Run to wait for the target application to complete its processing before you attempt to send more keystrokes and mouse actions.

To replay the script successfully, synchronization instructions must be included each time the target application performs processing on the input received. During this processing time, the target is "busy" and no further input must be generated. Typically, the time that this processing takes is variable and depends on the complexity of the process and the overall load on the host system.

For example, when using Testbed as the character-based application, the system needs time to process the userID and password that are entered. Testbed must display the MAIN MENU before the F1 key is pressed. Without synchronization, the F1 request is sent while Testbed is busy processing the logon information — which results in the keyboard lockup.

There are several ways to build synchronization instructions into a script — from including simple *Pauses* in the script to using an advanced sequence of *Events* and *System Variables*. Each method has its own merits. However, some methods are more reliable than others. The following sections discuss these methods and their advantages and disadvantages.

Pauses in Scripts

One method of synchronizing *QA*Run with a character-based application is to insert Pauses into the script. Inserting or learning pauses into the script is the simplest, and the most *unreliable*, method of synchronizing a script with a target application. Pauses are inserted directly into the script and cause the script to temporarily pause for a specified period of time. For example:

```
Pause( 2 , 'ticks' )
Pause( 5 , 'seconds' ) ; 1 tick = 1/10 second
```

Pauses can be manually inserted into a script or generated automatically during Learn by setting the Pause Threshold option on the Script Editor's Configure dialog box. The Pause Threshold value is measured in seconds. Setting the pause threshold value to a positive number generates Pause statements each time you stop interacting with the target application for longer than the number of seconds while learning the script.

Advantages: No user interaction is required, and Pauses are simple to implement.

Disadvantages: Response times from remote applications are often variable. The pauses learned during script development may not be long enough when the script is replayed if the host system is running slowly due to heavy workloads. The pauses must be long enough to ensure that, even when the host is responding slowly, *QA*Run does not proceed too soon. Unfortunately, inserting these types of long pauses wastes time when the host is running quickly.

Events

*QA*Run has an "event-driven" programming language that allows you to define actions or system responses that *QA*Run must wait for before continuing.

Events are *QA*Run's way of monitoring what is being processed in the system. *QA*Run can recognize many types of events. For example:

- Text appearing on the screen
- Users typing on the keyboard
- Users making menu selections
- Windows appearing, being moved, resized, etc.
- Mouse actions
- Dates and times
- Graphics appearing on the screen.

*QA*Run can be instructed to “Wait” for an event to occur before continuing with the script. It can also be instructed to react to an event “Whenever” it occurs.

Waiting for screen events — for certain text to appear on the screen — is the most reliable way of synchronizing with a character-based application with variable response times.

Advantages: Using events ensures that *QA*Run does not continue to process instructions until the event has actually occurred. This way, variable system response times are automatically accounted for. Events can also be used to build “intelligence” and error handling into scripts, which makes scripts more robust.

Disadvantages: You must define a Wait event each time synchronization with the host is necessary. You must also be careful when determining the Wait event. For example, if the text defined in a screen event changes in the next version of the target application, the Wait will fail.

System Variables

*QA*Run’s scripting language has several “system variables” that allow you to dictate how *QA*Run processes scripts and reacts to the target application. When used in conjunction with events, these system variables help make scripts more reliable.

One particularly useful system variable is the Replay.AutoWait command. The Replay.AutoWait command permits the construction of automatic synchronization routines, which allow *QA*Run to keep in-step with the host system using a single event definition.

Another useful system variable, which is used in conjunction with the Replay.AutoWait command, is the Replay.ActionKeys command. The Replay.ActionKeys command allows you to define the user actions that will trigger waiting on events.

Advantages: Using the Replay.AutoWait command means that fewer screen events need to be defined and synchronization is much more robust.

Disadvantages: This method of synchronization requires the system to display a reliable and predictable sign that it is ready to receive further input. This could be something like a “busy” or “keyboard locked” indicator, which is displayed when the system is processing and removed when it is ready for input. You could also use a screen title or ID that indicates the arrival at the next screen.

Implementing this method of synchronization requires that you be familiar with *QA*Run’s scripting command language because you will need to do some programming.

Getting Started

In the following exercise, you will learn how to use screen events to synchronize the *QA*Run script with the target application. You will also learn how to use the Arrived At statement to make your scripts and log files more readable.

When complete, this driver script should start Testbed, logon to the system, and select several MAIN MENU options. The script should then return to the logon screen and disconnect the session.

Exercise Prerequisites: Before beginning this exercise you must have completed the following prerequisites:

- Access to the script named *Example 1 Script* created during “Exercise 1 — Creating a Driver Script”
- Close Testbed
- Close *QA*Run.

Testing Requirements: The following test elements are created during this exercise:

- Event named *VM/System Product Screen*
- Event named *Testbed Menu*
- Event named *Customer Maintenance Menu*
- Event named *Find Documents*
- Script named *Exercise 2*.

You should allow 20–30 minutes to complete this exercise.

Learning the Script

Use the following procedure to begin learning the script:

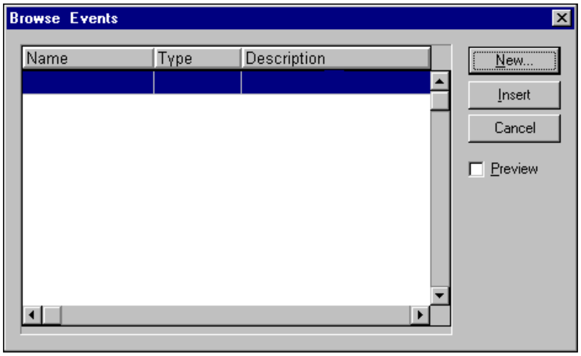
1. Start *QA*Run and click the **New** button from the toolbar to create a new script. The New dialog box displays.
2. Select **Script** from the **Create new** list and click **OK**. The Script window opens and contains the default information for the new script.
3. From the **Options** menu, choose **Learn Script**. *QA*Run minimizes to allow clear access to the desktop.
4. Start Testbed Version 1 in the normal manner (see page 2-10, if necessary).
5. When Testbed appears, click the **Connect** button.

Defining Events and Arrived At Statements

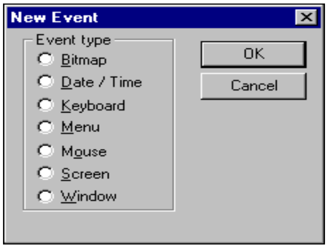
This section of the exercise creates a series a screen events that *QA*Run must wait for before sending the Arrived At statement to the log file. The events will be used to tell *QA*Run to continue script processing after each screen is reached in the target application.

Continue the above exercise using the following procedure:

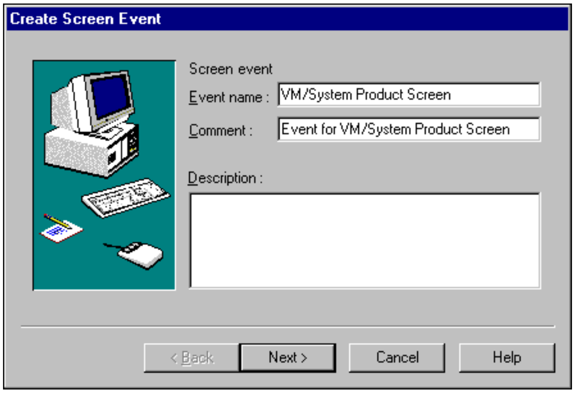
1. When the Testbed VIRTUAL MACHINE/SYSTEM PRODUCT logon screen appears, press the **Insert Event Hotkey, {Alt{F7}}**. The Browse Events dialog box displays:



2. Click the **New** button. The New Event dialog box displays:



3. Select the **Screen** option and click **OK**. The Create Screen Event dialog box displays:



Create Screen Event

Screen event

Event name : VM/System Product Screen

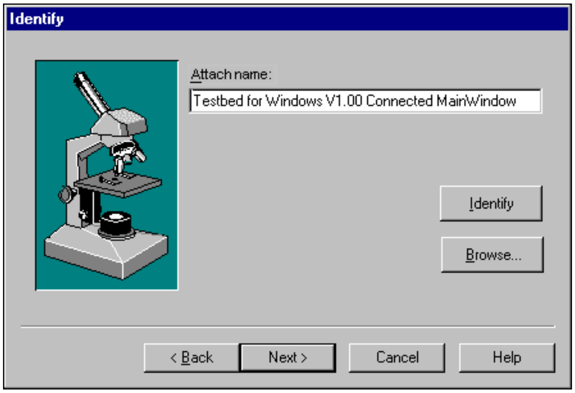
Comment : Event for VM/System Product Screen

Description :

4. Enter an event name such as, *VM/System Product Screen*, in the **Event name** field.

   Text entered in the **Comment** field will be inserted into the script as a comment preceding the event statement. Entering comments is always good practice; however, the sample scripts in this guide rarely include comments. If you insert comments, your script results will be slightly different than those documented in this book.

5. Click **Next**. The Identify dialog box displays:



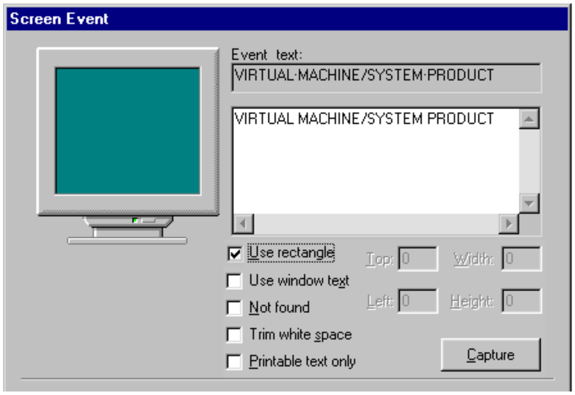
Identify

Attach name: Testbed for Windows V1.00 Connected MainWindow

6. Click the **Identify** button. *QA*Run is minimized and Testbed becomes visible.

   Position the pointer over Testbed's MainWindow and single-click. *QA*Run is restored and the window name appears in the Attach area of the Identify dialog box.

7. Click **Next**. The Screen Event dialog box displays:



Event text: VIRTUAL·MACHINE/SYSTEM·PRODUCT

VIRTUAL MACHINE/SYSTEM PRODUCT

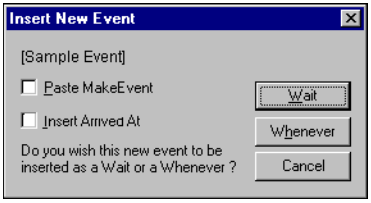
☑ Use rectangle
□ Use window text
□ Not found
□ Trim white space
□ Printable text only

Top: 0 Width: 0
Left: 0 Height: 0

8. Select the **Use rectangle** check box and click the **Capture** button. *QA*Run is minimized and Testbed becomes visible.

9. Position the cursor at the top portion of the window text "VIRTUAL MACHINE/ SYSTEM PRODUCT". Click-and-drag the cursor to the bottom-right corner of the text. Release the mouse button and the captured text displays in the text area of the Screen Event dialog box.

10. Click **Finish**. The Insert New Event dialog box displays:



[Sample Event]

□ Paste MakeEvent

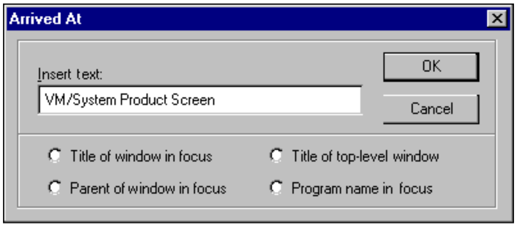
□ Insert Arrived At

Do you wish this new event to be inserted as a Wait or a Whenever ?

This dialog box allows you to use the event in either a Wait statement or a Whenever statement.

11. Select the **Insert Arrived At** check box.

12. Click the **Wait** button. The Arrived At dialog box displays:



Insert text: VM/System Product Screen

○ Title of window in focus ○ Title of top-level window
○ Parent of window in focus ○ Program name in focus

13. Type **VM/System Product screen** as name of the Testbed screen in the **Insert text** field.

14. Click **OK**. *QA*Run inserts the event and the LogComment statement into your script.

    The LogComment statement causes *QA*Run to insert an Arrived At statement into the Command Details column of the log file after the event occurs.

15. Press the **Learn Hotkey, {Alt{F10}}**, to turn Learn off and to display the captured script.

Understanding the Script

Let's focus on the section of the script that looks similar to the following example:

```
Attach "Testbed for Windows V1.00  ChildWindow~1"
   MouseClick 186, 20, 'Left SingleClick'
Wait(30, "", "VM/System Product Screen")
LogComment( "Arrived at VM/System Product Screen" )
```

In the first two lines of the above example, *QA*Run attaches to Testbed's toolbar and clicks the Connect button. The next line of the script tells *QA*Run to Wait for up to a maximum of 30 seconds for the *VM/System Product Screen* event to occur.

This event waits for the text "VIRTUAL MACHINE/SYSTEM PRODUCT" to appear within the specific coordinates identified during capture.

The last line of the example tells *QA*Run to insert the statement "Arrived at VM/System Product Screen" in the Command Details column of the log file.

Continuing the Exercise

To continue the exercise, you need to define a screen event and an Arrived At statement for each screen that appears in the target application.

Use the following procedure to continue learning the script:

1. Press the **Learn Hotkey**, **{Alt{F10}}**, to turn Learn on again. *QA*Run minimizes and Testbed's VIRTUAL MACHINE/SYSTEM PRODUCT logon screen is visible.

2. Logon to Testbed.

    a. Type **CW** in the **USERID** field and press **Tab**.

    b. Type **PASS** in the **PASSWORD** field and press **Enter**.

    After pressing **Enter**, Testbed processes the logon information and the Testbed's MAIN MENU screen displays.

3. Repeat steps 1–13 from "Defining Events and Arrived At Statements", except this time, define a new screen event named *Testbed Menu* and capture the text "MAIN MENU" for this event.

4. After defining the event and inserting the LogComment, press the **F1** key from Testbed's MAIN MENU screen.

5. Repeat steps 1–13 from "Defining Events and Arrived At Statements", except this time, define a new event named *Customer Maintenance Menu* and capture the text "CUSTOMER MASTER MAINTENANCE MENU" for this event.

6. After defining the event and inserting the LogComment, press the **Esc** key to return to Testbed's MAIN MENU screen.

7. Insert the previously defined *Testbed Menu* event (by selecting it from the Browse Screen Events dialog box) and a new LogComment to show that you have returned to Testbed's MAIN MENU screen.

Note

You must create a new screen event or insert an existing event each time the Testbed display changes. You may reuse previously defined events by selecting and inserting them from the Browse Screen Events dialog box.

You must create a new screen event or insert an existing event each time the Testbed display changes.

8. After inserting the event and a new LogComment, press the **F4** key from Testbed's MAIN MENU screen.

9. Define another new screen event named *Find Documents* and capture the text "FIND DOCUMENTS" for this event. Insert a new LogComment.

   (If necessary, refer to steps 1–13 in "Defining Events and Arrived At Statements" to complete this step.)

10. After inserting the screen event and a new LogComment, press the **Esc** key to return to Testbed's MAIN MENU screen and insert the appropriate event.

Closing Testbed

1. From Testbed's MAIN MENU screen, press the **Esc** key again to return to the VIRTUAL MACHINE/SYSTEM PRODUCT screen.

2. Again, insert the appropriate screen event and LogComment.

3. Click the **Disconnect** button on Testbed's toolbar, and click **Yes** in the resulting dialog box to confirm your choice.

4. From the **File** menu, choose **Exit** to close Testbed.

5. Press the **Learn Hotkey, {Alt{F10}}**, to turn Learn off.

The Resulting Script

After completing the above steps, your test script should look very similar to the following example:

```
Function Main

Attach "PopupWindow~1"
   Button "Start", 'Left SingleClick'
   PopupMenuSelect "Run..."

Attach "Run PopupWindow"
   ComboText "&Open:",
      """C:\Program Files\Compuware\QARun\Demos\TESTBED.EXE"" -v1"
   Button "OK", 'Left SingleClick'

Attach "Testbed for Windows V1.00  ChildWindow~1"
   MouseClick 186, 20, 'Left SingleClick'

Wait(30, "", "VM/System Product Screen")
LogComment( "Arrived at VM/System Product Screen" )

Attach "Testbed for Windows V1.00 Connected MainWindow"
    Type "CW{Tab}pass{Return}"

Wait(30, "", "Testbed Menu")
LogComment( "Arrived at Testbed Menu Screen" )
   Type "{F1}"

Wait(30, "", "Customer Maintenance Menu")
LogComment( "Arrived at Customer Master Maintenance Menu Screen" )
   Type "{Esc}"

Wait(30, "", "Testbed Menu")
LogComment( "Arrived at Testbed Menu Screen Again" )
   Type "{F4}"

Wait(30, "", "Find Documents")
LogComment( "Arrived at Find Documents Screen" )
   Type "{Esc}"

Wait(30, "", "Testbed Menu")
LogComment( "Arrived at Testbed Menu Screen Again" )
   Type "{Esc}"

Wait(30, "", "VM/System Product Screen")
LogComment( "Arrived at VM/System Product Screen Return" )

Attach "Testbed for Windows V1.00 Connected ChildWindow~1"
   MouseClick 239, 31, 'Left SingleClick'

Attach "Testbed for Windows PopupWindow"
   Button "&Yes",'Left SingleClick'

Attach "Testbed for Windows V1.00  MainWindow"
   MenuSelect "File~Exit"

End Function ; Main
```

Running the Script

Before attempting to run the script, verify that all active versions of Testbed are closed. Use the following procedure to run the driver script from *QA*Run:

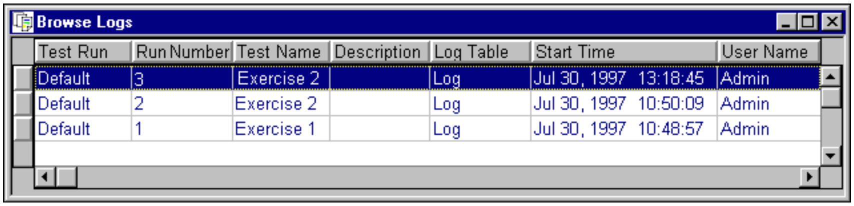
1. Click the **Run** button on *QA*Run's toolbar. The Summary Info dialog box displays the first time the script is run.
2. Enter a name for the script such as *Exercise 2* (conventional Windows long file names are supported) and click **OK**. The Run Script dialog box appears.
3. Click the **Edit** button. The Run Environment Settings dialog box displays.
4. Click the **Logging** tab and ensure that the **Logging** and **Auto Increment** check boxes are selected.
5. Click **OK**. The Run Script dialog box re-appears.
6. Click **OK** to begin running the script.

   The script should run without errors and return to the *QA*Run Script Editor display when finished. If the script encounters compilation errors, an Output window displays. Compilation error messages are displayed in this window. Double-click on an error message to automatically view the associated line of code in the script.

Analyzing the Results

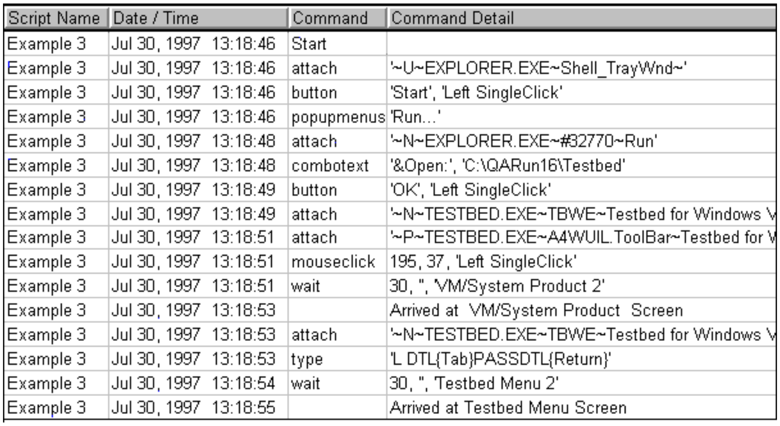
When the script finishes running, you should view the log file to verify that the script ran correctly.

1. To view the list of available logs, click the **Log Browser** button on the Script Editor's toolbar. The Browse Logs dialog box displays.



Test Run	Run Number	Test Name	Description	Log Table	Start Time	User Name
Default	3	Exercise 2		Log	Jul 30, 1997 13:18:45	Admin
Default	2	Exercise 2		Log	Jul 30, 1997 10:50:09	Admin
Default	1	Exercise 1		Log	Jul 30, 1997 10:48:57	Admin

2. Double-click on the script you just created (you may need to press the **F5** key to refresh the display). The Log View window opens and displays the log information for the script.



Script Name	Date / Time	Command	Command Detail
Example 3	Jul 30, 1997 13:18:46	Start	
Example 3	Jul 30, 1997 13:18:46	attach	'~U~EXPLORER.EXE~Shell_TrayWnd~'
Example 3	Jul 30, 1997 13:18:46	button	'Start', 'Left SingleClick'
Example 3	Jul 30, 1997 13:18:46	popupmenus	'Run...'
Example 3	Jul 30, 1997 13:18:48	attach	'~N~EXPLORER.EXE~#32770~Run'
Example 3	Jul 30, 1997 13:18:48	combotext	'&Open:', 'C:\QARun16\Testbed'
Example 3	Jul 30, 1997 13:18:49	button	'OK', 'Left SingleClick'
Example 3	Jul 30, 1997 13:18:49	attach	'~N~TESTBED.EXE~TBWE~Testbed for Windows V
Example 3	Jul 30, 1997 13:18:51	attach	'~P~TESTBED.EXE~A4WUIL.ToolBar~Testbed for V
Example 3	Jul 30, 1997 13:18:51	mouseclick	195, 37, 'Left SingleClick'
Example 3	Jul 30, 1997 13:18:51	wait	30, '', 'VM/System Product 2'
Example 3	Jul 30, 1997 13:18:53		Arrived at VM/System Product Screen
Example 3	Jul 30, 1997 13:18:53	attach	'~N~TESTBED.EXE~TBWE~Testbed for Windows V
Example 3	Jul 30, 1997 13:18:53	type	'L DTL{Tab}PASSDTL{Return}'
Example 3	Jul 30, 1997 13:18:54	wait	30, '', 'Testbed Menu 2'
Example 3	Jul 30, 1997 13:18:55		Arrived at Testbed Menu Screen

The log information shows when the script ran, what it did, and when it finished. The Log View is configurable, and your results may display different columns than those shown in the above example.

3. From the **View** menu, choose **Grid>Column Layout** and change the column layout to your desired specifications. Click **OK**.

   It is possible to define a log filter file before running the script or to filter the log file after the script runs. Filtering is discussed in more detail in later exercises, particularly when analyzing the results of checks.

Exercise Summary

This exercise focused on using screen events and Arrived At statements to synchronize scripts with the target application. After completing this exercise, you should be able to successfully:

- Define a screen event
- Insert a screen event into a Wait statement
- Create and insert an Arrived At statement
- Use Arrived At statements as LogComments
- View a script’s log
- Change the log’s column layout.

In future exercises, we’ll discuss how to avoid defining multiple events and Arrived At statements to synchronize scripts. Instead, you’ll learn how to use system variables and Whenever statements for synchronization.

Exercise 3 — Synchronizing Scripts Using System Variables

In “Exercise 2 — Implementing Basic Synchronization” you synchronized the script using a series of events that waited for specific screen information to appear. Although this is a very reliable synchronization method, it may require numerous event definitions to synchronize a script that contains many screens.

In addition to requiring multiple event definitions, this method may produce varied results if the event you are waiting for appears before the emulator processes the entire screen image. For example, if you defined an event that waits for the Testbed version number to appear in the top-left portion of the screen, and the emulator builds screen images from left to right, then it is possible for the event definition to be satisfied before that screen’s processing is complete — which could cause the script and the target application to become “out-of-sync.”

In this exercise, you will use another method of synchronization. This time, you’ll use a combination of system variables and events to synchronize the script and the target application. We’ll reverse the logic, and instead of waiting for something to appear on the screen, we’ll wait for something to *disappear* from the screen. This technique requires defining a “not found” screen event. The emulator’s “system busy” indicator makes this possible.

Like most mainframe emulators, Testbed displays a system busy indicator at the bottom of the screen whenever it processes input. When the system is busy, the words “X SYSTEM” appear at the bottom of the screen. The busy indicator displays each time a key is pressed that causes the system to process input. Keys that invoke processing are called “action keys.” In Testbed, the action keys are F1, F2, F3, F4, Return, Enter, Escape.

Testbed cannot accept further keyboard input while the X SYSTEM indicator is displayed. Attempting to input causes the “X - f” error to appear on the screen. The X SYSTEM indicator makes it possible to use the Replay.ActionKeys and Replay.AutoWait (described below) system variables in conjunction with one single screen event to achieve complete synchronization with Testbed. You can use this single event to synchronize every screen as the script drives the application. This way, it is not necessary to define a unique screen event each time the screen changes.

Replay.ActionKeys and Replay.AutoWait

The Replay.ActionKeys system variable defines the list of keys that trigger AutoWait processing. The Replay.ActionKeys list can contain any number of keystrokes and can include normal alpha keys (a, b, c), special keys (Return, Ctrl), or key combinations (Ctr + a, Ctrl + Alt + F1, Ctrl + a + b). Typically, the list defines the keys that cause the target application to begin processing data.

Replay.AutoWait is an optional system variable that specifies the length of time *QA*Run should pause after typing an action key into the target application. After an action key is pressed, the script automatically pauses to allow time for the target application to finish processing, and then it executes the instructions contained in the Whenever ActionKey construct (if defined).

Only one active Whenever ActionKey statement is allowed per script. However, a script can redefine Whenever ActionKey processing by using the Whenever statement again with a different function.

The Whenever ActionKey statement works across multiple scripts. A parent script can define a Whenever Replay.ActionKey globally for all child scripts it executes, and the individual child scripts can redefine the Replay.ActionKey locally.

Changing Configuration Options

Before you begin creating and inserting events for this exercise, you must change some of the Script Editor's configuration options. Changing the Use Wait Timeout option alters the way that Wait statements are pasted into the script — to allow you to check if the event occurred within the timeout period. Use the following procedure to change the Script Editor's configuration options:

1. From the Script Editor's **Options** menu, choose **Configure**. The Configure dialog box displays.
2. Click the **Editor** tab and ensure that the **Wait Event Timeout** option is set to **30**.
3. Select the **Use Wait Timeout** check box and click **OK**. Selecting this option causes *QA*Run to paste If … Else event logic with the Wait statement.

Getting Started

This exercise demonstrates a new, more reliable synchronization method. While implementing this technique, you will establish the AutoWait processing as a separate script that will be "included" in all your future scripts. Using the Include command to incorporate scripts into other scripts allows multiple scripts to share common code and procedures, and it provides a single source of script maintenance.

Exercise Prerequisites: Before beginning this exercise you must have completed the following prerequisites:

- Understand the concepts described in "Exercise 1 — Creating a Driver Script" and "Exercise 2 — Implementing Basic Synchronization"
- Change the Script Editor's configuration options (see "Changing Configuration Options" on page 2-30)
- Start *QA*Run
- Start Testbed.

Testing Requirements: The following test elements are created during this exercise:

- Function named *StartAutosync*
- Function named *OnAutosync*
- Event named *Autosync*
- Script named *AUTOSYNC.*

You should allow 15–20 minutes to complete this exercise.

Learning the Script and Defining *Not Found* Screen Events

Use the following procedure to begin learning the script.

1. Click the **New** button from *QA*Run's Script Editor toolbar to create a new script.

2. Rename the `Function Main` statement and the `End Function; Main` statement to the following:

```
Function StartAutosync
End Function ; StartAutosync
```

*QA*Run allows you to define only one Function Main per script (which will eventually be used in our main script), so you must rename this function because this script eventually will be included into other scripts and will not run as a stand-alone script.

3. Add the following statements after the `Function StartAutosync` statement at the top of the script new script:

```
Replay.ActionKeys = "{F1}{F2}{F3}{F4}{Return}{Enter}{Esc}"
Replay.AutoWait = 3000
```

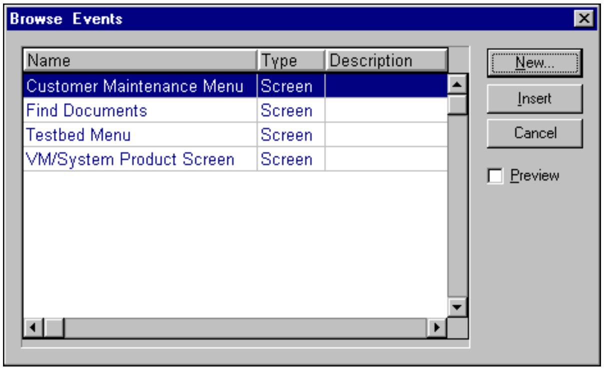
   The Replay.ActionKeys statement lists all function keys and special keys that are used by Testbed as action keys. When listing the action keys, make sure there are no spaces between the action keys; inserting spaces causes *QA*Run to treat the spaces as action keys. The Replay.AutoWait command pauses the script for a specified amount of time (defined in milliseconds) after any action key is pressed.

4. Connect to Testbed Version 1. The Testbed logon screen displays.

Defining the *Not Found* Screen Event

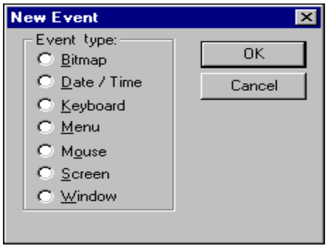
In “Exercise 2 — Implementing Basic Synchronization”, you defined a screen event in conjunction with the Arrived At statement for each screen. In this exercise, you only need to define one screen event and one Arrived At statement. Use the following procedure to define the screen event:

1. With the new *QA*Run script still open and the cursor positioned on a blank line following the `Replay.AutoWait` statement, click the **Browse Events** button on *QA*Run’s toolbar. The Browse Events dialog box displays:

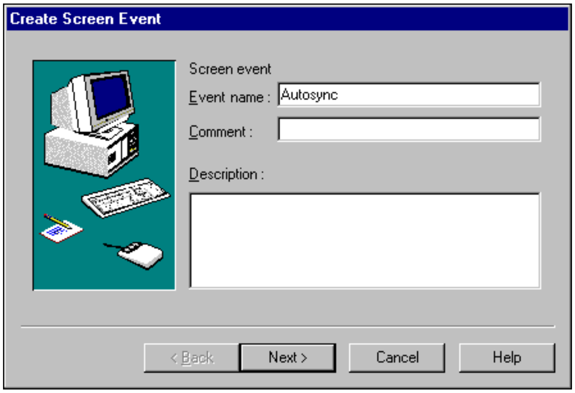


Name	Type	Description
Customer Maintenance Menu	Screen	
Find Documents	Screen	
Testbed Menu	Screen	
VM/System Product Screen	Screen	

2. Click the **New** button. The New Event dialog box displays:



3. Select the **Screen** option and click **OK**. The Create Screen Event dialog box displays:



Create Screen Event

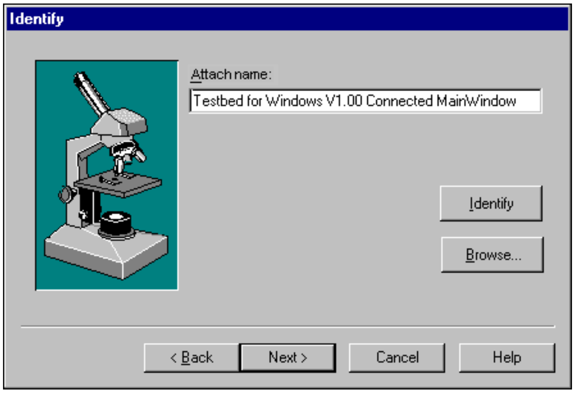
Screen event

Event name : Autosync

Comment :

Description :

4. Enter a name such as *Autosync* in the **Event name** field (no spaces are allowed) and click **Next**. The Identify dialog box displays:



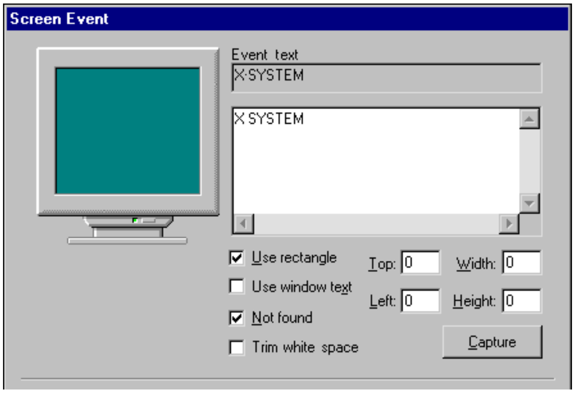
Identify

Attach name: Testbed for Windows V1.00 Connected MainWindow

5. Click the **Identify** button. *QA*Run is minimized and Testbed becomes visible.

   Position the pointer over Testbed's MainWindow title bar and single-click. *QA*Run is restored and the window name appears in the Attach area of the Identify dialog box.

6. Click **Next**. The Screen Event dialog box displays:



Screen Event

Event text: X·SYSTEM

X SYSTEM

☑ Use rectangle

□ Use window text

☑ Not found

□ Trim white space

Top: 0 Width: 0

Left: 0 Height: 0

7. Select the **Use rectangle** check box.

8. Click the **Capture** button. *QA*Run is minimized and Testbed becomes visible.

   You need to define a screen event that looks for the “X SYSTEM” busy indicator. However, the busy indicator is only available when Testbed is processing an instruction and will probably disappear before you can capture it. Because Testbed’s busy indicator always appears in the same location, so you can define the area where *QA*Run should look for it.

   Because it is difficult to judge the size of the area taken up by the X SYSTEM message, it is best to define an area that is larger than necessary. This ensures that *QA*Run recognizes the message regardless of its position on the status line.

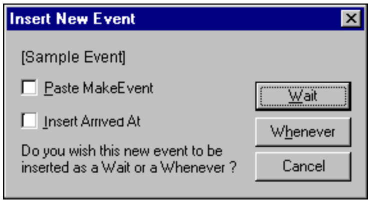
9. Drag the cursor to create a boxed outline that stretches between the `4A` symbol on the extreme left side and the `L 21 C 16` at the right side. Release the mouse button.

10. When the Screen Event dialog box reappears, delete any information and blank spaces in the text entry area and enter the text **X SYSTEM**.

11. Select the **Not found** check box at the bottom of the Screen Event dialog box.

    This option reverses the logic of the event. In the previous exercise, you specified that *QA*Run must Wait until the text “Virtual Machine/System Product” appeared in the window. In this exercise, you are telling *QA*Run to wait until the text is *not* displayed in the window. In other words, continue processing the script when “X SYSTEM” is removed (the system is not busy).

12. Click **Finish**. The Insert New Event dialog box displays:



Insert New Event

[Sample Event]

□ Paste MakeEvent

□ Insert Arrived At

Do you wish this new event to be inserted as a Wait or a Whenever ?

13. Clear the **Insert Arrived At** check box. Then, this time, click the **Whenever** button to insert the event.

The Resulting Script

Inserting the Whenever statement not only adds the Whenever statement, but it also inserts an additional function definition at the bottom of the script. This is the function that the Whenever statement calls when the event occurs. After completing the above steps, your test script should look very similar to the following example (code associated with the inserted Whenever statement is highlighted in bold typeface):

```
Function StartAutosync
    Replay.ActionKeys = "{F1}{F2}{F3}{F4}{Return}{Enter}{Esc}"
    Replay.AutoWait = 3000
    Whenever "Autosync" Call OnAutosync
End Function     ; StartAutosync
Function OnAutosync
     ;
     ; Function to handle the event 'autosync'
     ;
End Function ; OnAutosync
```

Modifying the Script

To modify the script to successfully handle the Whenever statement, you must perform the following adjustments:

- Redefine the Whenever statement
- Complete the *OnAutosync* function by inserting a Wait statement and a text panel.

Use the following procedure to modify your script:

1. Edit the line that reads `Whenever "Autosync" Call OnAutosync` to the following:

   ```
   Whenever "ActionKey" Call OnAutosync
   ```

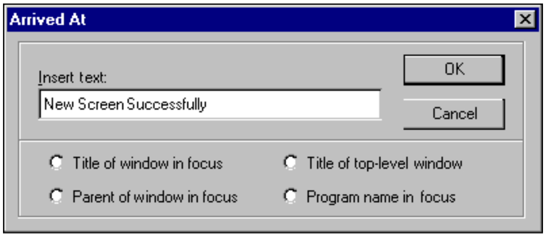
   This tells *QA*Run to call the function named *OnAutosync* each time the script uses one of the keys defined in the Replay.ActionKeys statement.

2. Place the cursor in a blank line following the `Function OnAutosync` statement and click the **Browse Events** button. The Browse Events dialog box displays.

3. Ensure that the **Preview** check box is cleared, select the event named *Autosync,* and click the **Insert** button. The Insert New Event dialog box displays.

   For this exercise, you will tell *QA*Run to write a successful LogComment to the log if the event occurs within 30 seconds. If the event does not occur within 30 seconds, a text panel displays to indicate that the host system did not respond.

4. Select the **Insert Arrived At** check box and click the **Wait** button. The Arrived At dialog box displays:



Insert text: New Screen Successfully

Arrived At
OK
Cancel
○ Title of window in focus
○ Title of top-level window
○ Parent of window in focus
○ Program name in focus

5. In the Insert text area, enter **New Screen Successfully** and click **OK**.

   A LogComment statement is added to your script that tells *QA*Run to write a comment to the log *if* the event is successful.

   The following code (highlighted in bold typeface) is inserted into your script:

```
Function OnAutosync
    If Wait(30, "", "Autosync") = 1
            ;
            ; Event Passed
            ;
            LogComment( "Arrived At New Screen Successfully" )
    Else
            ;
            ; Timeout of 30 seconds has been exceeded
            ;
    EndIf
            ;
            ; Function to handle the event 'autosync'
            ;
End Function ; OnAutosync
```

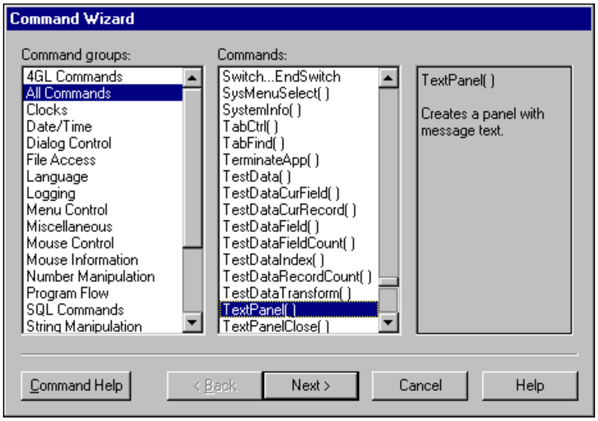
In “Exercise 2 — Implementing Basic Synchronization”, inserting a Wait statement only added one line of code to the script. However, because you changed the Script Editor’s configuration options in this script, *QA*Run inserted the Wait statement with a series of If…Else statements. This is called event logic, and it allows you to determine how *QA*Run should proceed when the defined event occurs or does *not* occur.

Note

You may remove any comment lines (beginning with “`;`”) in your script.

You now need to tell *QA*Run how to proceed if the event is not successful. To do this, you will use *QA*Run’s Command Wizard to insert a text panel into the script. The text panel will display when the event is not successful.

6. Place the cursor in the line following the `Else` statement and click the **Command Wizard** button. The Command Wizard dialog box displays:

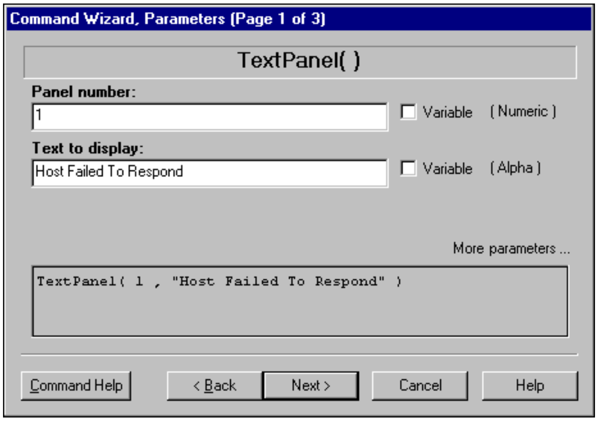


TextPanel( )

Creates a panel with message text.

The Command Wizard is a utility that assists you in selecting a command’s parameters and options. Once you’re familiar with a command’s proper syntax, you may enter it directly into the script; however, the Command Wizard is a helpful way to enter commands if you are a new *QA*Run user, or if you are not familiar with a particular command.

7. Select **TextPanel( )** from the **Command** scroll list and click **Next**. The Command Wizard, Parameters dialog box displays:



Command Wizard, Parameters (Page 1 of 3)

TextPanel( )

Panel number: 1 □ Variable (Numeric)

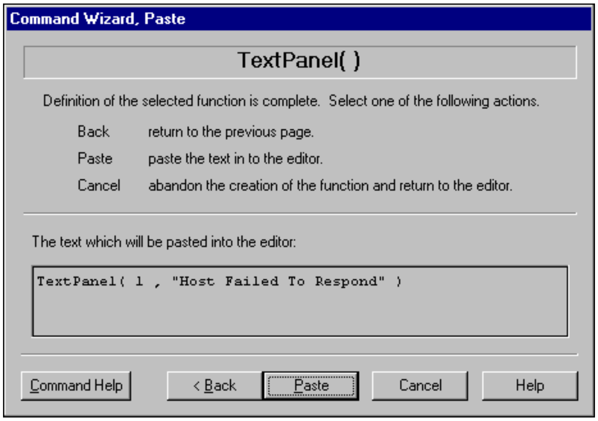
Text to display: Host Failed To Respond □ Variable (Alpha)

```
TextPanel( 1 , "Host Failed To Respond" )
```

8. Enter **1** in the Panel number area (each panel must have a unique ID number between 1 and 30) and enter **Host Failed To Respond** in the Text to display area.

9. Click **Next**.

   Two additional Command Wizard Parameters dialog boxes appear, allowing you to specify the size and position of the TextPanel. These fields are optional. Advance through the dialog boxes by clicking the **Next** button until the Command Wizard Paste dialog box displays:



Command Wizard, Paste

TextPanel( )

Definition of the selected function is complete. Select one of the following actions.

Back return to the previous page.
Paste paste the text in to the editor.
Cancel abandon the creation of the function and return to the editor.

The text which will be pasted into the editor:

```
TextPanel( 1 , "Host Failed To Respond" )
```

10. Verify that the information displayed in the command’s build area is correct and click the **Paste** button. The following code is inserted into your script:

```
TextPanel( 1 , "Host Failed To Respond" )
```

11. Remove any unnecessary comment lines from the script.

The Resulting Script

The resulting script should resemble the example below:

```
Function StartAutosync
   Replay.ActionKeys = "{F1}{F2}{F3}{F4}{Return}{Enter}{Esc}"
   Replay.AutoWait = 3000
   Whenever "ActionKey" Call OnAutosync
End Function    ; StartAutosync
Function OnAutosync
   If Wait(30, "", "Autosync") = 1
      LogComment( "Arrived At New Screen Successfully" )
   Else
      TextPanel( 1 , "Host Failed To Respond" )
   EndIf
End Function ;  OnAutosync
```

The script tells *QA*Run to wait 30 seconds after each action key is pressed and then call the *OnAutosync* function. The function waits 30 seconds for the X SYSTEM busy indicator to disappear from the screen and writes a LogComment statement to the log after the event is satisfied. A text panel displays on the screen if the busy indicator does not disappear within the allotted time period.

Saving the Script

In previous exercises, the script was automatically saved and compiled when you ran it. In this exercise, the script is not intended to run independently. You do not need to compile the script. You just need to save it so it can be included into the driver script later. Use the following procedure to save the script:

1. From the **File** menu, choose **Save**. The Summary Info dialog box displays.
2. Enter a name such as *AUTOSYNC* in the Name area, complete a brief description of the script, and click **OK**.

Exercise Summary

This exercise focused on creating a synchronization script that will be included in future test scripts. After completing this exercise, you should be able to successfully:

- Understand the Replay.ActionKeys and Replay.AutoWait system variables
- Change the Script Editor’s configuration options
- Synchronize *QA*Run scripts with character-based systems using a system busy indicator
- Define a not found screen event
- Insert commands into scripts using the Command Wizard
- Create functions that implement event logic
- Save scripts without compiling.

Exercise 4 — Completing the Driver Script

This is the final exercise for creating a driver script. When complete, this new driver script will start Testbed, logon to the system, and select a few of the available options. The script will then return to the logon screen and disconnect the session.

This driver script is similar to the one created in “Exercise 2 — Implementing Basic Synchronization”, except you will not need to create as many screen events. Instead, you will modify the driver script to include the AutoWait logic you defined in “Exercise 3 — Synchronizing Scripts Using System Variables” and use this as synchronization. This way, the script will verify that the appropriate screens are reached before continuing.

Changing Configuration Options

Before you create the script for this exercise, you need to change a Script Editor replay option. Changing this option will cause the script’s log to be automatically loaded and displayed after the script is run. Use the following procedure to change the replay option:

1. From the Script Editor’s **Options** menu, choose **Configure**. The configure dialog box displays.
2. Click the **Replay** tab and select the **Auto Load Log** check box.
3. Click **OK**.

Getting Started

This exercise demonstrates how to create a new driver script that includes another previously created script. Before you begin, make sure that Testbed is closed and start *QA*Run.

Exercise Prerequisites: Before beginning this exercise you must have completed the following prerequisites:

- Understand the concepts described in “Exercise 1 — Creating a Driver Script” and “Exercise 2 — Implementing Basic Synchronization”
- Change the Script Editor’s configuration options (see “Changing Configuration Options” on page 2-40)
- Access to the script named *AUTOSYNC* created during “Exercise 3 — Synchronizing Scripts Using System Variables”
- Access to the Event named *Autosync* created during “Exercise 3 — Synchronizing Scripts Using System Variables”
- Start *QA*Run.

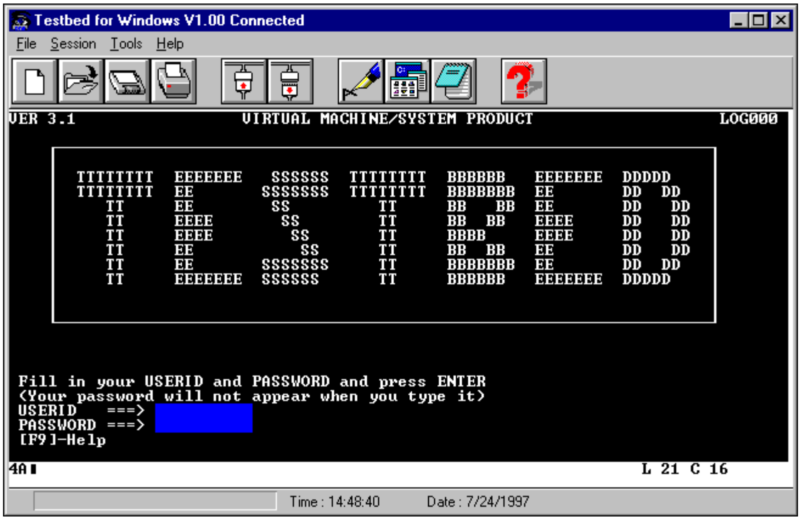
Testing Requirements: The following test elements are created during this exercise:

- Script named *Driver2*.

You should allow approximately 20 minutes for this exercise.

Learning the Script

1. Close all copies of Testbed and close the *Autosync* script created in “Exercise 3 — Synchronizing Scripts Using System Variables”.
2. From the Script Editor’s **File** menu, choose **New** to start a new script.
3. Click the **Learn Script** button on the toolbar. *QA*Run minimizes, allowing clear access to the desktop.
4. Start Testbed Version 1 in the normal manner (see page 2-10 for instructions, if necessary).
5. When Testbed appears, click the **Connect** button. Testbed’s VIRTUAL MACHINE/ SYSTEM PRODUCT logon screen displays:



```
VER 3.1                 VIRTUAL MACHINE/SYSTEM PRODUCT                   LOG000

       TTTTTTTT  EEEEEEE   SSSSSS  TTTTTTTT  BBBBBB   EEEEEEE  DDDDD
       TTTTTTTT  EE       SSSSSSS  TTTTTTTT  BBBBBBB  EE       DD  DD
          TT     EE        SS         TT     BB   BB  EE       DD   DD
          TT     EEEE       SS        TT     BB  BB   EEEE     DD   DD
          TT     EEEE        SS       TT     BBBB     EEEE     DD   DD
          TT     EE           SS      TT     BB  BB   EE       DD   DD
          TT     EE       SSSSSSS     TT     BBBBBBB  EE       DD  DD
          TT     EEEEEEE  SSSSSS      TT     BBBBBB   EEEEEEE  DDDDD

 Fill in your USERID and PASSWORD and press ENTER
 (Your password will not appear when you type it)
 USERID    ===>
 PASSWORD ===>
 [F9]-Help
4A▮                                                              L 21 C 16
```

Time : 14:48:40   Date : 7/24/1997

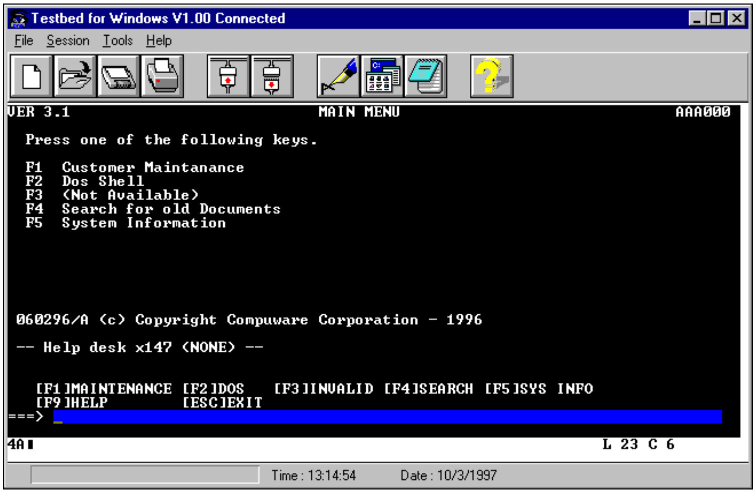
Notice the six-character screen ID in the top-right portion of the screen. In future exercises, you’ll use the screen IDs (rather than the screen titles) to identify screens.

6. Logon to Testbed:

   a. Type **CW** in the **USERID** field and press **Tab**.

   b. Type **PASS** in the **PASSWORD** field and press **Enter**.

   Notice that the busy indicator, “X SYSTEM,” appears in Testbed’s status bar while it processes the logon details. This is the event indicator that our synchronization is built on. After logging on, Testbed’s MAIN MENU (screen ID AAA000) appears:



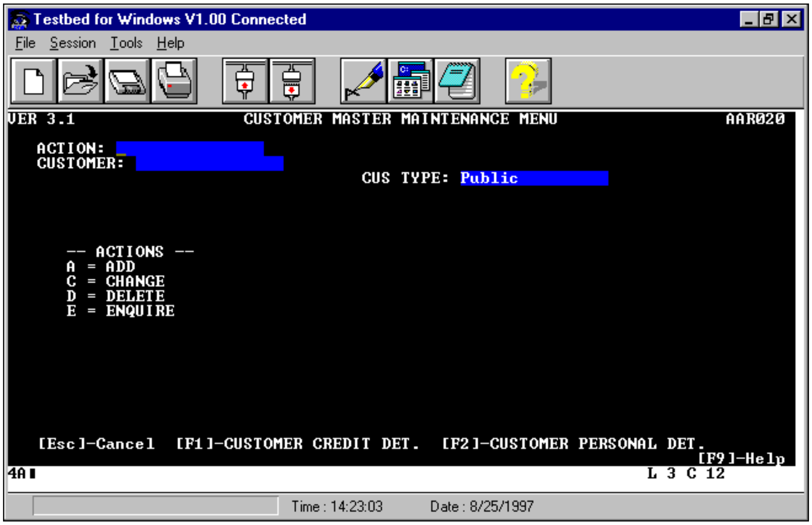
Testbed for Windows V1.00 Connected
File Session Tools Help

```
VER 3.1                                MAIN MENU                               AAA000

  Press one of the following keys.

  F1  Customer Maintanance
  F2  Dos Shell
  F3  <Not Available>
  F4  Search for old Documents
  F5  System Information




 060296/A (c) Copyright Compuware Corporation - 1996

 -- Help desk x147 <NONE> --

   [F1]MAINTENANCE [F2]DOS    [F3]INVALID [F4]SEARCH [F5]SYS INFO
   [F9]HELP        [ESC]EXIT
===> _
4A▮                                                                L 23 C 6
```

Time : 13:14:54 Date : 10/3/1997

7. Press the **F1** key to access screen ID AAR020 (CUSTOMER MASTER MAINTENANCE MENU screen).

   You do not need to insert a screen event or a LogComment because you already defined the necessary synchronization logic in the *Autosync* script, and you will include this script at the end of the exercise.



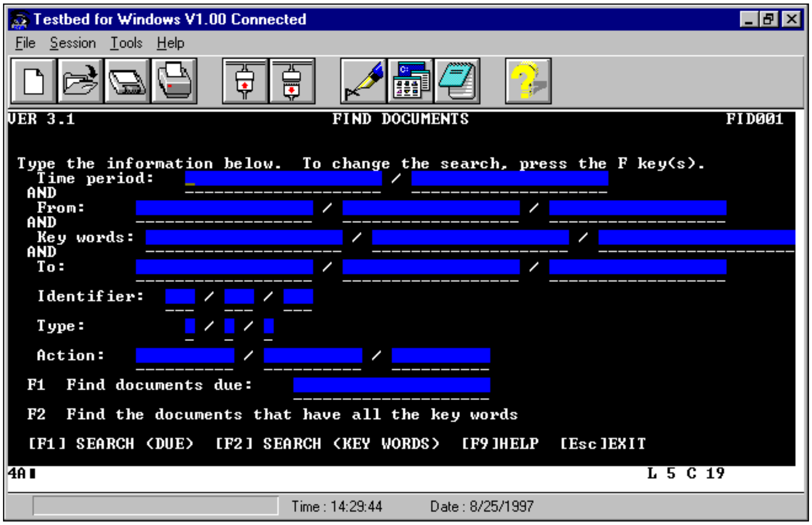
Testbed for Windows V1.00 Connected
File Session Tools Help

```
VER 3.1                    CUSTOMER MASTER MAINTENANCE MENU                    AAR020

   ACTION:
   CUSTOMER:
                                      CUS TYPE: Public



      -- ACTIONS --
      A = ADD
      C = CHANGE
      D = DELETE
      E = ENQUIRE





   [Esc]-Cancel  [F1]-CUSTOMER CREDIT DET.  [F2]-CUSTOMER PERSONAL DET.
                                                                   [F9]-Help
4A▮                                                              L 3 C 12
```

Time : 14:23:03 Date : 8/25/1997

8. Press the **Esc** key to return to Testbed’s MAIN MENU screen.

9. Press the **F4** key from Testbed’s MAIN MENU screen. Screen ID FID001 (FIND DOCUMENTS screen) displays:



```
VER 3.1                              FIND DOCUMENTS                    FID001

 Type the information below.  To change the search, press the F key(s).
   Time period: ____________________ / ____________________
  AND
   From: __________________ / __________________ / __________________
  AND
   Key words: ____________________ / ____________________ / ____________________
  AND
   To: __________________ / __________________ / __________________

   Identifier: ___ / ___ / ___

   Type:         _ / _ / _

   Action: __________ / __________ / __________

  F1  Find documents due: ____________________

  F2  Find the documents that have all the key words

  [F1] SEARCH (DUE)  [F2] SEARCH (KEY WORDS)  [F9]HELP  [Esc]EXIT
4A▮                                                          L 5 C 19
                  Time : 14:29:44       Date : 8/25/1997
```

10. Press the **Esc** key to return to Testbed's MAIN MENU screen.

Closing Testbed

After you return to Testbed's MAIN MENU screen, complete the following steps to close Testbed:

1. Press the **Esc** key again to return to Testbed's VIRTUAL MACHINE/SYSTEM PRODUCT logon screen.

2. Click the **Disconnect** button on the toolbar and click the **Yes** button from the confirmation dialog box.

3. From Testbed's **File** menu, choose **Exit**.

4. Press the **Learn Hotkey, {Alt {F10}}**, to stop learning the script. *QA*Run is restored and the script you just created is visible.

The Resulting Script

Like the driver script created in “Exercise 1 — Creating a Driver Script”, this driver script starts Testbed, logs on to the system, and selects a few of the available options. The script then returns to the logon screen and disconnects the session. The script you created in this exercise should resemble the following example:

```
Function Main
Attach "PopupWindow~1"
    Button "Start", 'Left SingleClick'
    PopupMenuSelect "Run..."
Attach "Run PopupWindow"
    ComboText "&Open:",
        """C:\Program Files\Compuware\QARun\Demos\TESTBED.EXE"" -v1"
    Button "OK", 'Left SingleClick'
Attach "Testbed for Windows V1.00  ChildWindow~1"
    MouseClick 186, 20, 'Left SingleClick'
Attach "Testbed for Windows V1.00 Connected MainWindow"
    Type "CW{Tab}pass{Return}"
    Type "{F1}{Esc}{F4}{Esc}{Esc}"
Attach "Testbed for Windows V1.00 Connected ChildWindow~1"
    MouseClick 239, 31, 'Left SingleClick'
Attach "Testbed for Windows PopupWindow"
    Button "&Yes", 'Left SingleClick'
Attach "Testbed for Windows V1.00  MainWindow"
    MenuSelect "File~Exit"
End Function ; Main
```

Modifying the Script

To modify the script so that it includes the *Autosync* script (which contains the necessary synchronization logic) and generates meaningful log comments, you must perform the following adjustments:

- Add an Include statement
- Call the *StartAutosync* function
- Separate the Type statements into individual lines.

Use the following procedure to modify your script:

1. To add the Include statement, position the cursor at the top of the driver script and insert the following code before the `Function Main` statement:

    ```
    Include "Autosync"
    ```

    The Include statement instructs *QA*Run to incorporate the contents of the *Autosync* script to this script when it compiles. During compilation, *QA*Run treats the contents of any included script as if it were pasted directly in the main script; therefore included scripts should only contain complete functions or constants. Because included scripts contain complete functions, they cannot be inserted within existing functions (Function Main, for example). They must reside outside any previously defined functions.

2. To Call the synchronization logic, position the cursor on a blank line following the `Function Main` statement and insert the following code:

```
Call StartAutosync
```

The Call statement must be inserted after the `Function Main` statement so *QA*Run can access the synchronization logic in the *Autosync* script each time an action key is pressed. *StartAutosync* is the name of the function that contains the Replay.ActionKeys, Replay.AutoWait, and Whenever statements.

Hint

Once you have created the synchronization script that will be included in other test scripts, you can insert the Include statement, Function Main statement, Call statement, and End Function statement as part of the default script information. This means that every script you create will automatically contain the synchronization logic, and you will not need to make those modifications.

To add these statements, select Configure from the Script Editor's Options menu and click the Default Script tab. The remaining procedures assume that you have added this information to the Default Script header.

3. To modify the Type statement, position your cursor on the line that reads: `Type "{F1}{Esc}{F4}{Escape}{Escape}"`. Separate the single line into individual lines as shown below:

```
Type "{F1}"
Type "{Escape}"
Type "{F4}"
Type "{Escape}"
Type "{Escape}"
```

*QA*Run Learns all action keys in a single Type statement. Separating the Type statements often makes the log file easier to understand.

After you have completed the modification described above, the beginning of the driver script should resemble the following example (changes are indicated in bold typeface):

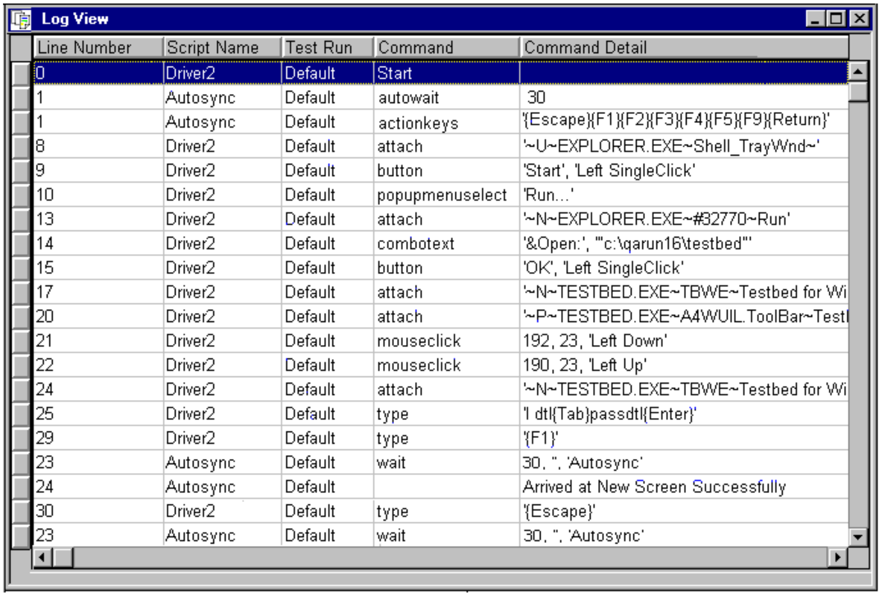
```
Include "Autosync"

Function Main
Call StartAutosync

Attach "PopupWindow~1"
    Button "Start", 'Left SingleClick'
    PopupMenuSelect "Run..."

Attach "Run PopupWindow"
    ComboText "&Open:",
        """C:\Program Files\Compuware\QARun\Demos\TESTBED.EXE"" -v1"
    Button "OK", 'Left SingleClick'

Attach "Testbed for Windows V1.00  ChildWindow~1"
    MouseClick 186, 20, 'Left SingleClick

Attach "Testbed for Windows V1.00 Connected MainWindow"
    Type "CW{Tab}pass{Return}"
    Type "{F1}"
    Type "{Escape}"
    Type "{F4}"
    Type "{Escape}"
    Type "{Escape}"
```

Running the Script

1. Click the **Run** button on *QA*Run's toolbar. The Summary Info dialog box displays.
2. Enter a script name such as *Driver2* in the **Name** area, complete a brief description of the script, and click **OK**. The Run Script dialog box displays.
3. Click **OK** to begin running the script. After the script completes, Log View displays the results of the test run.

Analyzing the Results

Once the script finishes, Log View automatically loads and displays the script results. The row and column format of the Log View display can be modified using the Grid>Column Layout command from the View menu. Your log should resemble the following example:



Line Number	Script Name	Test Run	Command	Command Detail
0	Driver2	Default	Start	
1	Autosync	Default	autowait	30
1	Autosync	Default	actionkeys	'{Escape}{F1}{F2}{F3}{F4}{F5}{F9}{Return}'
8	Driver2	Default	attach	'~U~EXPLORER.EXE~Shell_TrayWnd~'
9	Driver2	Default	button	'Start', 'Left SingleClick'
10	Driver2	Default	popupmenuselect	'Run...'
13	Driver2	Default	attach	'~N~EXPLORER.EXE~#32770~Run'
14	Driver2	Default	combotext	'&Open:', '"c:\qarun16\testbed"'
15	Driver2	Default	button	'OK', 'Left SingleClick'
17	Driver2	Default	attach	'~N~TESTBED.EXE~TBWE~Testbed for Wi
20	Driver2	Default	attach	'~P~TESTBED.EXE~A4WUIL.ToolBar~Testl
21	Driver2	Default	mouseclick	192, 23, 'Left Down'
22	Driver2	Default	mouseclick	190, 23, 'Left Up'
24	Driver2	Default	attach	'~N~TESTBED.EXE~TBWE~Testbed for Wi
25	Driver2	Default	type	'l dtl{Tab}passdtl{Enter}'
29	Driver2	Default	type	'{F1}'
23	Autosync	Default	wait	30, '', 'Autosync'
24	Autosync	Default		Arrived at New Screen Successfully
30	Driver2	Default	type	'{Escape}'
23	Autosync	Default	wait	30, '', 'Autosync'

Examine the log for the following results:

- Establishing the ActionKeys and AutoWait logic
- Exchange of commands between the *Driver2* script and the *Autosync* script
- Automatic use of the *Autosync* event after each Type command
- Automatic appearance of successful Arrived at statements after each event.

You should also note that the Arrived At statement is generic; that is, the same statement is written to the log each time a new screen is reached.

In most cases, the generic Arrived At statement should be adequate; however, it is possible to establish a single Arrived At statement that will display unique screen details each time a new screen is reached. This technique produces very specific log information. If you are interested in exploring this method, proceed to the optional exercise, “Advanced Logging Techniques” on page 2-48.

Advanced Logging Techniques

This section contains an optional exercise designed to generate specific screen-dependent Arrived At statements in the script log. This exercise is considered more advanced than the previous exercises because it requires that you edit the script to use the CaptureBox and LogOff commands.

In this exercise, you will modify the *Autosync* script and define a CaptureBox that retrieves or “captures” the screen ID from each screen as it displays. This screen ID will then be dynamically written to the Arrived At statement that appears in the log.

Use the following procedure to modify your script:

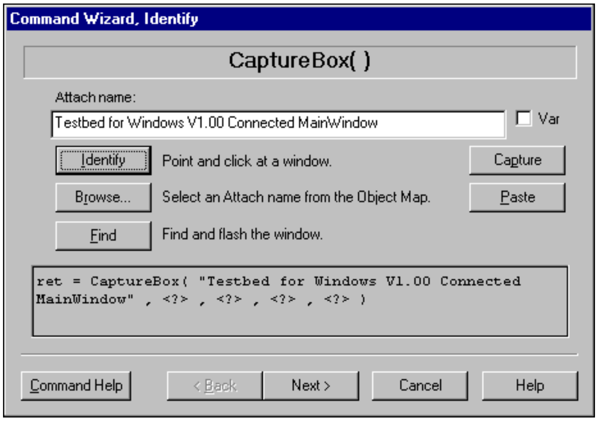
1. Make sure that Testbed is running on your desktop and click the **Connect** button.

2. Start *QA*Run and click the **Scripts** button. The Browse Scripts dialog box displays.

3. Double-click the *Autosync* script to open it.

4. Position the cursor on a blank line following the line that reads
`If Wait(30, "", "autosync") = 1.`

5. From the **Insert** menu, choose **CaptureBox**. The Command Wizard CaptureBox( ) dialog box displays:



Command Wizard, Identify

CaptureBox( )

Attach name: Testbed for Windows V1.00 Connected MainWindow

□ Var

Identify — Point and click at a window.

Browse... — Select an Attach name from the Object Map.

Find — Find and flash the window.

```
ret = CaptureBox( "Testbed for Windows V1.00 Connected
MainWindow" , <?> , <?> , <?> , <?> )
```

The CaptureBox command allows you to capture any text that is displayed in a rectangular area of the attached window.

6. Click the **Identify** button. *QA*Run is minimized and Testbed becomes visible

Position the pointer over Testbed’s MainWindow and single-click. *QA*Run is restored and the window name appears in the Attach area of the Identify dialog box.

7. Click the **Capture** button to define the specific screen area to capture.

   For this exercise, position the cursor at the beginning of the six-character screen ID located in the top-right corner of the screen.

   Screen IDs are unique for each screen displayed in Testbed. This screen ID number will be used to uniquely identify each screen in the *QA*Run log.

8. Drag the cursor over the screen area containing the six-character screen ID and release the mouse button. *QA*Run is restored and the Command Wizard's CaptureBox( ) dialog box is again displayed.

9. Click the **Paste** button. The following line of code is inserted into your script:

   ```
   ret = CaptureBox( "Testbed for Windows V1.00 Connected
      MainWindow" , 582 , 46 , 51 , 11 )
   ```

   This statement tells *QA*Run to capture the text from the specific coordinates in Testbed's MainWindow and to store it in a variable named `Ret`. To generate a detailed log, you will build the contents of the `ret` variable into the existing Arrived At statement.

10. Edit the line that reads `LogComment( "Arrived At New Screen Successfully" )` to the following (changes are indicated in bold typeface):

    ```
    LogComment( "Arrived At Screen ID [" + ret + "]" )
    ```

    This modification tells *QA*Run to paste the contents of the CaptureBox data (the screen ID) into the log's Arrived At statement each time a screen is displayed.

    To prevent the CaptureBox statement from being written to the log each time a screen is displayed and unnecessarily cluttering the log file, you can use the LogOff( ) command to tell *QA*Run not to write specific information to the log.

11. Position your cursor on a blank line following the `Function StartAutosync` statement and insert the following line of code:

    ```
    Logoff( "CaptureBox")
    ```

12. From the **File** menu, choose **Save** to save the script.

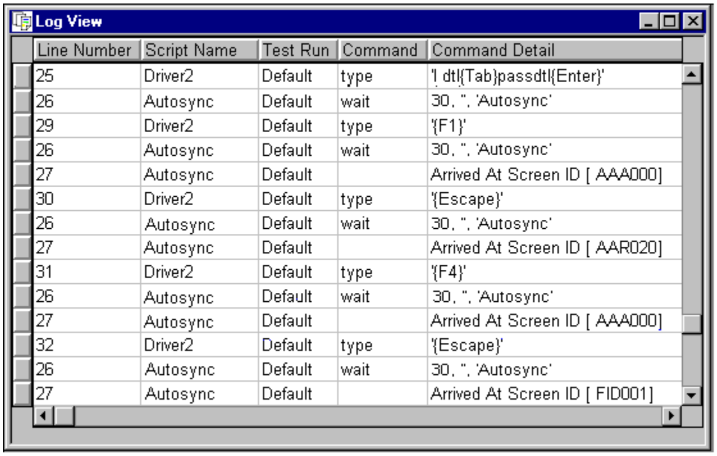
The Resulting Script

After you complete the modifications described above, the *Autosync* script should resemble the following example (changes are indicated in bold typeface):

```
Function StartAutosync
    Logoff( "CaptureBox")
    Replay.ActionKeys = "{F1}{F2}{F3}{F4}{Return}{Enter}{Escape}"
    Replay.AutoWait = 3000
    Whenever "ActionKey" Call Onautosync
End Function    ; StartAutosync

Function OnAutosync
    If Wait(30, "", "Autosync") = 1
        ret = CaptureBox( "Testbed for Windows V1.00 Connected
            MainWindow" , 575 , 47 , 58 , 10 )
        LogComment( "Arrived At Screen ID [" + ret + "]" )

    Else

        TextPanel( 1 , "Host Failed To Respond" )
    EndIf
End Function ;  OnAutosync
```

Analyzing the Results

The next time you run the *Driver2* script, Log View automatically loads and displays the script results. Your log should now resemble the following example:



Line Number	Script Name	Test Run	Command	Command Detail
25	Driver2	Default	type	'! dtl{Tab}passdtl{Enter}'
26	Autosync	Default	wait	30, '', 'Autosync'
29	Driver2	Default	type	'{F1}'
26	Autosync	Default	wait	30, '', 'Autosync'
27	Autosync	Default		Arrived At Screen ID [AAA000]
30	Driver2	Default	type	'{Escape}'
26	Autosync	Default	wait	30, '', 'Autosync'
27	Autosync	Default		Arrived At Screen ID [AAR020]
31	Driver2	Default	type	'{F4}'
26	Autosync	Default	wait	30, '', 'Autosync'
27	Autosync	Default		Arrived At Screen ID [AAA000]
32	Driver2	Default	type	'{Escape}'
26	Autosync	Default	wait	30, '', 'Autosync'
27	Autosync	Default		Arrived At Screen ID [FID001]

Figure 2-1. Sample Log View with Dynamic CaptureBox Commands

Notice that each Arrived At statement now details the screen ID (enclosed in brackets) for each screen as it is reached. In this exercise, you used the CaptureBox( ) command to capture the screen ID; however, you can use it to capture any area of information from a screen. For instance, you may want to use it to capture the screen's title instead.

Exercise Summary

This exercise focused on creating a driver script that includes a separate synchronization script into the main script and generates detailed log information. After completing this exercise, you should be able to successfully:

- Add Include statements to an existing script
- Add Include statements into the Default Script information
- Call a function from an included script
- Separate Type statements to make the log more readable
- Analyze log data
- Open an existing script (from optional exercise)
- Define a CaptureBox (from optional exercise)
- Make dynamic Arrived At statements (from optional exercise)
- Turn logging off for specific *QA*Run commands (from optional exercise).

In future exercises, we'll discuss creating individual test scripts to be used in conjunction with the driver script.

Chapter 3. Building Test Scripts

Test scripts are created the same way as driver scripts. A test script is a modular script that is designed to test specific elements of a test site. For example, a driver script might advance the target application to its main menu and then pass control to a test script that "checks" the menus for the available options. After the test script completes its check, control would be returned to the driver script, which advances the target application to the next test site and then passes control to the next test script. The driver script and test scripts should be created separately — as this makes it easier to modify tests when there is a change to the target application.

The purpose of the following exercises is to build test scripts that will eventually be called by the driver script created during "Exercise 4 — Completing the Driver Script", thus creating a complete test suite.

- "Exercise 5 — Using Text Checks" demonstrates how text checks can be used to validate data in the target application's fields. The text check can be used to check for textual, numeric, and date and time values.
- "Exercise 6 — Using Clock Checks To Test Performance" demonstrates how to check the performance of an application using a clock check. Clock checks use *QA*Run's internal stopwatches to measure the time it takes to complete a specific function.
- "Exercise 7 — Using External TestData Files" demonstrates how to enter variable data into the target application — to test the application's behavior with different input or to test behavior under heavy load conditions. The test script will carry out volume testing using an independent data file to enter information into Testbed.
- "Exercise 8 — Inserting Bitmap Checks" demonstrates how to build a simple test script that can be used to regression test. This test script will use a bitmap check as the regression comparison facility.
- "Exercise 9 — Inserting Script Dialog Boxes" demonstrates how to use *QA*Run's Dialog Editor to create a dialog box that causes your script to prompt you for userID and password information, rather than having them "hard coded" directly in the script.

Exercise 5 — Using Text Checks

To confirm that an application is working correctly, you need to verify or "check" that it is doing what it should be doing. In *QA*Run, you define what a system should be doing in the form of a "check."

A check is a definition of what the application's expected state should be at a particular point. After you create and define a check, you can insert it into your script and use it to verify the target application's state at a specific point.

*QA*Run has many different types of checks, but probably the most valuable check for mainframe applications is the text check. Text checks can be used to validate data such as ASCII text strings, numeric values, and date and time fields. *QA*Run checks actual text and is not sensitive to changes in font type or style. The purpose of this exercise is to build a simple test script that incorporates *QA*Run's text checking features.

Getting Started

In "Exercise 4 — Completing the Driver Script", you built a driver script that started Testbed, logged on to the system, and then selected some of the options from Testbed's MAIN MENU. One of those options accessed the CUSTOMER MASTER MAINTENANCE MENU screen. This exercise creates a test script that takes control from the driver script when the CUSTOMER MASTER MAINTENANCE MENU screen is reached and then makes an inquiry on an existing customer.

When the customer's details are displayed, the test script will implement a text check to check the data contained in certain fields before it returns to the CUSTOMER MASTER MAINTENANCE MENU screen and passes control back to the driver script.

Exercise Prerequisites: Before beginning this exercise you must have completed the following prerequisites:

- Access to the script named *AUTOSYNC* created during "Exercise 3 — Synchronizing Scripts Using System Variables"
- Access to the Event named *Autosync* created during "Exercise 3 — Synchronizing Scripts Using System Variables"
- Include *Autosync* script and Call statement in the default script (see page 2-45)
- Close Testbed
- Close *QA*Run.

Testing Requirements: The following test elements are created during this exercise:

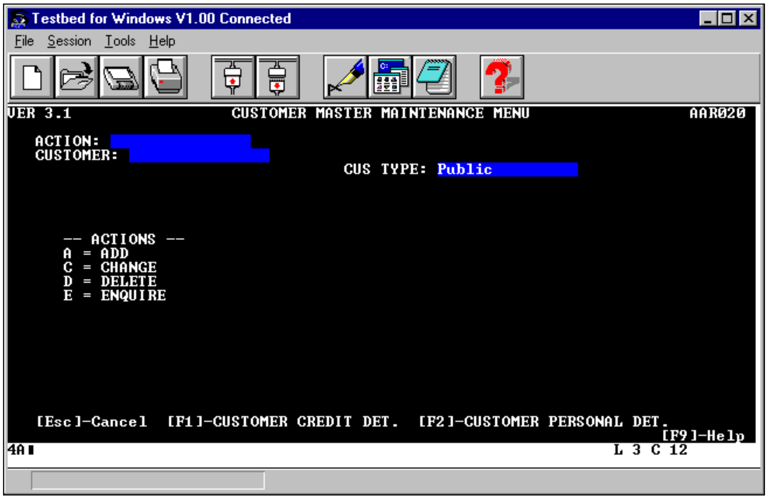
- Text check named *Master Credit Data Text Check*
- Script named *Custcred.*

You should allow 20–30 minutes to complete this exercise.

Learning the Script

Use the following procedure to advance Testbed to the CUSTOMER MASTER MAINTENANCE MENU screen and begin learning a new test script:

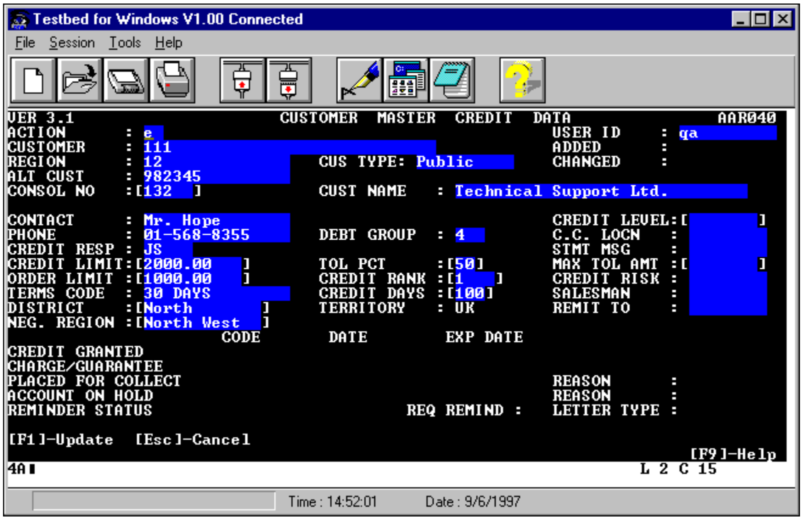
1. Start *QA*Run and open a new script (ensure the default script information contains the Include statement for the synchronization logic).
2. Start Testbed Version 1in the normal manner and logon to the system (see the procedures on page 2-10, if necessary). Testbed's MAIN MENU screen displays.
3. Press the **F1** key to select the **Customer Maintenance** option. The CUSTOMER MASTER MAINTENANCE MENU screen displays:



```
VER 3.1                  CUSTOMER MASTER MAINTENANCE MENU                AAR020

   ACTION:
   CUSTOMER:
                                         CUS TYPE: Public



      -- ACTIONS --
      A = ADD
      C = CHANGE
      D = DELETE
      E = ENQUIRE







   [Esc]-Cancel  [F1]-CUSTOMER CREDIT DET.  [F2]-CUSTOMER PERSONAL DET.
                                                                  [F9]-Help
4A                                                            L 3 C 12
```

The CUSTOMER MASTER MAINTENANCE MENU screen is the screen that this test script will start from and return to.

4. Press the **Learn Hotkey, {Alt{F10}}**, to start Learn.
5. Complete the fields on the CUSTOMER MASTER MAINTENANCE MENU with the following information:
   - **a.** Type **E** in the **ACTION** field and press **Tab** to move to the next field.
   - **b.** Type **111** into the **CUSTOMER** field. This is a valid customer record number.
6. Press the **F1** key. The CUSTOMER MASTER CREDIT DATA screen displays:



```
VER 3.1                     CUSTOMER  MASTER  CREDIT  DATA               AAR040
ACTION      : e                                         USER ID    : qa
CUSTOMER    : 111                                       ADDED      :
REGION      : 12                CUS TYPE: Public        CHANGED    :
ALT CUST    : 982345
CONSOL NO   :[132  ]            CUST NAME   : Technical Support Ltd.

CONTACT     : Mr. Hope                                  CREDIT LEVEL:[        ]
PHONE       : 01-568-8355       DEBT GROUP  : 4         C.C. LOCN   :
CREDIT RESP : JS                                        STMT MSG    :
CREDIT LIMIT:[2000.00    ]      TOL PCT     :[50]       MAX TOL AMT :[        ]
ORDER LIMIT :[1000.00    ]      CREDIT RANK :[1  ]      CREDIT RISK :
TERMS CODE  : 30 DAYS           CREDIT DAYS :[100]      SALESMAN    :
DISTRICT    :[North       ]     TERRITORY   : UK        REMIT TO    :
NEG. REGION :[North West  ]
                      CODE       DATE        EXP DATE
CREDIT GRANTED
CHARGE/GUARANTEE
PLACED FOR COLLECT                                      REASON      :
ACCOUNT ON HOLD                                         REASON      :
REMINDER STATUS                          REQ REMIND :   LETTER TYPE :

[F1]-Update  [Esc]-Cancel
                                                                      [F9]-Help
4A▮                                                              L 2 C 15
```

7. Press the **Arrived at Hotkey, {ALT{F9}}** and type **CUSTOMER MASTER CREDIT DATA** as the name of the Testbed screen in the **Insert text** field.

8. Click **OK**.

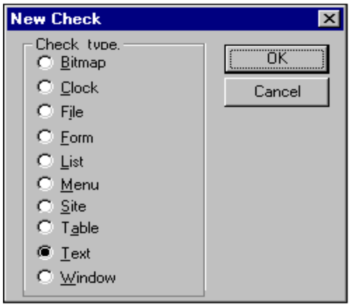
   *QA*Run inserts a LogComment statement into your script.

Defining the Text Check

The CUSTOMER MASTER CREDIT DATA screen contains detailed information on customer number 111. You will use this information as part of the text check.

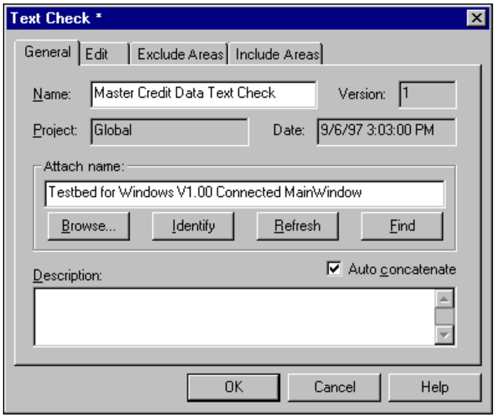
1. Press the **Insert Check Hotkey, {Alt{F8}}**. The Browse Checks dialog box displays.

2. Click the **New** button to create a new check. The New Check dialog box displays.



Check type
- ○ Bitmap
- ○ Clock
- ○ File
- ○ Form
- ○ List
- ○ Menu
- ○ Site
- ○ Table
- ● Text
- ○ Window

3. Select the **Text** option to create a new text check and click **OK**. The Text Check dialog box displays:



Text Check *

Name: Master Credit Data Text Check

Version: 1

Project: Global

Date: 9/6/97 3:03:00 PM

Attach name: Testbed for Windows V1.00 Connected MainWindow

☑ Auto concatenate

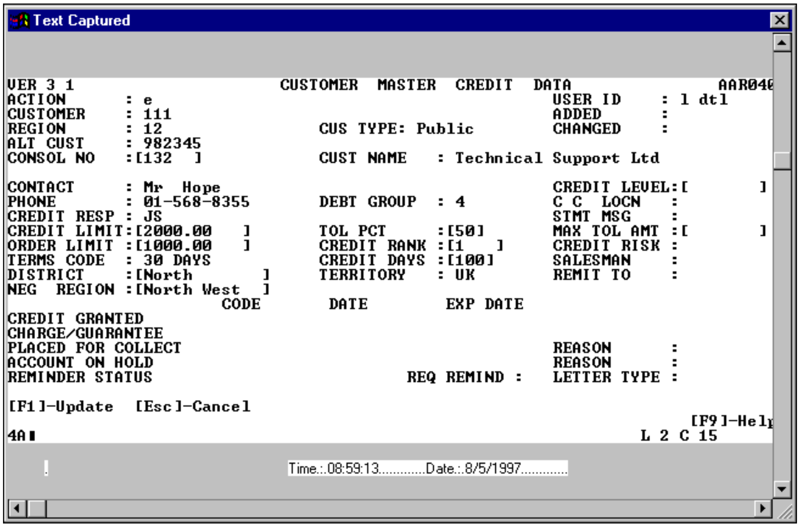
Description:

4. In the Text Check dialog box **Name** field, enter a check name such as *Master Credit Data Text Check*.

5. Click the **Identify** button. *QA*Run minimizes and the Testbed screen is now visible.

   Position the pointer over Testbed’s MainWindow and click. *QA*Run is restored and the window name appears in the **Attach name** area.

   A new window, the Text Captured window, displays the contents of the captured window (the Text Check dialog box is still open and is usually under the Text Captured window):



Text Captured

```
VER 3 1                    CUSTOMER  MASTER  CREDIT  DATA                AAR04
ACTION       : e                                      USER ID     : 1 dtl
CUSTOMER     : 111                                    ADDED       :
REGION       : 12               CUS TYPE: Public      CHANGED     :
ALT CUST     : 982345
CONSOL NO    :[132  ]           CUST NAME    : Technical Support Ltd

CONTACT      : Mr  Hope                               CREDIT LEVEL:[        ]
PHONE        : 01-568-8355      DEBT GROUP   : 4      C C  LOCN    :
CREDIT RESP  : JS                                     STMT MSG     :
CREDIT LIMIT:[2000.00    ]      TOL PCT      :[50]    MAX TOL AMT  :[        ]
ORDER LIMIT :[1000.00    ]      CREDIT RANK  :[1   ]  CREDIT RISK  :
TERMS CODE   : 30 DAYS          CREDIT DAYS  :[100]   SALESMAN     :
DISTRICT     :[North        ]   TERRITORY    : UK     REMIT TO     :
NEG  REGION  :[North West   ]
                      CODE       DATE        EXP DATE
CREDIT GRANTED
CHARGE/GUARANTEE
PLACED FOR COLLECT                                    REASON       :
ACCOUNT ON HOLD                                       REASON       :
REMINDER STATUS                        REQ REMIND :   LETTER TYPE  :

[F1]-Update  [Esc]-Cancel
                                                                   [F9]-Hel
4A▮                                                            L 2 C 15
```

Time.:.08:59:13............Date.:.8/5/1997............

You’ll use this window to define the areas where the text check looks for specific information on the screen using the Exclude Areas and Include Areas tabs.

- The Exclude Areas tab allows you to ignore portions of the screen during the text check. This is a way of ignoring values that are not significant or values that are supposed to change (such as free disk space, the date, etc.). When used, all text outside the rectangle is checked, and all text within the rectangle is omitted from the check.
- The Include Areas tab allows you to check specific areas of the screen. The include area can be defined as a date, number, time, or ASCII value. When this option is used, all text outside of the rectangle is omitted from the check.

You may define multiple include or exclude areas on any window; however, you may not define include and exclude areas in the same check.

If you define multiple include or exclude areas, the Area number spin control allows you to navigate to the areas and redisplay the contents of the rectangle. If the defined areas are include areas, then you may edit the text within the Text in area section of the dialog box. If the defined areas are exclude areas, you may not edit the text.

Defining Include Areas

In this portion of the exercise, you will define the information on the CUSTOMER MASTER CREDIT DATA screen that should be included in the check, so that *QA*Run verifies the following information:

- Testbed returned the correct customer number, 111
- The CONTACT field contains the text “Mr. Hope”
- The CREDIT LIMIT field’s value is 2000
- The ORDER LIMIT field’s value is less than 1000
- The TERMS CODE field contains two numbers and three or four alphabetic characters.

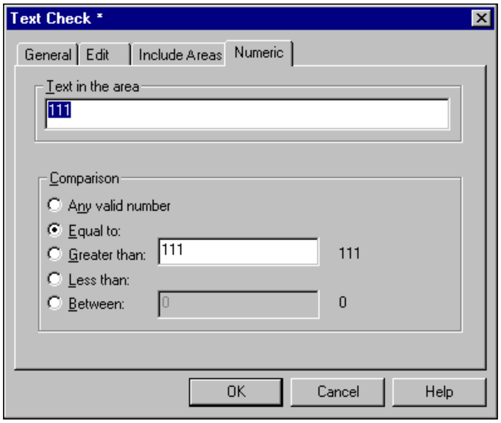
Defining the First Include Area

Use the following procedure to include an area for a text check:

1. With the *QA*Run Text Check dialog box active, click the **Include Areas** tab.
2. Click the **New** button. The Text Check dialog box is minimized and the Text Captured window displays.

   The first area that you will include in the text check is the **CUSTOMER** field.
3. Position the cursor over the top-right portion of the **CUSTOMER** field (the value 111). Click-and-drag the cursor to the bottom-left of the field and release the mouse button. After you release the mouse button, a green rectangle appears around the value “111”, and the Text Check dialog box displays with the value “111” in the **Text in the area** field.

4. Click the **Numeric** button on the Text Check dialog box's **Include Areas** tab to define the contents of the **CUSTOMER** field as a numeric value. The **Numeric** tab displays:



Text in the area: 111

Comparison
○ Any valid number
● Equal to:
○ Greater than: 111 111
○ Less than:
○ Between: 0 0

This dialog box allows you to set the text check's comparison options to check if the value in the CUSTOMER field is exactly 111, greater than 111, less than 111, between two values that you determine, or any valid number when the check is run.

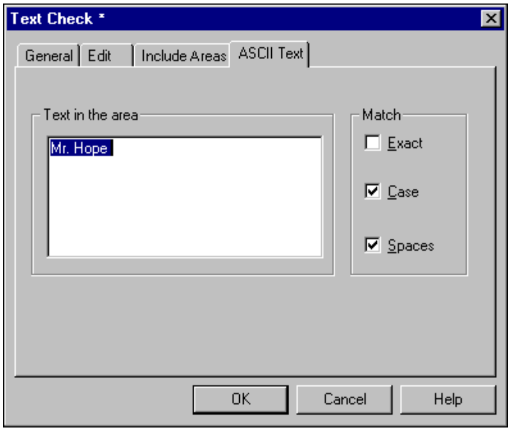
5. Select the **Equal to** option.

The first include area for this text check is now complete. Do not click OK to close the Text Check dialog box. You will continue to define additional include areas as part of this same text check.

Defining ASCII Include Areas

Use the following procedure to define additional include areas in the same text check:

1. Click the **Include Areas** tab to define a check area for the **CONTACT** field.
2. Click the **New** button. The Text Captured window displays.
3. Select the contents of the **CONTACT** field, "Mr. Hope", for this include area. It's always good practice to select an area that is larger than the actual text, to accommodate situations where the field may contain a longer name.
4. Click the **ASCII** button on the Text Check dialog box's **Include Areas** tab to define the contents of the **CONTACT** field as an ASCII value. The **ASCII Text** tab displays:



Text in the area: Mr. Hope

Match
☐ Exact
☑ Case
☑ Spaces

5. Select the **Exact**, **Case**, and **Spaces** check boxes.

   - **Exact check box —** Matches the exact text of the Include area. If additional characters are found before or after the include text, the check will fail.
   - **Case check box —** Matches the text's capitalization that appears in the include area. If you don't want capitalization to be considered, clear this check box.
   - **Spaces check box —** Matches the spaces in the text in the include area. If you do not want spaces to be considered in the text check, clear this check box.

   It is also possible to edit the current captured values and enter the values that you expect to encounter when the script is run.

6. Type **Mr. Smith** in the **Text in the area** box.

   Changing this value tells *QA*Run to expect to find the name "Mr. Smith" in Include area 2, instead of "Mr. Hope" the next time the script is run.

Defining Additional Numeric Include Areas

Use the following procedure to define the final numeric include area:

1. Click the Include Areas tab to define a numeric Include check area for the CREDIT LIMIT field. Repeat steps 2–5 from "Defining the First Include Area" to define the next Include check area, except select the value "2000.00" from the CREDIT LIMIT field.
2. To define a numeric Include area for the ORDER LIMIT field, click the Include Areas tab again and then click **New**.
3. Select "1000.00" from the ORDER LIMIT field and click the **Numeric** button.

   For this check area, you'll modify the captured value in order to use a different type of numeric check.

4. On the Numeric tab, change the value in the Text in the area field from 1000.00 to **1001.00**. Then, select the Less than radio button.

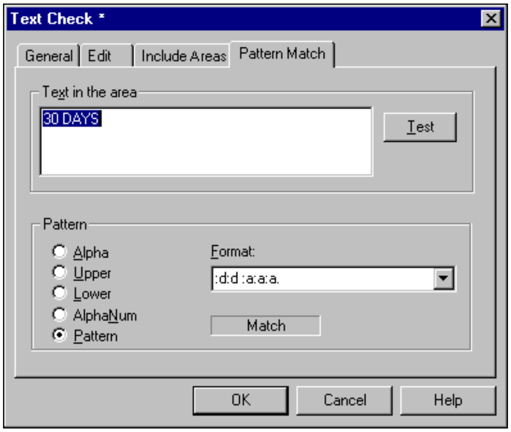
   If *QA*Run encounters any number in the check area that is less than 1001, the text check will pass.

Defining Text Patterns in Include Areas

A pattern match allows you to check an area for a specific alpha-numeric character pattern within the selected text. You will select specific comparison options to determine that the check will verify that the value and text pattern in the TERMS CODE field.

Use the following procedure to define a pattern check in an include area for the same text check:

1. To define a pattern Include area for the TERMS CODE field, click the Include Areas tab again and then click **New**.
2. Select the text "30 DAYS" from the TERMS CODE field, and click the **Pattern** button on the Include Areas tab.



Text Check *

Text in the area: 30 DAYS

Pattern:
○ Alpha
○ Upper
○ Lower
○ AlphaNum
● Pattern

Format: :d:d :a:a:a.

3. Click the **Pattern** radio button to define a specific pattern of characters.
4. Click the Format drop-down list to view *QA*Run's predefined patterns. None of the pre-defined patterns will match the "30 DAYS" text you captured, so you will need to define a new pattern match. Some of the available codes are:

:a	Any alphabetic character
:d	Any digit/number
:n	Any alphanumeric character, and **.** [a period].

   You can combine these codes into strings to create a pattern for *QA*Run to search for during the check. To view a complete list of the pattern format variables, press **F1** from the Pattern Match tab to access the context-sensitive online help.

5. Enter the following pattern definition in the Format list box.

   ```
   :d:d :a:a:a.
   ```

   Based on this pattern definition, *QA*Run expects to find two numbers, followed by a space, followed by any three alphabetic characters, and then any character other than a new line. Therefore, the check will pass if the value displayed is 30 DAYS or 01 DAY. The pattern you just defined is now added to the definitions contained in the Format field's drop-down list.

6. To verify that *QA*Run will locate the selected text based on the pattern you just defined, click the **Test** button on the Pattern Match tab. In the Text in the area field, *QA*Run highlights the text that matches the defined pattern and displays the word "Match." If the text did not match, the system displays the words "No Match" at the bottom of the tab.

7. To save the check and insert it into your script, click the **OK** button on the Text Check dialog box.

The system returns you to Testbed when the text check is complete. Remember, Learn is still on.

Continue Learning the Script

Use the following procedure to Learn the last portion of this test script:

1. Press the **Escape** key to return to the CUSTOMER MASTER MAINTENANCE MENU screen.

2. Press the **Learn Hotkey**, **{Alt{F10}}**, to stop learning the script. *QA*Run appears and the script you just defined is visible.

The Resulting Script

After completing the above steps, your test script should look very similar to the following example. The code associated with the inserted check is highlighted in bold typeface:

```
Include "Autosync"
Function Main
Call StartAutosync
    Attach "Testbed for Windows V1.00 Connected MainWindow"
        Type "E{Tab}111{F1}"
    LogComment( "Arrived at Customer Master Credit Data" )
    Check "Master Credit Data Text Check"
        Type "{Esc}"
End Function ; Main
```

This example script contains the synchronization logic that is necessary to run the script independently of the driver script. The synchronization logic is not necessary if the script will only be run as part of the driver script, because the driver script makes the logic available to all child scripts; however, if you intend to run the script independently, it must contain its own synchronization logic.

Running the Script

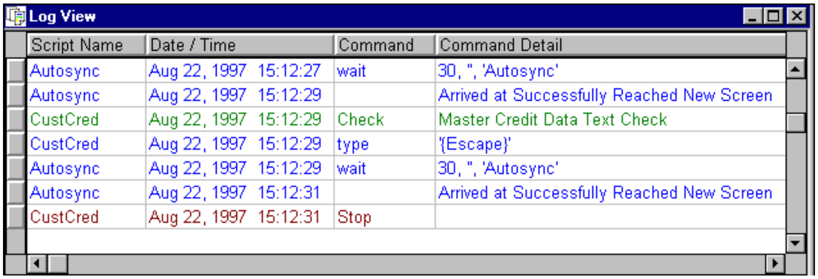
It is possible to run test scripts independently of the associated driver script if you have included the required synchronization techniques and advanced the target application to the appropriate test site. Running the test script independently is a useful way to verify that a test script runs without errors.

Use the following procedure to run the test script:

1. Click the **Run** button on *QA*Run's toolbar. The Summary Info dialog box displays.
2. Enter a name for the script such as *Custcred* in the Name field.
3. Enter a description that explains what the script does, then click **OK**.
4. Click **OK** from the Run Script dialog box to run the *Custcred* script.

Analyzing the Results

When the script finishes, Log View automatically loads with the *CustCred* log file:



Script Name	Date / Time	Command	Command Detail
Autosync	Aug 22, 1997 15:12:27	wait	30, '', 'Autosync'
Autosync	Aug 22, 1997 15:12:29		Arrived at Successfully Reached New Screen
CustCred	Aug 22, 1997 15:12:29	Check	Master Credit Data Text Check
CustCred	Aug 22, 1997 15:12:29	type	'{Escape}'
Autosync	Aug 22, 1997 15:12:29	wait	30, '', 'Autosync'
Autosync	Aug 22, 1997 15:12:31		Arrived at Successfully Reached New Screen
CustCred	Aug 22, 1997 15:12:31	Stop	

Figure 3-1. View Text Check Results

The result of the check is logged to the log file. If the check passed, it is highlighted in green (the default color for checks that pass). Failed checks are highlighted in red.

In this exercise, the *Master Credit Data Text Check* was designed to fail when you ran the script because you told *QA*Run to expect to find the text “Mr. Smith” (see steps on page 3-8), but the application contained the text “Mr. Hope.”

No further action is necessary at this point. Later, when the entire test suite is run against Testbed Version 2, many of the checks will fail. It will then be possible to analyze the differences between the expected and actual responses.

Exercise Summary

This exercise focused on creating a test script that contained a text check with multiple include areas. After completing this exercise, you should be able to successfully:

- Create new text checks
- Define a text check that verifies specific numeric values
- Define a text check that verifies specific text values
- Define a text check that verifies character and numeric patterns
- Modify a text check to verify information that is different than the original captured text
- Insert a text check into a script
- Recognize the formats used to define pattern text checks.

Exercise 6 — Using Clock Checks To Test Performance

The purpose of this exercise is to check the performance of an application using a clock check. Clock checks use *QA*Run's internal stopwatches to measure the time it takes to complete a specific function.

In "Exercise 3 — Synchronizing Scripts Using System Variables" the driver script accessed Testbed's FIND DOCUMENTS screen by pressing **F4** at Testbed's MAIN MENU screen. When complete, this next test script will instruct Testbed to find documents that contain the keyword "Bank" and measure the time it takes for Testbed's DOCUMENTS FOUND SCREEN to appear. In order to perform a true performance test, you'll use the Repeat command to make the script loop several times.

Changing Configuration Options

Before you begin creating and inserting events for this exercise, you must change some of the Script Editor's configuration options. Changing these options will cause *QA*Run to insert a simple Wait event into your script, rather than the "event logic" inserted in "Exercise 3 — Synchronizing Scripts Using System Variables".

Use the following procedure to change the Script Editor's configuration options:

1. From the Script Editor's **Options** menu, choose **Configure**. The Configure dialog box displays.
2. Click the Editor tab and double-click the Wait Event Timeout option.
3. Change the New Value field to **0** in the Wait Event Timeout dialog box and click **OK**. The Configure dialog box reappears.
4. Clear the Use Wait Timeout check box to turn it off and click **OK**.

Getting Started

Before creating the test script, close all open copies of *QA*Run and Testbed. The following procedure will open a new script, advance Testbed to the appropriate test site, and begin learning new testing activity.

Exercise Prerequisites: Before beginning this exercise, you must have completed the following prerequisites:

- Access to the script named *AUTOSYNC* created during “Exercise 3 — Synchronizing Scripts Using System Variables”
- Access to the Event named *Autosync* created during “Exercise 3 — Synchronizing Scripts Using System Variables”
- Include *Autosync* script and Call statement in the default script (see page 2-45)
- Close Testbed
- Close *QA*Run.

Testing Requirements: The following test elements are created during this exercise:

- Screen event named *FindDoc*
- Clock check named *DocumentsFound*
- Script named *FINDDOCS*.

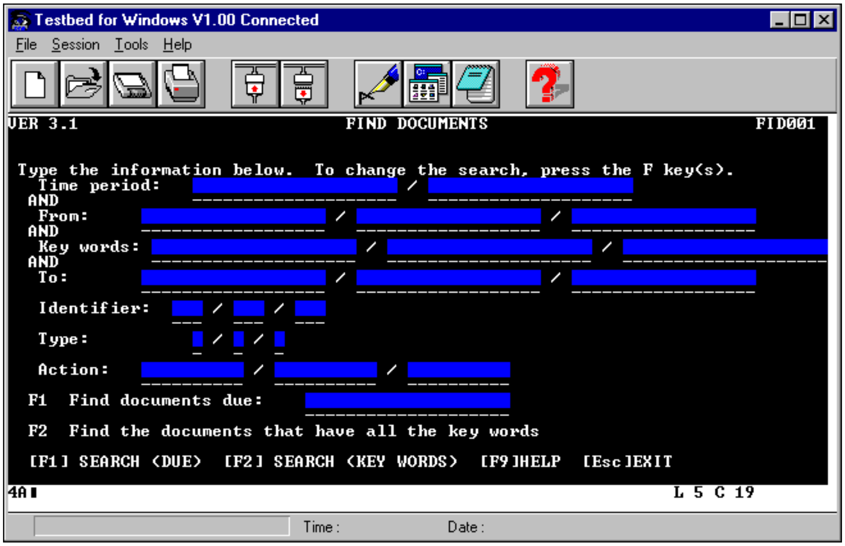
You should allow 30–45 minutes to complete this exercise.

Learning the Script

Use the following procedure to begin learning the new test script:

1. Start *QA*Run and open a new script (ensure the default script information contains the Include statement for the synchronization logic).
2. Start Testbed Version 1 using the same method described in previous exercises.
3. When Testbed appears, click the **Connect** button. The VIRTUAL MACHINE/ SYSTEM PRODUCT logon screen displays.
4. Logon to Testbed:
   a. Type **CW** in the USERID field and press **Tab**.
   b. Type **PASS** in the PASSWORD field and press **Enter**.

   After logging on, Testbed’s MAIN MENU displays.
5. Press **F4** (Search) from Testbed’s MAIN MENU. The FIND DOCUMENTS screen displays:



```
VER 3.1                          FIND DOCUMENTS                         FID001

 Type the information below.  To change the search, press the F key(s).
   Time period:                    /
  AND
   From:                      /                  /
  AND
   Key words:                  /                  /
  AND
   To:                        /                  /

   Identifier:       /     /

   Type:             /  /

   Action:                    /          /

  F1  Find documents due:

  F2  Find the documents that have all the key words

  [F1] SEARCH (DUE)  [F2] SEARCH (KEY WORDS)  [F9]HELP  [Esc]EXIT
4A                                                                L 5 C 19
```

6. Press the **Learn Hotkey**, **{Alt{F10}}**, to start Learn.

7. Press the **Tab** key until the cursor reaches the Key words field and enter the keyword **BANK**.

8. Press **F2**.

   After a short pause, Testbed displays the PROCESS THE DOCUMENTS FOUND screen. For this test script, this screen signifies the end of the process that you are timing. You must define an event that indicates when this screen has been reached and an Arrived At statement to record the screen in the log.

Defining Events and Arrived At Statements

This event will be used to tell *QA*Run that the PROCESS DOCUMENTS FOUND screen has been reached and to stop the clock. Continue with the above exercise using the following procedure:

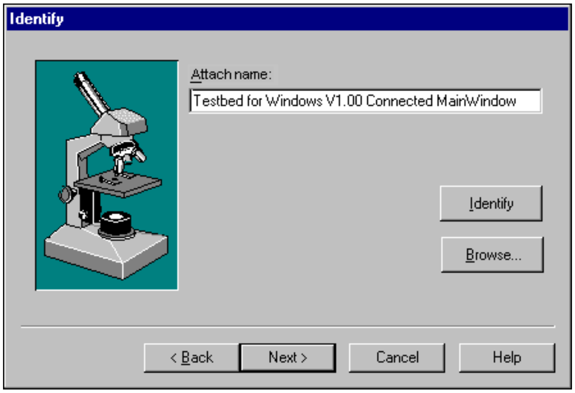
1. When the PROCESS THE DOCUMENTS FOUND screen displays, press the **Insert Event Hotkey**, **{ALT{F7}}**. The Browse Events dialog box displays.

2. Click the **New** button. The New Event dialog box displays.

3. Select the Screen radio button and click **OK**. The Create Screen Event dialog box displays.

4. In the Create Screen Event dialog box, enter an event name such as *FindDoc*.

5. Click the **Next** button. The Identify dialog box displays:



Attach name: Testbed for Windows V1.00 Connected MainWindow

6. Click the **Identify** button. *QA*Run is minimized and Testbed becomes visible.

   Position the pointer over Testbed’s MainWindow and click. *QA*Run is restored and the window name appears in the Attach area of the Identify dialog box.

7. Click **Next**. The Screen Event dialog box displays.

8. Select the Use Rectangle check box and click the **Capture** button. *QA*Run is minimized and Testbed becomes visible.

9. Position the cursor at the top-left portion of the window text “PROCESS THE DOCUMENTS FOUND” and drag the cursor to the bottom-right corner of the text. Release the mouse button and the captured text will display in the text area of the Screen Event dialog box.

10. Click the **Finish** button. The Insert New Event dialog box displays.

11. Click the **Wait** button and the event is inserted into your script as a standard Wait statement.

12. Press the **Arrived at Hotkey**, **{ALT{F9}}**. The Arrived At dialog box displays.

13. Enter the name of the Testbed screen, **Process the Documents Found Screen**, in the Insert Text field.

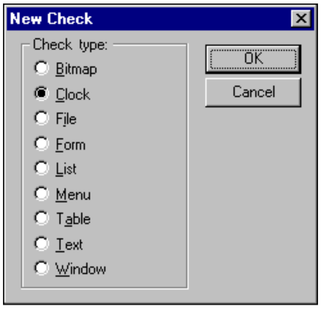
14. Click **OK**. *QA*Run inserts a LogComment statement into your script.

Defining the Clock Check

You must now define a clock check that specifies the expected processing time. Use the following procedure to insert a clock check into your script:

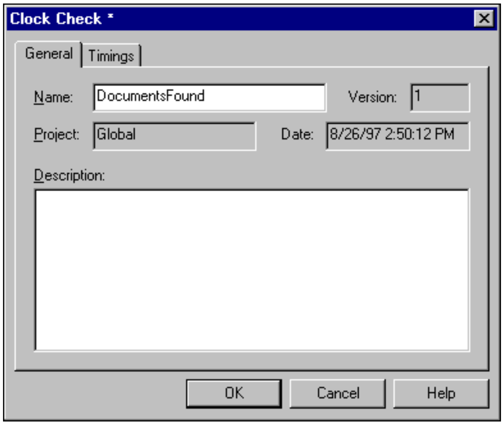
1. With Learn still on, press the **Insert Check Hotkey**, **{ALT{F8}}**. The Browse Checks dialog box displays.

2. Click **New**. The New Check dialog box displays:



Check type:
○ Bitmap
● Clock
○ File
○ Form
○ List
○ Menu
○ Table
○ Text
○ Window

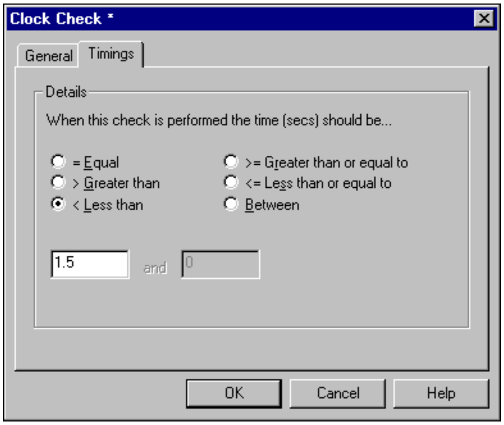
3. Click the Clock radio button and click **OK**. The Clock Check dialog box displays:



Name: DocumentsFound
Version: 1
Project: Global
Date: 8/26/97 2:50:12 PM
Description:

4. Click the General tab and enter a name such as *DocumentsFound*, in the Name field.

5. Click the Timings tab. The following information displays:



Clock Check *

General | Timings

Details

When this check is performed the time (secs) should be...

○ = Equal
○ > Greater than
● < Less than
○ >= Greater than or equal to
○ <= Less than or equal to
○ Between

1.5 and 0

6. Select the Less Than radio button and enter **1.5** in the text area.

7. Click **OK** to save the clock check and insert it into your script. The view of Testbed should be restored.

8. Once the view of Testbed is restored, press the **Esc** key to return to the FIND DOCUMENTS screen.

9. Press the **Learn Hotkey**, **{ALT{F10}}**, to stop Learn.

The Resulting Script

The script you created in this exercise should resemble the following example:

```
Include "Autosync"
Function Main
Call StartAutosync
    Attach "Testbed for Windows V1.00 Connected MainWindow"
        Type "{Tab}{Tab}{Tab}{Tab}{Tab}bank{F2}"
    Wait(0, "", "FindDoc")
    LogComment( "Arrived at Process the Documents Found Screen" )
    Check "DocumentsFound"
        Type "{Esc}"
End Function ; Main
```

“Exercise 4 — Completing the Driver Script” created a new driver script that advanced Testbed to the FIND DOCUMENTS screen. The script in this exercise assumes you are already at the FIND DOCUMENTS screen and continues the exercise by submitting a request to find all records that contain the word “Bank.” It also uses a clock check to verify that the time taken to retrieve the documents is less than 1.5 seconds; however, the full checking mechanism is not quite complete because you haven’t determined when the clock is started or stopped. In the next section, you will complete the check by inserting commands to start and stop the clock at the correct points.

Modifying the Script

The commands required to perform the actions of starting and stopping the script's clock cannot be generated using Learn. You must modify the script to include this functionality. There are three commands associated with clock checks that must be added to the script to make the clock behave like a stopwatch:

- ClockReset( )
- ClockStart( )
- ClockStop( )

These commands must be inserted into the script at the appropriate points. The clock should be reset at the beginning of the script; started after the **F2** action key is pressed; and stopped when the screen Wait is satisfied, but before the LogComment — otherwise the performance value will also include the time it takes to log the comment, which is not part of the test.

Use the following procedure to insert the necessary clock commands into the script at the appropriate positions:

1. Add the following code on a blank line after the `Call StartAutosync` statement at the top of the new script:

   ```
   ClockReset( "DocumentsFound" )
   ```

2. Add the following code on a blank line after the `Type "{Tab}{Tab}{Tab}{Tab}{Tab}bank{F2}"` statement:

   ```
   ClockStart( "DocumentsFound" )
   ```

3. Add the following code on a blank line after the `Wait(0, "", "FindDoc")` statement:

   ```
   ClockStop( "DocumentsFound" )
   ```

Additional Script Modifications

This script is almost ready to run. However, the current script only performs the search operation once. The purpose of the script is actually to repeat the process several times and to record the performance. In order to accomplish this, some extra code must be inserted. The *QA*Run scripting language contains commands that cause scripts to repeat lines of code until a pre-defined condition occurs. The syntax is as follows:

```
Counter = 1
Repeat
   <Instructions>
   Counter = Counter + 1
Until Counter = 5
```

For this script, you need to establish a count variable and continue the repetition until the instructions complete four cycles.

It is also good practice to limit non-essential logging activity during performance testing. The amount of time it takes to write each *QA*Run transaction to the log file affects the results of the clock check. There are two ways to alter the logging activity. You can change logging using a filter when the script runs; however, these run environment filters affect the entire script. Or, as in this exercise, you can apply selective filtering — to turn off only the non-essential log items — while still logging most commands and check results. You can selectively filter logging activity by inserting system variables at specific points in the script.

In this script, you want to suspend logging for commands and comments at specific points. The syntax is as follows:

```
Log.Commands = 0
Log.Comments = 0
```

Where the state is `1` (on) or `0` (off).

Use the following example script as a guide when modifying your test script to include a repeat loop and suspend logging activity. Changes to the script are indicated in bold typeface.

```
Include "Autosync"
Function Main
Call StartAutosync
Counter = 1
Repeat
   Log.Commands = 0
   Log.Comments = 0
   ClockReset( "DocumentsFound" )
   Attach "Testbed for Windows V1.00 Connected MainWindow"
      Type "{Tab}{Tab}{Tab}{Tab}{Tab}bank{F2}"
   ClockStart( "DocumentsFound" )
```

```
    Wait(0, "", "FindDoc")
    ClockStop( "DocumentsFound" )
    LogComment( "Arrived at Process the Documents Found Screen" )
    Log.Commands = 1
    Log.Comments = 1
    Check "DocumentsFound"
    Type "{Esc}"
    Counter = Counter + 1
Until Counter = 5
End Function ; Main
```

The above script will now repeat the instructions four times, carrying out a clock check each time.

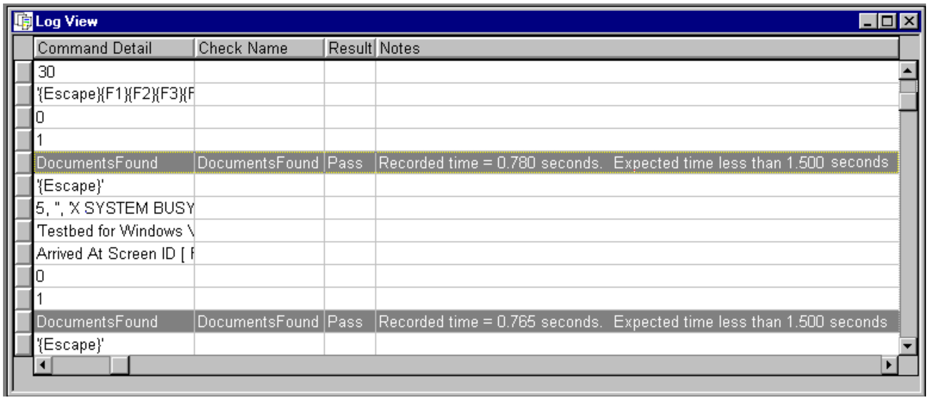
Running the Script

1. Click the **Run Script** button from the Script Editor's toolbar. The Summary Info dialog box displays.
2. Enter a name such as *FINDDOCS* in the Name area, complete a brief description of the script, and click **OK**. The Run Script dialog box displays.
3. Click the **OK** button to run the script.

Analyzing the Results

After you run the script, Log View automatically displays the script results. The Log View is configurable, and your results may display different columns than those shown in Figure 3-2.

You may change the column layout by selecting Grid>Column Layout from the View menu. Your log should resemble the following example:



Log View

Command Detail	Check Name	Result	Notes
30			
'{Escape}{F1}{F2}{F3}{F			
0			
1			
DocumentsFound	DocumentsFound	Pass	Recorded time = 0.780 seconds. Expected time less than 1.500 seconds
'{Escape}'			
5, '', 'X SYSTEM BUSY			
'Testbed for Windows \			
Arrived At Screen ID [F			
0			
1			
DocumentsFound	DocumentsFound	Pass	Recorded time = 0.765 seconds. Expected time less than 1.500 seconds
'{Escape}'			

Figure 3-2. Viewing Clock Check Results

Scroll through the log file and view the results of each individual clock check. By default, checks that pass appear in green text, and checks that fail appear in red text in the log. The Notes column contains specific information related to the expected and actual values of the clock check.

It is possible to gain a quick overview of how your target application performed by displaying a summary of your check results or the full Log View using the following toolbar buttons:

- Click the **Test Run Statistics** button to view test statistics.
- Click the **Check Statistics** button to view check statistics.
- Click the **Full Display** button to return to the full Log View display.

Exercise Summary

This exercise focused on checking the performance of an application's function using clock checks. After completing this exercise, you should be able to successfully:

- Create a clock check
- Initialize *QA*Run's internal clock
- Start and stop *QA*Run stopwatch using appropriately placed Clock( ) commands
- Create a counting mechanism for Repeat loops
- Suspend logging activity for specific commands and comments
- Identify transaction cycles for performance measurements
- View the expected and actual results of a performance check
- View clock check statistics.

Exercise 7 — Using External TestData Files

An important testing capability is the ability to enter variable data into the target application — to test the application's behavior with different input or to test behavior under heavy load conditions. *QA*Run is able to enter external data into the target application using an external testdata file. This exercise will build a test script that carries out volume testing using an independent data file to enter information into Testbed.

Getting Started

In “Exercise 3 — Synchronizing Scripts Using System Variables”, the driver script advanced Testbed to the CUSTOMER MASTER MAINTENANCE MENU screen by pressing the **F1** key from Testbed's MAIN MENU. The test script created in this exercise will add a customer record to Testbed and update the database. The script will use the Repeat command in conjunction with *QA*Run's system variables to loop several times and enter a new customer record each time.

Exercise Prerequisites: Before beginning this exercise, you must have completed the following prerequisites:

- Access to the script named *AUTOSYNC* created during “Exercise 3 — Synchronizing Scripts Using System Variables”
- Access to the Event named *Autosync* created during “Exercise 3 — Synchronizing Scripts Using System Variables”
- Access to the CustChar.csv file (by default located in *QA*Run's Data directory)
- Include *Autosync* script and Call statement in the default script (see page 2-45)
- Close Testbed
- Close *QA*Run.

Testing Requirements: The following test elements are created during this exercise:

- Screen event named *RecordUpdate*
- Script named *AddCust*.

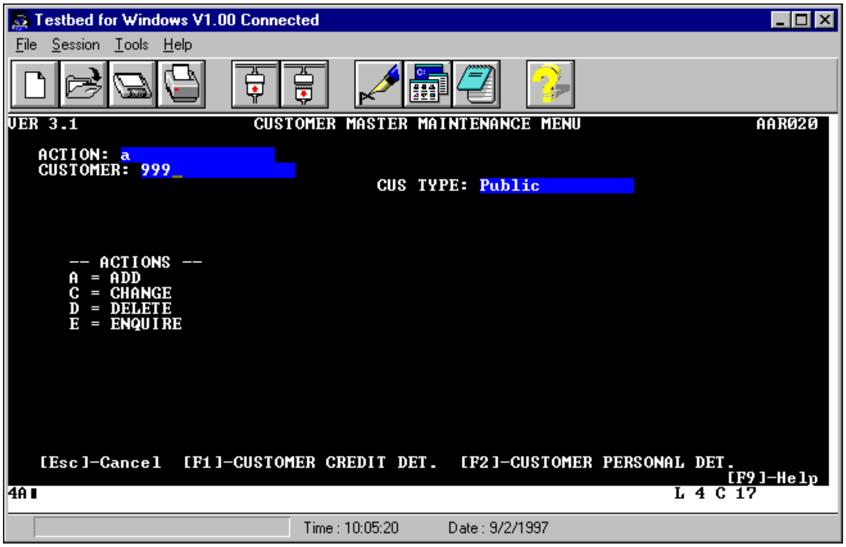
You should allow 20–30 minutes to complete this exercise.

Learning the Script

Use the following procedure to begin learning the new test script:

1. Start a new *QA*Run script (ensure the default script information contains the Include statement for the synchronization logic).
2. Start Testbed using the same method described in previous exercises and click the **Connect** button on the toolbar.
3. Logon to Testbed and select **F1** from the Testbed's MAIN MENU screen.

   Testbed should now be at the CUSTOMER MASTER MAINTENANCE MENU screen. This is the point where the test script begins:



```
VER 3.1                    CUSTOMER MASTER MAINTENANCE MENU                AAR020

   ACTION: a
   CUSTOMER: 999_
                                   CUS TYPE: Public



      -- ACTIONS --
      A = ADD
      C = CHANGE
      D = DELETE
      E = ENQUIRE



   [Esc]-Cancel  [F1]-CUSTOMER CREDIT DET.  [F2]-CUSTOMER PERSONAL DET.
                                                                  [F9]-Help
4A▮                                                          L 4 C 17
```

4. Pressing the **Learn Hotkey**, **{ALT{F10}}**, to start Learn.
5. Enter the following data on the CUSTOMER MASTER MAINTENANCE MENU screen:

   a. Type **a** in the ACTION field to add a customer and press the **Tab** key to move to the next field.

b. Type **999** in the CUSTOMER NUMBER field.

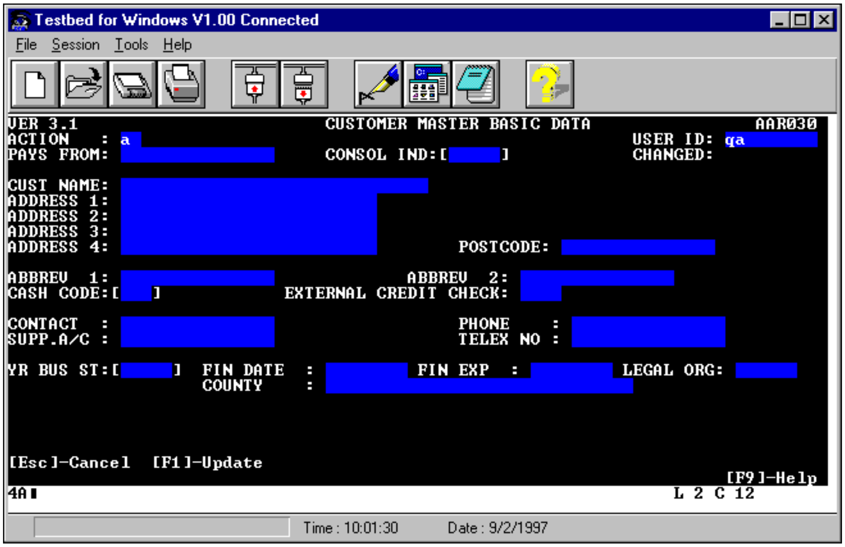
Note

Testbed only allows a customer to be added to the database once. If you attempt to add customer 999 again (by running the script or otherwise), then the script will fail.

To avoid this, delete the testbed.dat file, make a copy of the Testbed.sav file, and rename it testbed.dat each time you run the script (these files are located in *QA*Run's Demos directory by default). This resets the database and allows you to reenter customers.

6. Press **F2** to view the personal details for customer 999.

The CUSTOMER MASTER BASIC DATA screen displays. All fields except the ACTION field should be blank:



```
VER 3.1                                  CUSTOMER MASTER BASIC DATA                  AAR030
ACTION    : a                                                            USER ID: qa
PAYS FROM:                               CONSOL IND:[        ]          CHANGED:

CUST NAME:
ADDRESS 1:
ADDRESS 2:
ADDRESS 3:
ADDRESS 4:                                            POSTCODE:

ABBREV   1:                                       ABBREV   2:
CASH CODE:[   ]                         EXTERNAL CREDIT CHECK:

CONTACT   :                                           PHONE    :
SUPP.A/C  :                                           TELEX NO :

YR BUS ST:[     ]  FIN DATE   :            FIN EXP   :            LEGAL ORG:
                   COUNTY     :



[Esc]-Cancel   [F1]-Update
                                                                               [F9]-Help
4A▮                                                                       L 2 C 12
```

Time : 10:01:30 Date : 9/2/1997

7. Press the **Tab** key to move to the PAYS FROM FIELD, type **Boston** in the field, and press the **Tab** key.

8. Enter a name and address in the CUSTOMER and ADDRESS fields (you can use the **Tab** key to move between fields).

9. Complete the remaining fields on the CUSTOMER MASTER BASIC DATA screen with the following information:

a. In the ABBREV 1 field, enter your own family name.

b. In the CASH CODE field, enter a two-number code such as **12**.

c. In the CONTACT field, enter a colleague's name.

d. In the SUPP A/C field, enter a numeric value such as **23**.

- **e.** In the YR BUS ST field, enter a numeric value such as **99**.
- **f.** In the CONSOLE IND field, enter a numeric value such as **18**.
- **g.** In the POSTCODE field, enter your zip code.
- **h.** In the EXTERNAL CREDIT CHECK field, enter **Y**.
- **i.** In the PHONE field, enter a phone number.
- **j.** In the TELEX NO field, enter a different number.

This concludes the entry of the new customer record.

10. Press the **F1** key to add the customer record to the database.

    Testbed displays the message “Customer Record Updated” in the bottom-left corner of the screen. This message is an indicator that Testbed has finished processing the record.

    You must define a screen event to tell *QA*Run to wait until this message displays before issuing further instructions.

11. Press the **Insert Event Hotkey**, **{ALT{F7}}**, and define a screen event that waits for the message to appear. Enter a name such as *RecordUpdate* in the Name field and insert the event as a Wait statement.

12. Press **Escape** to return to the CUSTOMER MASTER MAINTENANCE MENU screen and stop Learn.

Understanding Data-Driven Scripts

So far, this script enters one record — the one you entered. To effectively use an external testdata file for volume testing, the script must now be modified to loop so it enters a new record each time. *QA*Run can do this using an external testdata file. This script will eventually use the information from the testdata file to enter several customer records.

TestData Files

*QA*Run can read external data files in CSV (Comma Separated Variable) format. Many spreadsheet and database applications are able to export data in this file format, so you can use data from these applications within your *QA*Run scripts.

Because these data files often contain varying amounts of records and fields, *QA*Run must employ a mechanism for determining where each record starts and ends. To accomplish this, *QA*Run uses a combination of system variables and commands to read the contents of the data file. *QA*Run automatically creates its own index to ensure that individual data items can be quickly accessed.

A testdata file is a CSV file in which each line constitutes a record. Each record contains fields that are separated by commas. Consider the following testdata file. Here, there are three records, each with six fields exported from a database application as a CSV file:

```
J. Smith,The Mall,London,TW7 4DS,UK,8471666
J. Cotez,Ave St Maria,Madrid,987621,Spain,98762301
A. Gilbert,Rue Albert,Issigeac,24679,France,53234512
```

Notice that there should not be any spaces between the fields and their comma separators. Fields containing commas can be included within the testdata file if they are enclosed in double quotes. For example:

```
Compuware Ltd,"163, Bath Road","Slough, SL1 4AA"
```

*QA*Run can read the contents of a testdata file. To tell *QA*Run which testdata file to use in a script, you set the TestData( ) command to the name of the file, for example:

```
TestData( "CustChar.CSV" )  ;Use default QARun Directory
```

or:

```
TestData( "C:\TESTDATA\CustCar.CSV" );Specify directory
```

*QA*Run does not use the original CSV file, but instead generates a special indexed file — custchar.INX. This special index file is automatically generated if it does not exist or if the date does not match the CSV data file date. The following is an example index file:

```
0 0       Field 1    Field 2      Field 3     .......
Record 1  J. Smith,   The Mall,     London,     .......
Record 2  J. Cotez,   Ave St Maria, Madrid,     .......
Record 3  A. Gilbert, Rue Albert,   Issigeac,   .......
```

*QA*Run's record pointer is automatically positioned just before the first field at position 0 0. The first record in the file is record 1, and the first field of each record is field 1.

Every time you change the testdata file name by setting the TestData( ) command, the current field and record markers are “set” to 0, even if you use the same testdata file name.

The information in the indexed file is retrieved using the Type command. The Type command is used to access each record and field in turn. The Type syntax is:

```
Type "{Record.Field}"
```

For example, if you entered the following Type commands into your script:

```
Type "{1.1}{Return}"        ; First Record - First Field
Type "{1.2}{Return}"        ; First Record - Second Field
```

The results would be:

```
J. Smith
The Mall
```

The following wildcard characters may be used instead of explicit record/field references:

=	Same record or field
+	Next record or field
-	Previous record or field
*	Random record or field

Therefore:

```
Type "{=.6}"      ; Selects 6th field of the current record
Type "{+.4}"      ; Selects 4th field of the next record
Type "{=.+}"      ; Selects next field of the current record
Type "{*.5}"      ; Selects 5th field of any record
Type "{*.*}"      ; Selects any field from any record
```

Looping

Two commands, TestDataFieldCount( ) and TestDataRecordCount( ), tell you how many records and fields per record exist in the current data file.

Similarly, the TestDataCurField( ) and TestDataCurRecord( ) commands identify the record and field that is currently selected.

Together, these four commands allow you to build loops that can access each record and field in the testdata file. The following exercise discusses the script modifications that are required to instruct the script to access the testdata file CustChar.csv. Once accessed, *QA*Run will enter every item in the testdata file into Testbed's database as instructed.

Modifying the Script

It is now necessary to change your Testbed script so that it uses the testdata and looping functionality described in the previous sections.

1. Insert the following code on a blank line following the `Function Main` statement:

   ```
   TestData("CustChar.csv")
   ```

   The complete directory path has not been included because the file is located in the default *QA*Run Data directory. If you installed *QA*Run in another directory, then you must include the complete path for the file's location.

2. Insert the following code on a blank line following the `Call StartAutosync` statement:

   ```
   Repeat
   ```

   The Repeat command inserted at this position begins the primary loop that tells *QA*Run to process each record, one after the other. You also need to tell *QA*Run when to stop processing records, or how many records to process. You will use the Until command to accomplish this.

3. Insert the following code on a blank line following the `Type "{Esc}"` statement:

```
    Until TestDataCurRecord = TestDataRecordCount
```

This statement tells *QA*Run to continue processing records until the testdata record that is currently being processed is the same record as the last record in the testdata file.

The Until command completes the primary loop (the loop that tells *QA*Run to move from record-to-record); however, we still need to insert a secondary loop that provides the instructions that tell *QA*Run to retrieve the next field in the current record and enter it on the CUSTOMER BASIC MASTER DATA screen; however, you must first replace the customer information that we entered manually with the correct syntax that will automatically pull information from the .csv file.

4. Locate the section of the script the contains the Attach command and related Type commands. This section of your script should resemble the following information:

```
Attach "Testbed for Windows V1.00 Connected MainWindow"
    Type "a{Tab}999{F2}{Tab}Boston{Tab}Customer Name...
```

This is the section of your script that manually adds the customer's information to Testbed's database. It contains the details you typed in when learning the script; however, you no longer want to manually enter the customer's data. You now want to automate the process using an external testdata file. You will need to replace the manual instructions with syntax that tells *QA*Run the appropriate fields and records to use from the testdata file.

5. Replace the Type commands with the following code (comments have been added to assist you):

```
    Type "a{Tab}{+.1}" ; Next Record First field
    Type "{F2}"        ; CUSTOMER PERSONAL DET Option
      Type "{Tab}"     ; Move to PAYS FROM field
      Type "{=.+}"     ; Get the next field from the same record
    Type "{F1}"        ; Update the record
```

This tells *QA*Run to use the testdata file to complete the customer information, but this code is really only completing one field and then attempting to update the record. We need to insert a secondary loop that tells *QA*Run to keep entering data into the fields until each field is entered, and then update the record.

6. Insert the following code on a blank line following the `Type "{F2}"` statement:

```
    Repeat
```

This Repeat command begins the secondary loop that tells *QA*Run to process the next field in the current record and enter it on the CUSTOMER BASIC MASTER DATA screen.

We now need to tell *QA*Run when to stop processing the next field and move to the next record.

7. Insert the following code on a blank line before the `Type "{F1}"` statement:

```
Until TestDataCurField = TestDataFieldCount
```

This statement tells *QA*Run to continue processing fields until the testdata field that is currently being processed is the field that is the last field in the current testdata record.

8. Save the script with a name such as *AddCust*.

The Resulting Script

After completing the above steps, your test script should look very similar to the following example. The code associated with the Loop and Type commands is highlighted in bold typeface:

```
Function Main
TestData("CustChar.csv")
Call StartAutosync
Repeat
Attach "Testbed for Windows V1.00 Connected MainWindow"
   Type "a{Tab}{+.1}"
   Type "{F2}"
      Repeat
       Type "{Tab}"
       Type "{=.+}"
      Until TestDataCurField = TestDataFieldCount
   Type "{F1}"
   Wait(0, "", "RecordUpdated")
   Type "{Esc}"
Until TestDataCurRecord = TestDataRecordCount
End Function ; Main
```

Exercise Summary

This exercise focused on using an external data file to automatically supply information to a script, rather than typing the information in manually. After completing this exercise, you should be able to successfully:

- Call an external testdata file to retrieve data records.
- Use the Type command to extract field and record data from a testdata file.
- Use the Repeat command to repeatedly enter data from a testdata file.

Exercise 8 — Inserting Bitmap Checks

The purpose of this exercise is to build a simple test script for regression testing. This test script will use a bitmap check to regression test. In "Exercise 4 — Completing the Driver Script", the driver script advanced Testbed to the FIND DOCUMENTS screen by pressing **F4** from the Testbed's MAIN MENU. In this exercise, you will create a script that performs a bitmap check against the FIND DOCUMENTS screen's toolbar and status bar.

Bitmap checks compare the actual bitmaps with previously defined bitmaps. These checks are often used to verify the appearance of toolbars, the desktop, and other windows that contain non-textual information. When you create a bitmap check, you capture the image within a rectangular area of the screen. When the check is verified, the same area is captured and compared to the defined image. If the two match exactly, the check passes. If they are different, the check fails.

Getting Started

In the following exercise, you will create a bitmap check and insert it into a new script. You will also once again include the *Autosync* script created in "Exercise 3 — Synchronizing Scripts Using System Variables" to synchronize the Testbed screens. At the end of the exercise you will save the script without running it.

Exercise Prerequisites: Before beginning this exercise, you must have completed the following prerequisites:

- Access to the script named *AUTOSYNC* created during "Exercise 3 — Synchronizing Scripts Using System Variables"
- Access to the Event named *Autosync* created during "Exercise 3 — Synchronizing Scripts Using System Variables"
- Include *Autosync* script and Call statement in the default script (see page 2-45)
- Close Testbed
- Close *QA*Run.

Testing Requirements: The following test elements are created during this exercise:

- Bitmap check named *Find Documents Bitmap*
- Script named *Version.*

You should allow 15–20 minutes to complete this exercise.

Learning the Script

Before you begin creating the test script, use the following procedure to start *QA*Run and navigate Testbed to the correct screen where the test script will start.

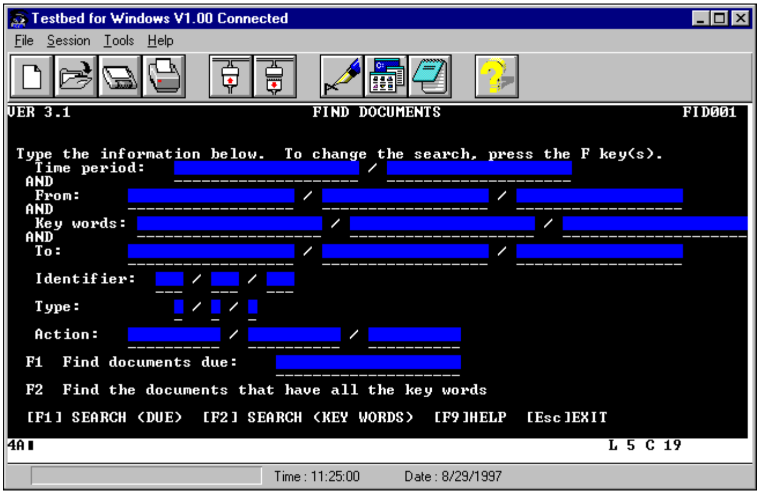
1. Start *QA*Run and click the **New** button to create a new script (ensure the default script contains the Include statement for the synchronization logic).

2. Start Testbed and click the **Connect** button on the toolbar.

3. Logon to Testbed.
   a. Type **CW** in the UserID field and press **Tab.**

   b. Type **PASS** in the Password field and press **Enter**.

4. Press the **F4** (Search) key. Testbed displays the FIND DOCUMENTS screen:



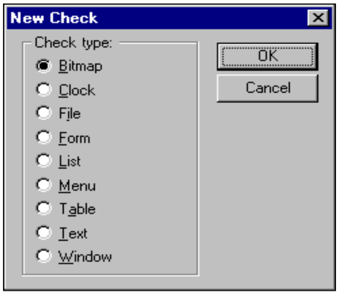
```
VER 3.1                          FIND DOCUMENTS                         FID001


 Type the information below.  To change the search, press the F key(s).
   Time period:                      /
  AND
   From:                      /                  /
  AND
   Key words:                  /                  /
  AND
   To:                        /                  /

   Identifier:     /     /

   Type:            /  /

   Action:                  /          /

  F1  Find documents due:

  F2  Find the documents that have all the key words

  [F1] SEARCH (DUE)  [F2] SEARCH (KEY WORDS)  [F9]HELP  [Esc]EXIT
4A▮                                                             L 5 C 19
```

Time : 11:25:00   Date : 8/29/1997

5. Press the **Learn Hotkey**, **{ALT{F10}}**, to start Learn.

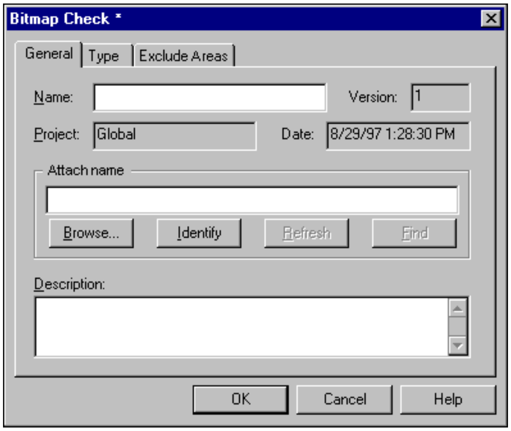
6. Insert a check by pressing the **Insert Check Hotkey**, **{ALT{F8}}**. The Browse Checks dialog box displays.

7. Click the **New** button. The New Check dialog box displays:



Check type:
● Bitmap
○ Clock
○ File
○ Form
○ List
○ Menu
○ Table
○ Text
○ Window

8. Select the Bitmap radio button and click **OK**. The Bitmap Check dialog box displays:



Bitmap Check *

Name:

Version: 1

Project: Global

Date: 8/29/97 1:28:30 PM

Attach name

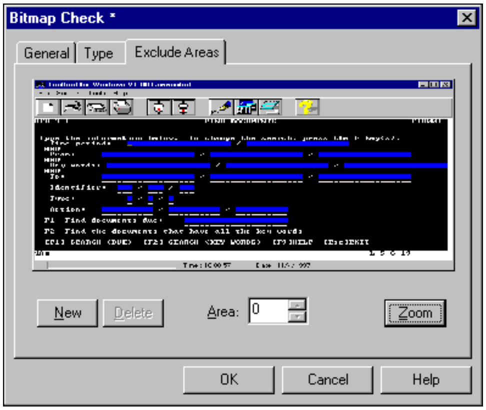
Description:

9. In the Name field, enter a name such as *Find Document Bitmap* for the new bitmap check.

10. Click the **Identify** button. *QA*Run is minimized and Testbed becomes visible.

    Position the pointer over Testbed's MainWindow and click. *QA*Run is restored and the window name appears in the Attach name area of the Bitmap Check dialog box.

11. Click the Exclude Areas tab. The tab displays as shown below:



Bitmap Check *

Area: 0

12. Click the **Zoom** button or double-click on the bitmap image to view an enlarged view of the bitmap in the Captured Image window.

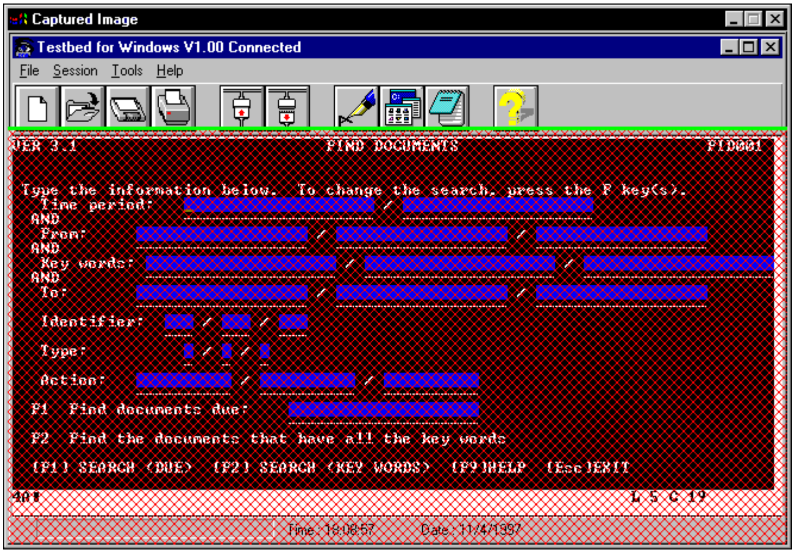
    This enlarged bitmap view makes it easier to mask image areas to Exclude, but the Bitmap Check dialog box is still open behind the Captured Image window.

13. Click the Bitmap Check dialog box and click the **New** button on the Exclude Areas tab. The Bitmap Check dialog box minimizes and the Captured Image window becomes active. The cursor changes to a cross-hair.

14. In the Captured Image window, drag the cursor over the portion of the image that contains Testbed's FIND DOCUMENTS screen. Be sure exclude the toolbar from your selection.

15. Release the mouse button. Everything beneath the toolbar should be covered by crosshatches and the Bitmap Check dialog box reappears.

    When the check is run, the toolbar will be the only part of the image considered for comparison. The Captured Image window should now resemble the following image:



```
VER 3.1                         FIND DOCUMENTS                        FID001

 Type the information below.  To change the search, press the F key(s).
   Time period:                  /
  AND
   From:                  /                  /
  AND
   Key words:              /                  /
  AND
   To:                    /                  /

   Identifier:      /     /

   Type:              /  /

   Action:         /         /

  F1   Find documents due:

  F2   Find the documents that have all the key words

  (F1) SEARCH (DUE)   (F2) SEARCH (KEY WORDS)   (F9)HELP   (Esc)EXIT
4A1                                                          L 5 C 19
                     Time : 18:08:57        Date : 11/4/1997
```

The FIND DOCUMENT screen area (covered with crosshatches) will not be included in the check.

16. Click the **OK** button on the Bitmap Check dialog box to insert the check into the script

17. Turn Learn off.

The Resulting Script

After completing the above steps, your test script should look very similar to the following example. The code associated with the bitmap check is highlighted in bold typeface:

```
Include "Autosync"
Function Main
Call StartAutosync
   Check "Find Document Bitmap"
End Function ;Main
```

This script is not intended to perform any action other than the bitmap check.

Saving the Script

From the **File** menu, choose **Save As** to save the script. Assign a name such as *Version*. This is a simple test script that will eventually be called by a driver script, so we do not need to run the script at this time.

Exercise Summary

This exercise focused on creating a test script that uses a bitmap check to compare images for regression testing. After completing this exercise, you should be able to successfully:

- Create a bitmap check that looks for specific information on the screen
- Exclude specific areas of a bitmap from a check
- Save a script without running it.

In the next chapter, you'll complete an exercise that includes each individual test script into the driver script — thereby completing the test suite.

Exercise 9 — Inserting Script Dialog Boxes

*QA*Run contains a Dialog Editor that can be accessed from the Script Editor. The purpose of the Dialog Editor is to allow you to design dialogs that will be displayed when you run your scripts. Dialogs allow you to enter variable or confidential data into *QA*Run as it is running, rather than hard-coding it into a script. This is most useful for entering password or userID information.

Dialogs are created graphically using the tools provided by the Dialog Editor. These graphical representations can then be inserted into the Script Editor as a function. The Dialog function can be called by the script at any time and can be used by any number of scripts. If you need to change the dialog's definition, you can load the image into the Dialog Editor to make the modifications. After you make the changes, all scripts that call the dialog will automatically call the modified version. There is no need to change the function definition within the script (unless you must add logic to process new controls).

Information entered into the dialogs — from pushing a button to entering passwords — can then be processed by the script at runtime.

The Dialog Editor offers all of the standard controls for designing Windows-compliant dialogs.

Getting Started

The driver script built during "Exercise 1 — Creating a Driver Script" logs on to Testbed and drives the application. The User ID and Password are hard-coded into this script. The following exercise explains how to build a dialog that asks for an ID and password at runtime. It will also ask which version of Testbed to run. The information entered into the dialog will be transferred to user-defined variables within the script and replace the hard-coded information.

The resulting dialog should resemble the example below:

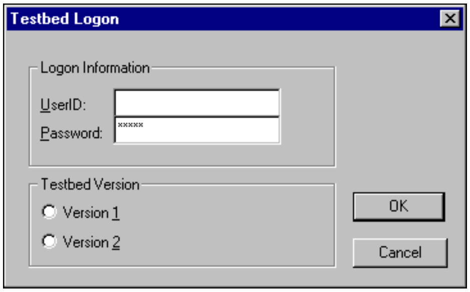


Figure 3-3. Sample Dialog Box

Exercise Prerequisites: Before beginning this exercise, you must have completed the following prerequisites:

- Access to the script named *Driver2* created in “Exercise 4 — Completing the Driver Script”.
- Access to the event named *VM/System Product Screen* created in “Exercise 2 — Implementing Basic Synchronization”
- Close Testbed
- Close *QA*Run.

Testing Requirements: The following test elements are created during this exercise:

- Dialog named *Start Script Dialog*

You should allow 20–30 minutes to complete this exercise.

Creating the Dialog

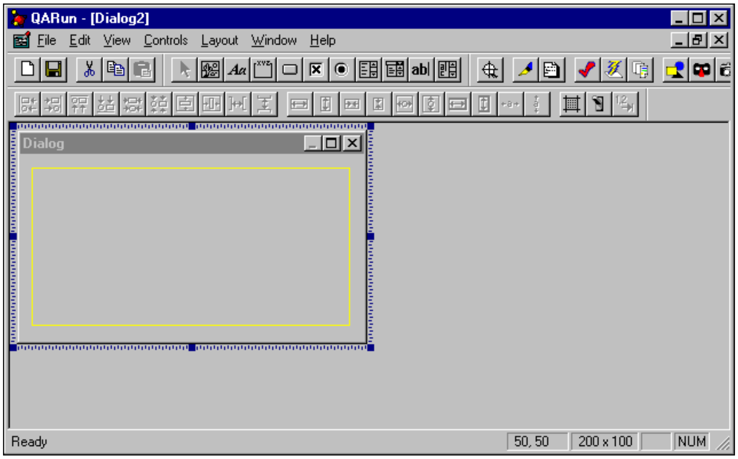
Use the following procedure to create a user-defined dialog box from the Dialog Editor:

1. Start *QA*Run and load the *Driver2* script.

2. Position the cursor on a blank line following the `Call StartAutosync` command.

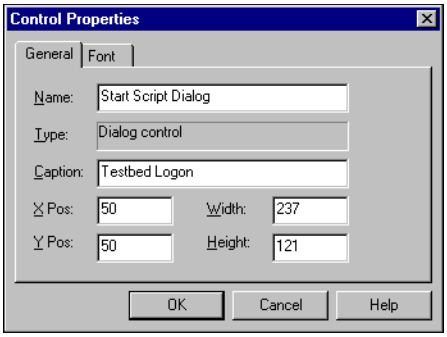
3. Click the **Dialogs** Button from the Script Editor’s toolbar to access the Browse Dialogs table. The Browse Dialogs dialog box displays.

4. Click the **New** button to define a new dialog. The Dialog Editor displays with a blank dialog definition as shown below:



The Dialog Editor window contains two additional toolbars to assist you with creating the dialog boxes. The window also displays a blank dialog box with active control handles.

5. Double-click on the blank dialog box to define its properties. The Control properties dialog box displays:



Name: Start Script Dialog

Type: Dialog control

Caption: Testbed Logon

X Pos: 50

Width: 237

Y Pos: 50

Height: 121

6. Complete the Name field with a dialog name such as *Start Script Dialog*. This name will be used within the script by the function definition.

7. Enter **Testbed Logon** in the Caption field. This text will appear in the title bar of the new dialog box.

8. Click **OK**.

9. Using Figure 3-3 on page 3-36 as a guide, resize the dialog by clicking on the right-bottom corner handle and dragging it to size the dialog.

Adding Dialog Controls

You can add controls to the dialog definition using a point-and-click principal. There are two extra tool bars in the Dialog Editor: the Controls toolbar and the Layout toolbar. You will use the Controls toolbar (Figure 3-4) to select the type of control needed:

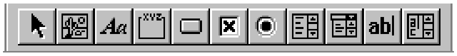


Figure 3-4. Dialog Editor Controls Toolbar

Adding Static Text Controls

Static text controls are used to provide labels for edit controls, list boxes, and combo boxes which, unlike radio buttons and check boxes, do not have a text caption of their own. Use the following procedure to add static text controls to the dialog box:

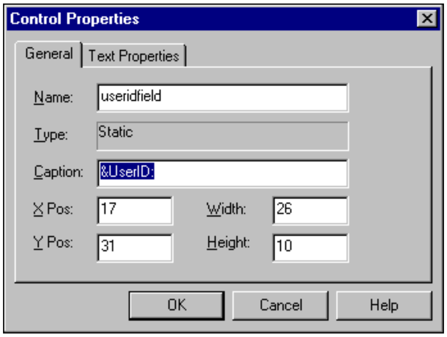
Aa

1. Click the **Text Control** button to select it and move the cursor over the dialog.

2. Click the left side of the dialog to drop the control.

   The control can be moved and resized using the grab handles.

3. Double-click on the text control to define its properties. The Control Properties dialog box displays:



Name: useridfield

Type: Static

Caption: &UserID:

X Pos: 17 Width: 26

Y Pos: 31 Height: 10

4. Enter **useridfield** in the Name field.

   The Name field contains the variable name that will be used to process the contents of the field. You can change it to a meaningful name to make referencing it easier.

5. Enter **&UserID:** in the Caption field. This is the actual text that will appear on the dialog. The ampersand (&) before the "U" denotes this as an underlined character.

6. Click the Text Properties tab and clear the Border and No Prefix options.

   The Border option puts a border around the text, and the No Prefix option switches off the underlined character option (you can press the **F1** key to receive an online help description of all Properties options).

7. Click **OK**.

8. Repeat steps 1–7 to create another text control. Enter **PasswordField** in the Name field and enter **&Password:** in the Caption field.

Aligning the Controls

You need to align the two text controls that you just created using Figure 3-3 on page 3-36 as a guide. The Layout toolbar allows you to format and align controls after placing them on the dialog box. Use the Layout toolbar (Figure 3-5) to arrange the controls:

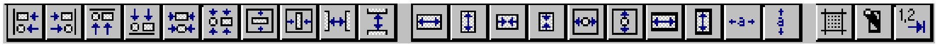


Figure 3-5. Dialog Editor Layout Toolbar

Use the following procedure to align the text controls:

1. Highlight both text controls by clicking the first control, pressing the **Shift** key, and clicking the second text control.

   The second control has a bolder highlight, indicating that this control is primary, and the alignment modification will be made according to this control's position.

2. Click the **Align Left** button to align the controls to the extreme left of the primary control.

Adding Edit Controls

Edit controls are used to receive user input that will be processed by the script at runtime. Edit controls are useful for inputting User IDs, passwords, and other variable information.

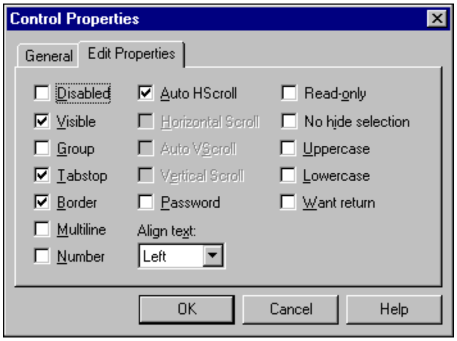
Use the following procedure to insert an edit control:

ab|

1. Click the **Edit Control** button.
2. Position the pointer over the dialog box and click on the right side of the User ID static text control to drop the control.

   The control can be moved and resized using the grab handles.
3. Double-click on the edit control to display the Control Properties dialog box.

   Unlike the static text control, the Name of this control will be used by the script at runtime and should be changed to something more meaningful.
4. Enter **userid** in the Name field. Remember, variable names are case-sensitive.
5. Click the Edit Properties tab. The following information displays:



☐ Disabled
☑ Visible
☐ Group
☑ Tabstop
☑ Border
☐ Multiline
☐ Number

☑ Auto HScroll
☐ Horizontal Scroll
☐ Auto VScroll
☐ Vertical Scroll
☐ Password
Align text: Left

☐ Read-only
☐ No hide selection
☐ Uppercase
☐ Lowercase
☐ Want return

6. Ensure that the Visible and the Tabstop options are selected. The Tabstop option allows the user to move to this control using the **TAB** key instead of the mouse.
7. Click **OK**.
8. Repeat the above steps to create a second edit control for the Password option.
   a. Enter **password** in the Name field.
   b. Click the Edit Properties tab and ensure that the Visible, Tabstop, and Password check boxes are all selected. The Password option displays the contents of the control as a series of asterisks rather than visible text.
   c. Click **OK**.
9. Align the controls in the same way as the static text controls (see “Aligning the Controls” on page 3-39).

Adding Group Boxes

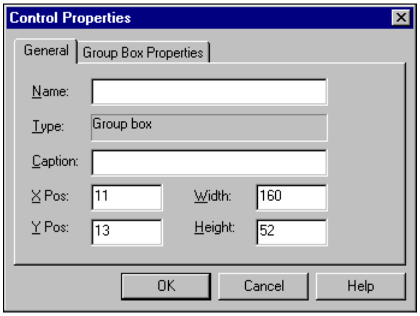
Group Boxes are used to provide a visual grouping of related controls. Use the following procedure to add a group box to the dialog:

1. Click the **Add Group Box** button on the toolbar and click on the right side of the dialog definition.

   A rectangle with the words “Group Box” displays on the dialog. You can move the box by clicking inside the box and dragging. In each corner of the box is a size handle. Dragging these handles will change the size of the box.

2. Using Figure 3-3 on page 3-36 as a guide, size the group box so that there is adequate space to contain the UserID and Password fields. Don’t worry if you get the size of the box wrong. It can always be resized later.

3. Double-click on the group box to define its control properties. The Control Properties dialog box displays:



Control Properties

General | Group Box Properties

Name:

Type: Group box

Caption:

X Pos: 11 Width: 160

Y Pos: 13 Height: 52

Hint

Ensure that you’ve double-clicked the Group Box and the Group Box Control Properties dialog box displays. It’s easy to accidently select the dialog box’s Control Properties.

4. Enter **idBox** in the Name field.

5. Enter **Logon Information** in the Caption field. This changes the caption or label on the group box at the top of the rectangle.

6. Click **OK**.

7. Repeat the above steps to create a second group box to eventually contain the Testbed version radio buttons:

   a. Enter **VersionBox** in the Name field.

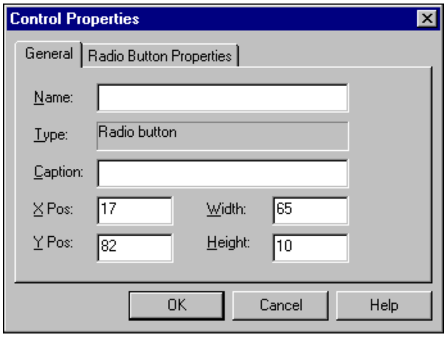
   b. Enter **Testbed Version** in the Caption field. This changes the caption or label on the group box at the top of the rectangle.

   c. Click **OK**.

Adding Radio Buttons

A radio button is a user option that is exclusive. This means that only one option per group may be selected. Use the following procedure to add radio buttons to the dialog box:

1. Click the **Radio Button Control** button on the toolbar.
2. Move the pointer into the Testbed Version group box and single-click to drop the radio button. A radio button with the words “Radio Button” should appear on the dialog box.
3. Double-click on the button to edit its control properties. The Control Properties dialog box displays:



Name:

Type: Radio button

Caption:

X Pos: 17 Width: 65

Y Pos: 82 Height: 10

4. Enter **Version1** in the Name field. Remember, variable names are case-sensitive.
5. Enter **Version &1** in the Caption field. The ampersand indicates which character has the underscore.
6. Click the Radio Button Properties tab and ensure that Visible, Tabstop, and Auto check boxes are selected. The Auto option ensures that the radio buttons contain a black fill when selected.
7. Click **OK**.
8. Repeat the above steps to create a second radio button:
   a. Enter **Version2** in the Name field.
   b. Enter **Version &2** the Caption field. The ampersand indicates which character has the underscore.
   c. Click **OK**.
9. Align both radio buttons using the same technique you used to align the static text controls (see “Aligning the Controls” on page 3-39).

Adding Push Buttons

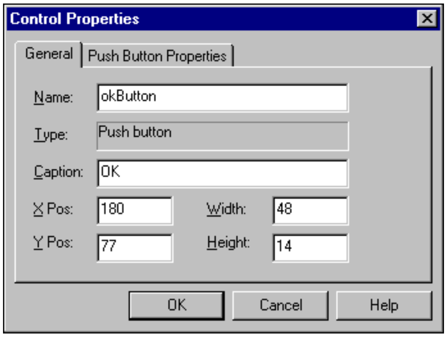
A push button, also known as a command button, is generally used to dismiss a dialog box (for example, **OK** and **Cancel** buttons). Use the following procedure to add two push buttons to the dialog box.

1. Select the **Push Button** control from the toolbar.

2. Move the pointer to the bottom of the dialog box and single-click to drop the push button.

   A push button with the words "Push Button" should appear on the dialog. This control can be sized using the size handles.

3. Double-click on the control to edit its control properties. The Control Properties dialog box displays:



Name: okButton

Type: Push button

Caption: OK

X Pos: 180 Width: 48

Y Pos: 77 Height: 14

4. Enter **okButton** in the Name field.

5. Enter **OK** in the Caption field. This is the name that will actually appear on the button.

6. Click the Push Button Properties tab and ensure that Visible, Tabstop, and Default check boxes are selected. The Default check box indicates that this button will be automatically selected if the **Enter** key is pressed.

7. Click the Control Properties dialog box's **OK** button.

8. Repeat the above steps to create a second push button:

   a. Enter **cancelButton** in the Name field.

   b. Enter **Cancel** in the Caption field.

   c. Click **OK**.

9. Align both push buttons using the same technique used to align the static text controls, except this time use the Align Right button (see "Aligning the Controls" on page 3-39 for more detail).

10. From the Dialog Editor’s **File** menu, choose **Save** to save the dialog box. The Summary Info dialog box displays.
11. Enter a name such as *Start Script Dialog* in the Name field and click **OK**.
12. From the **File** menu, choose **Close** to close the Dialog Editor.

The Resulting Dialog Definition

The following example shows the dialog function definition that is pasted into the script. Each element in the array is inserted as a comment to help you identify its elements for future use:

```
var Start_Script_Dialog_Controls[]
//  Start_Script_Dialog_Controls[ "useridfield" ]
//  Start_Script_Dialog_Controls[ "PasswordField" ]
//  Start_Script_Dialog_Controls[ "userid" ]
//  Start_Script_Dialog_Controls[ "password" ]
//  Start_Script_Dialog_Controls[ "idBox" ]
//  Start_Script_Dialog_Controls[ "VersionBox" ]
//  Start_Script_Dialog_Controls[ "Version1" ]
//  Start_Script_Dialog_Controls[ "Version2" ]
//  Start_Script_Dialog_Controls[ "okButton" ]
//  Start_Script_Dialog_Controls[ "cancelButton" ]
dialog "Start Script Dialog", Start_Script_Dialog_Controls
```

`Where:`

`var Start_Script_Dialog_Controls[]`is a local array.

An array is a collection of related data values referred to by a single variable name, in this case, `Start_Script_Dialog_ Controls`. This array references all the controls (elements) on the dialog box — known as elements — by name. The value of any control can be processed by referencing it within the `[ ]` square brackets. For example, to get the value entered in the UserID field:

```
userid = Start_Script_Dialog_Controls["userid"]
```

Where:

`userid` is a variable where the value contained in the dialog element named `userid` is placed.

The second part of the definition,

```
dialog "Start Script Dialog", Start_Script_Dialog_Controls
```

executes the dialog at runtime. Double-clicking on the word `dialog` displays a graphical image of the dialog box.

Modifying the Driver Script

The section of the driver script that you need to modify is the section following the new dialog definition's syntax, but before the script attaches to the taskbar.

Use the following procedure to modify the script to use the dialog information:

1. Add the following code on a blank line following the `// Start_Script_Dialog_Controls[ "cancelButton" ]` statement:

   ```
   Start_Script_Dialog_Controls[ "userid" ] = ""
   Start_Script_Dialog_Controls[ "Password" ] = ""
   ```

   These two statements clear the contents of the userid and password controls before the dialog box is called from the script.

   The next step is to create a variable that will hold whatever values the user enters into these controls when the dialog box displays.

2. Add the following code on a blank line following the `dialog "Start Script Dialog", Start_Script_Dialog_Controls` statement:

   ```
   user = Start_Script_Dialog_Controls[ "userid" ]
   pswd = Start_Script_Dialog_Controls[ "Password" ]
   ```

   These statements create two variables: `user` and `pswd`. The variables hold the values that the user enters into the userid and password fields when the Start Script Dialog box displays.

   Similarly, you will need to assign a variable that registers the **Cancel** button.

3. Add the following code on a blank line following the `pswd = Start_Script_Dialog_Controls[ "Password" ]`statement:

   ```
   cancelbutton = Start_Script_Dialog_Controls[ "cancelButton" ]
   ```

   Now that you've told *QA*Run to recognize the dialog box's **Cancel** button, you must provide *QA*Run with instruction on how to proceed if the button is actually pushed. To accomplish this, you will need to insert some If…Else logic following the variable's declaration.

4. Insert the following code on a blank line following the `cancelbutton = Start_Script_Dialog_Controls[ "cancelButton" ]` statement:

   ```
   If cancelbutton = 1
      Stop
   Endif
   ```

   This code tells *QA*Run to stop processing the script if the dialog box's **Cancel** button is selected.

The Resulting Script

After completing the above steps, the beginning of your test script should look very similar to the following example. The code that was manually entered based on the above steps is highlighted in bold typeface:

```
Include "Autosync"
Function Main
Call StartAutosync

var Start_Script_Dialog_Controls[]
//  Start_Script_Dialog_Controls[ "useridfield" ]
//  Start_Script_Dialog_Controls[ "PasswordField" ]
//  Start_Script_Dialog_Controls[ "userid" ]
//  Start_Script_Dialog_Controls[ "password" ]
//  Start_Script_Dialog_Controls[ "idBox" ]
//  Start_Script_Dialog_Controls[ "VersionBox" ]
//  Start_Script_Dialog_Controls[ "Version1" ]
//  Start_Script_Dialog_Controls[ "Version2" ]
//  Start_Script_Dialog_Controls[ "okButton" ]
//  Start_Script_Dialog_Controls[ "cancelButton" ]

Start_Script_Dialog_Controls[ "userid" ] = ""
Start_Script_Dialog_Controls[ "Password" ] = ""

dialog "Start Script Dialog", Start_Script_Dialog_Controls

user = Start_Script_Dialog_Controls[ "userid" ]
pswd = Start_Script_Dialog_Controls[ "Password" ]
cancelbutton = Start_Script_Dialog_Controls[ "cancelButton" ]

If cancelbutton = 1
    Stop
Endif

Attach "PopupWindow~1"
   Button "Start", 'Left SingleClick'
   PopupMenuSelect "Run..."
```

Additional Script Modifications

This script now contains sufficient information for *QA*Run to set up the dialog box; however, you now need to modify certain areas of the script to replace information that is “hard-coded” with the variable information obtained through the use of the dialog box. In order to accomplish this, some extra code must be inserted.

Use the following procedure to modify the script to use data obtained from the dialog box:

1. Locate the section of the script the contains the Attach statement which types the *QA*Run directory path into the Run dialog box. This section of your script should resemble the following information:

   ```
   Attach "Run PopupWindow"
   ```

```
ComboText "&Open:",
    """C:\Program Files\Compuware\QARun\Demos\TESTBED.EXE"" -v1"
Button "OK", 'Left SingleClick'
```

Because the driver script will eventually be used to run both Version 1 and Version 2 of Testbed, you need to replace the hard-coded information in the above example with If…Else logic that can respond to data that the user enters in the dialog box.

2. Modify the above sample code to match the following example (code associated with the changes is highlighted in bold typeface):

```
Attach "Run PopupWindow"
    If Start_Script_Dialog_Controls[ "version1" ] = 1
            ComboText "&Open:",
                """C:\Program Files\Compuware\QARun\Demos\TESTBED.EXE"" -v1"
    Else
            ComboText "&Open:",
               """C:\Program Files\Compuware\QARun\Demos\TESTBED.EXE"" -v2"
    Endif
    Button "OK", 'Left SingleClick'
```

This code tells *QA*Run to open Testbed Version 1 if the radio button named “version1” is selected from the dialog box. If that control is not selected, run Testbed Version 2.

Next, you must replace the logon information that is hard-coded into the script with code that extracts the logon information that is entered into the dialog box.

3. Locate the section of the script that contains the Attach statement that connects to Testbed’s logon screen and enters the logon information. This section of your script should resemble the following example:

```
Attach "Testbed for Windows V1.00 Connected MainWindow"
     Type "CW{Tab}pass{Return}"
     Type "{F1}"
```

You must modify this section of the driver script to replace the information that is hard-coded with the information the user entered into the dialog box.

4. Modify the sample code to match the following example (code associated with the changes is highlighted in bold typeface):

```
Attach "Testbed for Windows V1.00 Connected MainWindow"
    Type user
    Type "{Tab}"
    Type pswd
    Type "{Return}"
    Type "{F1}"
```

This code tells *QA*Run to pass the data entered in the dialog box’s “userid” and “password” controls to the Type command. This modification allows you to let any authorized user sign on to a system without hard-coding the logon information into the script.

You do, however, want to be sure that Testbed is at the VIRTUAL MACHINE/ SYTEM PRODUCT screen before *QA*Run sends the logon information to Testbed, so you should insert a standard screen event before the logon information.

5. Position your cursor on a blank line following the `Attach "Testbed for Windows V1.00 Connected MainWindow"` statement shown in step 4 and click the **Browse Events** button. The Browse Events dialog box displays.
6. Scroll through the list and select the *VM/System Product Screen* event and click the **Insert** button. The Insert New Event dialog box displays.
7. Click the **Wait** button. The screen event is inserted into your script. Your script should resemble the following example (code associated with the event is highlighted in bold typeface):

```
Attach "Testbed for Windows V1.00 Connected MainWindow"
Wait(30, "", "VM/System Product Screen")
    Type user
    Type "{Tab}"
    Type pswd
    Type "{F1}"
    Type "{Return}"
```

8. From the **File** menu, choose **Save** to save the changes to the *Driver2* script.

Exercise Summary

After completing this exercise, you should be able to successfully:

- Create a user-defined dialog box
- Add a variety of controls to the dialog box
- Align dialog box controls
- Insert a completed dialog box definition into the script
- Pass information entered into the dialog box back into the script through the use of variables.

Chapter 4. Using Driver and Test Scripts Together

This chapter brings together the test scripts and the driver script made during the previous exercises. The driver will also be modified so that Testbed Version 2 is run. Any differences between the two versions can then be analyzed from the log file.

If you remember, during “Exercise 4 — Completing the Driver Script” you created a driver script to run the target application, Testbed. The driver script logged on to Testbed and selected several of the application’s primary options. In subsequent exercises, you created individual test scripts that performed specific checks on the application and performed some volume data entry. This chapter discusses the concepts required to create a complete test system.

This chapter contains the following exercise:

- “Exercise 10 — Using the Run Command” demonstrates how to integrate individual test scripts into a single driver script.

Exercise 10 — Using the Run Command

During “Exercise 4 — Completing the Driver Script”, you created a driver script to run the target application and select several of its available menu options. In the exercises that followed, you created test scripts to perform specific checks on the application. The final step in building a complete test system is to combine the individual test scripts and the driver script. You accomplish this using the Run command.

The Run command’s syntax is as follows:

```
RUN "Scriptname"
```

The Run command allows any script to be called by another script; however, unlike the Include command, the Run command does not incorporate the contents of the called script into the main script. When the Run command is encountered, the current script (the calling script) is suspended, and the called script is run. When the called script finishes executing, it is unloaded from memory and control returns to the calling script. Execution of the calling script resumes on the line following the Run command.

The Run command allows you to divide large systems into smaller, more manageable units.

Getting Started

Before beginning this exercise, you should close any open versions of Testbed and start *QA*Run. You should also open the driver script, *Driver2*, that you created in “Exercise 4 — Completing the Driver Script”.

Exercise Prerequisites: Before beginning this exercise you must have completed the following prerequisites:

- Access to the following scripts:
  - *AUTOSYNC* created during “Exercise 3 — Synchronizing Scripts Using System Variables”
  - *Driver2* created in “Exercise 4 — Completing the Driver Script”
  - *CustCred* created during “Exercise 5 — Using Text Checks”
  - *FindDocs* created during “Exercise 6 — Using Clock Checks To Test Performance”
  - *AddCust* created during “Exercise 7 — Using External TestData Files”
  - *Version* created during “Exercise 8 — Inserting Bitmap Checks”.
- Access to the following events:
  - Screen event named *VM/System Product Screen*
  - Screen event named *FindDoc*
  - Screen event named *RecordUpdate.*

- Access to the following checks:
  - Bitmap check named *Find Documents Bitmap*
  - Text check named *Master Credit Data Text Check*
  - Clock check named *DocumentsFound.*
- Access to the following dialog boxes:
  - Dialog named *Start Script Dialog* created in “Exercise 9 — Inserting Script Dialog Boxes”.
- Close Testbed
- Close *QA*Run.

Testing Requirements: The following test elements are created during this exercise:

- Modified script named *Driver2.*

You should allow 10 minutes to complete this exercise.

Modifying the Driver Script

Each of the individual test scripts starts at a particular point in the target application. Table 4-1 identifies the relevant location in the target application where the test script conducts check and event testing. The test scripts must be inserted into the driver script after the Testbed start location has been accessed.

Table 4-1. Test Script Insertion Points

Test Script	Testbed Start Location
CustCred	CUSTOMER MASTER MAINTENANCE MENU screen
AddCust	CUSTOMER MASTER MAINTENANCE MENU screen
FindDocs	FIND DOCUMENTS screen
Version	FIND DOCUMENTS screen

Use the following procedure to incorporate the test scripts into the driver script:

1. Locate the area of the driver script that contains the following code:

```
Attach "Testbed for Windows V1.00 Connected MainWindow"
Wait(30, "", "VM/System Product Screen")
    Type user
    Type "{Tab}"
    Type pswd
    Type "{Return}"
    Type "{F1}"
```

2. Position your cursor on a blank line following the `Type "{F1}"` statement and insert the following lines of code:

   ```
   Run "CustCred"
   Run "AddCust"
   ```

   Inserted at this location, the Run command calls the *CustCred* and *AddCust* test scripts after Testbed reaches the CUSTOMER MASTER MAINTENANCE MENU screen.

   Next, you need to insert the two remaining test scripts after the FIND DOCUMENTS screen is reached. The FIND DOCUMENTS screen was accessed by pressing the **F4** option from Testbed's MAIN MENU screen.

3. Locate the area of the driver script that contains the following code:

   ```
   Type "{Esc}"
   Type "{F4}"
   ```

4. Position you cursor on a blank line following the `Type "{F4}"` statement and insert the following lines of code:

   ```
   Run "FindDocs"
   Run "Version"
   ```

   Inserted at this location, the Run command calls the *FindDocs* and *Version* test scripts after Testbed reaches the MAIN MENU screen.

5. From the **File** menu, choose **Save** to save the changes to the *Driver2* script.

The Resulting Script

The following example shows how the modified driver script should appear. The modified and added code appears in bold typeface.

```
Include "Autosync"
Function Main
Call StartAutosync

var Start_Script_Dialog_Controls[]
//  Start_Script_Dialog_Controls[ "useridfield" ]
//  Start_Script_Dialog_Controls[ "PasswordField" ]
//  Start_Script_Dialog_Controls[ "userid" ]
//  Start_Script_Dialog_Controls[ "password" ]
//  Start_Script_Dialog_Controls[ "VersionBox" ]
//  Start_Script_Dialog_Controls[ "version2" ]
//  Start_Script_Dialog_Controls[ "version1" ]
//  Start_Script_Dialog_Controls[ "okButton" ]
//  Start_Script_Dialog_Controls[ "cancelButton" ]
//  Start_Script_Dialog_Controls[ "idBox" ]

Start_Script_Dialog_Controls[ "userid" ] = ""
Start_Script_Dialog_Controls[ "Password" ] = ""
```

```
dialog "Start Script Dialog", Start_Script_Dialog_Controls
user = Start_Script_Dialog_Controls[ "userid" ]
pswd = Start_Script_Dialog_Controls[ "Password" ]
cancelbutton = Start_Script_Dialog_Controls[ "cancelButton" ]
If cancelbutton = 1
    Stop
Endif
Attach "PopupWindow~1"
   Button "Start", 'Left SingleClick'
   PopupMenuSelect "Run..."
Attach "Run PopupWindow"
   If Start_Script_Dialog_Controls[ "version1" ] = 1
      ComboText "&Open:",
         """C:\Program Files\Compuware\QARun\Demos\TESTBED.EXE"" -v1"
   Else
      Combotext "&Open:",
         """C:\Program Files\Compuware\QARun\Demos\TESTBED.EXE"" -v2"
   Endif
   Button "OK", 'Left SingleClick'
Attach "Testbed for Windows V1.00  ChildWindow~1"
   MouseClick 186, 20, 'Left SingleClick'
Attach "Testbed for Windows V1.00 Connected MainWindow"
Wait(30, "", "VM/System Product Screen")
   Type user
   Type "{Tab}"
   Type pswd
   Type "{Return}"
   Type "{F1}"
Run "CustCred"
Run "AddCust"
   Type "{Esc}"
   Type "{F4}"
Run "FindDocs"
Run "Version"
   Type "{Esc}"
   Type "{Esc}"
```

Running the Script Against a New Testbed Version

When you created the driver script and the individual test scripts, you created them using Testbed Version 1 (the default testbed version). Because you haven't made any alterations to the actual target application (Testbed Version 1), all of the checks that you inserted into your test scripts should pass if you run them against Testbed Version 1.

For the purposes of this exercise, we are going to test a new version of the target application, Testbed Version 2, to verify that all aspects of the new target application still work as expected. Testbed Version 2 contains some application changes that will cause many of the checks that were created against Version 1 to fail.

Because Testbed's windows contain the version number as part of the attach name, the scripts you learned using Version 1 will fail if you attempt to run them against Version 2 unless you created the significant fields mask described on "Configuring QARun" on page 2-3. This mask tells *QA*Run to ignore the title and parent title components of the attach name.

Hint

As an alternative to the exercise described on "Configuring QARun" on page 2-3, you may also manually change the significant fields by double-clicking the object's name in the Object Map and de-selecting the Title and Parent Title options on the Significant Fields tab.

Use the following procedure to run the completed driver script:

Note

Remember, Testbed only allows a customer to be added to its database once. If you already ran the *AddCust* script, you must reset Testbed's database or the script will fail.

To reset the database, delete the TESTBED.DAT file, make a copy of the TESTBED.SAV file, and rename it TESTBED.DAT.

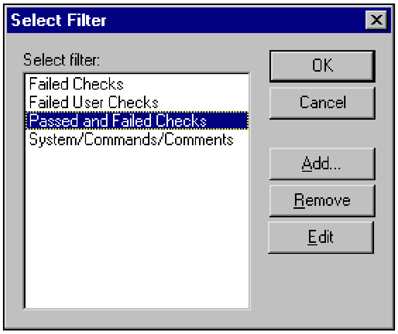
1. Ensure that all copies of Testbed are closed.
2. With the *Driver2* script open, click the **Run Script** button from the Script Editor's toolbar. The Run Script dialog box displays.
3. Click the **OK** button from the Run Script dialog box. The script begins to run and the Testbed Logon dialog box that you created appears.
4. Enter the appropriate Testbed logon information and select the Version 2 radio button.
5. Click **OK**. The *Driver2* script should continue to execute, calling each individual test script as indicated.

Analyzing the Results

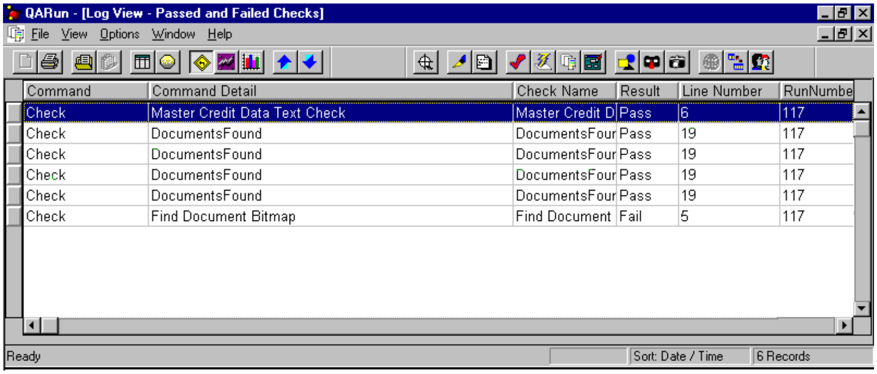
After the script completes running, Log View loads with the *Driver2* log file. You can scroll through the log file to display the results of all checks and commands *QA*Run executed. Each command and comment that *QA*Run executed is recorded; however, you are probably most interested in the results of the various checks that you inserted into the scripts. Using Log View, it is possible to filter the file so that only the check results are displayed.

Use the following procedure to filter the log file:

1. Click the **Select Filter** button from the Log View toolbar. The Select Filter dialog box displays:



2. Select Passed and Failed Checks from the Select filter area and click **OK**. The log view is filtered, now showing only the checks that were performed from the *Driver2* script as shown below:



Command	Command Detail	Check Name	Result	Line Number	RunNumbe
Check	Master Credit Data Text Check	Master Credit D	Pass	6	117
Check	DocumentsFound	DocumentsFour	Pass	19	117
Check	DocumentsFound	DocumentsFour	Pass	19	117
Check	DocumentsFound	DocumentsFour	Pass	19	117
Check	DocumentsFound	DocumentsFour	Pass	19	117
Check	Find Document Bitmap	Find Document	Fail	5	117

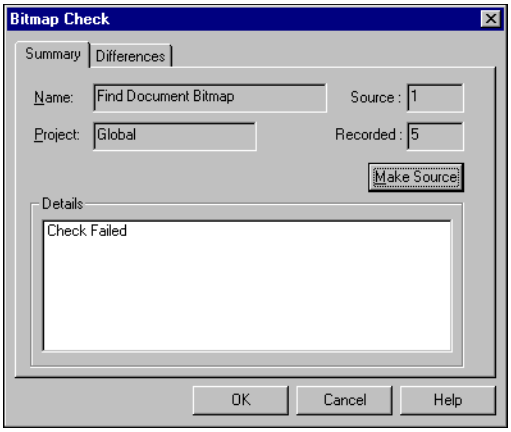
Viewing Failed Checks

Passed and failed checks are highlighted in different colors. The default colors are green and red respectively. You can change these colors by choosing Colors from the Options menu. In addition to viewing the failed check results, you can update the results of failed checks directly from the log file.

Failed Bitmap Checks

One check that should have failed when you ran the script was the toolbar check on the FIND DOCUMENTS screen. The name of this check was *Find Documents Bitmap*. This check failed because an additional toolbar button appears in Testbed Version 2 that was not present in Testbed Version 1.

1. To view the differences between the *expected* check results and the *actual* check results, double-click on the failed check's Check Name column in Log View. The Bitmap Check dialog displays:



Bitmap Check

Name: Find Document Bitmap

Project: Global

Source : 1

Recorded : 5

Details

Check Failed

2. Click the Bitmap Check dialog box's Differences tab to display the expected and actual bitmap image.

   - Click the **Zoom** button to display each bitmap in a separate window. These windows may be sized to achieve a clearer view of the bitmaps. Click the **Zoom** button again to close the window.
   - Click the **Show differences** button to see a consolidation of the results.

Failed Clock Checks

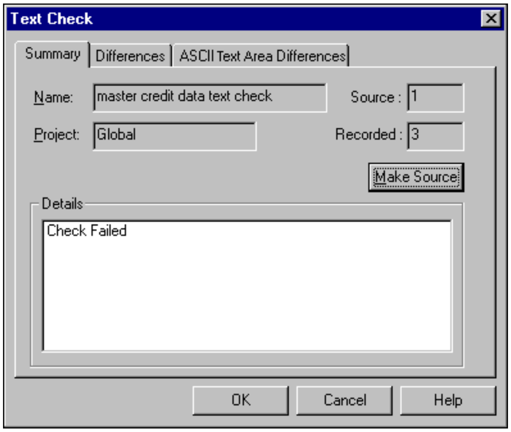
Clock checks record the *actual* and *expected* time results in the log file and can be viewed by displaying the Notes column.

If the Notes column is not displayed, choose Column Layout>Grid from the View menu and add the Notes column to the display

Failed Text Checks

The text check on the CUSTOMER MASTER CREDIT DATA screen has also failed (don’t panic, it was designed to).

1. Double-click the *Master Credit Data Text Check* check name from the log. The following dialog box displays:



Text Check

Name: master credit data text check

Source : 1

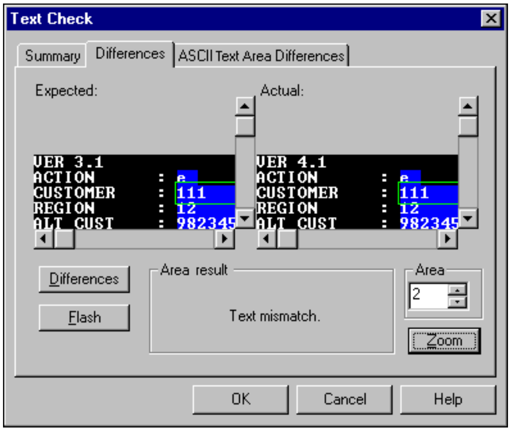
Project: Global

Recorded : 3

Details

Check Failed

2. Click the Differences tab to view the specific areas were the check failed. The following information displays:



Text Check

Expected:

```
VER 3.1
ACTION      : e
CUSTOMER    : 111
REGION      : 12
ALT CUST    : 982345
```

Actual:

```
VER 4.1
ACTION      : e
CUSTOMER    : 111
REGION      : 12
ALT CUST    : 982345
```

Area result

Text mismatch.

Area: 2

3. Click **Zoom**.

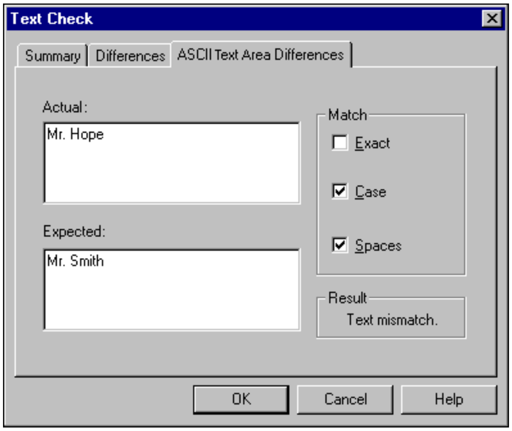
Two windows appear on the screen: one that displays the expected results and one that displays the actual results. The areas that passed are displayed in green. Those that failed are displayed in red.

4. Click the **Restore** button to again view the expected and actual results within the Text Check dialog box's Differences tab.

   In addition to using the **Differences** and **Flash** buttons to highlight the areas where the check failed, you can also use the Area spin control to view the results of each Include are in the Area Result box.

   Notice that the ASCII text check in area 2 failed.

5. Click the ASCII Text Area Differences tab to display the specific differences between the expected and actual check results.



Actual:
Mr. Hope

Expected:
Mr. Smith

Match
□ Exact
☑ Case
☑ Spaces

Result
Text mismatch.

When you created the check, you told *QA*Run to expect to find the text "Mr. Smith" in the CONTACT field. When the script was run, the text "Mr. Hope" appeared in the field, causing the check to fail.

Updating Checks

All failed checks, with the exception of clock checks, can be updated directly from the log file. Use the following procedure to update the failed checks:

1. From the log file, highlight the check name and right mouse-click.
2. Choose **Make Source** from the pop-up menu.

Exercise Summary

After completing this exercise, you should be able to successfully:

- Use the driver script to call individual test scripts
- Use the Run command from within scripts
- Use the driver script to run a different version of the target application
- Create a filter to view only check results
- View and update failed checks.

Index