CHAPTER 1

INTRODUCTION TO TESTING

"Testing is the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements, or to identify differences between expected and actual results"

- Institute of Electrical and Electronics Engineers (IEEE).

Software applications have grown in recent years from simple console based applications that perform a single task. Now they are complex, visually pleasingly and animated programs that perform a variety of tasks. It is no longer unusual for a single application to perform complex tasks, integrate with many other applications as well as multiple back-end systems, and process documents in a variety of formats. An example of this is online banking.

The features available in these software applications have substantially advanced out of a need to support a more sophisticated and computer savvy user base. Additionally, today's users have many more choices of computing devices such as PCs, laptops, PDAs, Tablet PCs, Smartphones, Pocket PCs etc. Also a great variety of operating system choices are available for these devices and all of these systems require software to be developed for performing various tasks.

The need for testing these software applications is therefore very clear because testing helps to ensure that applications developed meet with the original intent of the customer. The complexity that applications must support today comes at a price to developers because each of the many features being created have to be tested in multiple scenarios and different environments to ensure that they work properly as intended. And in the event that the user performs an invalid operation, the system should generate a meaningful message to the user about why the failure occurred.

It is not unusual to download and use an application and find that an attempt to do something impossible, like save a document directly to a CDROM drive, will cause the application to crash. This is because the developer of the application did not foresee the multiple scenarios that end-users of the application would attempt while using the application. While the end-user might be wrong to save a document to a read-only drive, it is still important for the application to recover from the scenario gracefully without causing the user to lose the work done to their document.

This is precisely why we do software testing. It provides the ability to rigorously test a software application to ensure that the application meets with the originating requirements and expectations. Testing also helps to ensure that the application continues to function even when used in unintended ways. If problems are found during the development phase of an application, they can be fixed before the application is made available to end-users.

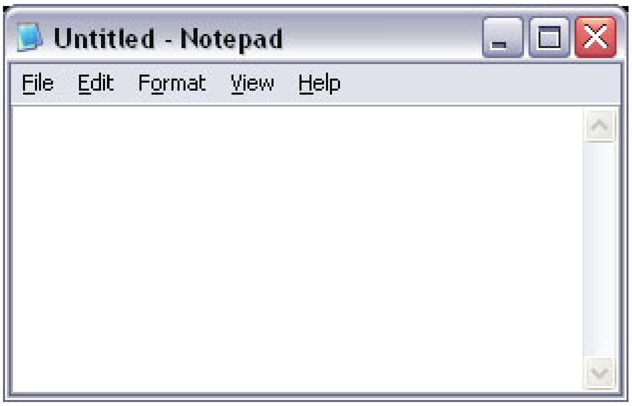


Figure 1.1: The Windows Notepad application.

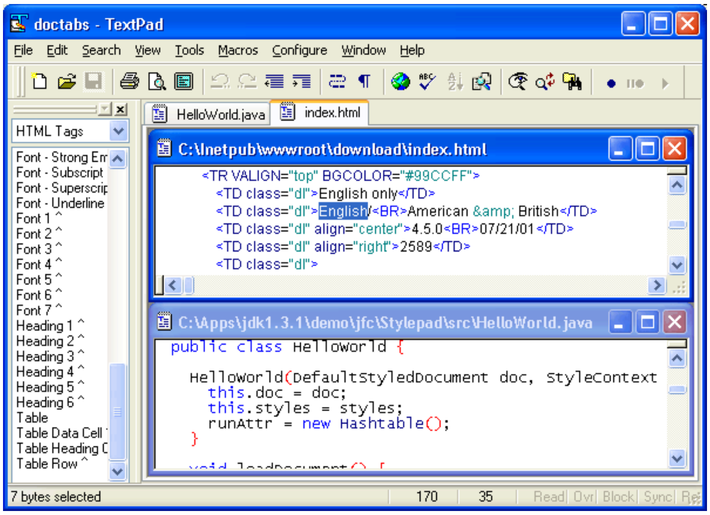
This directly impacts the cost of development. You may be familiar with the 1:10:100 rule of testing that states that the cost to fix a defect increases exponentially the longer it takes to identify the defect. Testing thereby helps ensure that the end-user is provided with a quality product and an error-free experience.

An example of the exponential growth in application features can be seen in document editing software. Notepad, a standard application shipped with all versions of Microsoft Windows and shown in Figure 1.1, is a simple text editor capable of editing text documents.

NOTE: The notepad application is accessible from Start ▶ Programs ▶ Accessories ▶ Notepad

---

Long since the standard for quick document editing exercises, Notepad is rapidly losing its dominant place to a host of new tools that provide more advanced features that are useful while editing text documents. Textpad (www.textpad.com) shown in Figure 1.2 is an example of the new generation of text editors. The versatile document editing application that can be used to edit documents of many formats providing such useful features as formatting, multiple document view and syntax highlighting.



```
    <TR VALIGN="top" BGCOLOR="#99CCFF">
      <TD class="dl">English only</TD>
      <TD class="dl">English</BR>American &amp; British</TD>
      <TD class="dl" align="center">4.5.0<BR>07/21/01</TD>
      <TD class="dl" align="right">2589</TD>
      <TD class="dl">
```

```
public class HelloWorld {

    HelloWorld(DefaultStyledDocument doc, StyleContext
      this.doc = doc;
      this.styles = styles;
      runAttr = new Hashtable();
    }
```

Figure 1.2: The Textpad application

In Figure 1.2 Textpad is shown editing both a HTML and Java document in two windows. Notice that it provides syntax highlighting features to these different file formats.

Developers of editing tools such as this would need to put the application through a rigorous process to test all these advanced features.

Software Development Life Cycle

The development of software applications is based on a lifecycle process known as the software development lifecycle (SDLC). SDLC is a software engineering process that provides information about steps that need to be taken to advance a software project from concept to deployment. SDLC is broken into 5 phases with each phase dependent upon results from the preceding phase. Each phase also specifies what must happen during that portion of the software process. The testing process, which we are primarily concerned with in this book, is the fourth phase in the SDLC process.

SDLC helps describe an approach which, when followed properly, ensures that applications meet both the functional and visual requirements stipulated in their creation documents. The stages contained within the SDLC process include:

Requirements Analysis – The gathering of details about what needs to be done, and how the application should perform those tasks. This phase is handled by Business Analysts (BA) who must meet with all the stakeholders in the application to capture all the needs and details of what they want the application to do. The most difficult part of this process is ensuring that the requirements are:

Complete – containing all necessary functionalities needed by the users.

Non-conflicting – Often with a large user base it is easy to have a group of users need a feature that conflicts with features needed by other users. This must be negotiated during this stage because the application cannot be built with conflicting requirements.

Unambiguous – written in clear language so that the same meaning can be derived when different parties read the document.

The result of this process is determining the features that must exist in the final application.

Design Phase – High-level design of the application to specify how the application will be created. This is often managed by a software architecture team who take into consideration the programming language and software platform in which the application will be developed. These details will determine how the internal structure of the application will be designed. Additional activities during this phase includes the creation of flowcharts that describe how components within the application will interact, designing the screen layout that dictate what the application would look like, and also creating the database structures that the application might use to store data.

Implementation and Unit Testing – Involves writing the code for the application using the chosen programming language and software platform from the design phase. A software application being created in the Java programming for instance might be coded using the Abstract Windowing Toolkit (AWT), or Java Foundation Classes (JFC) or even the Standard Widget Toolkit (SWT) for the object library. The choice of the tool to use is made during the design phase, and the implementation would simply create the application conforming to this decision. The choice of interface technology determines the actual type of software code that the developer will write. During this process, unit testing is performed at the code level to ensure the correctness of the software functions that are created by the programmer. Unit testing also ensures the several functions can work together in completing tasks.

Testing – The testing phase covers the process of planning tests that will be run on the application to examine its functionality. The phase also describes how the test would the execution of these tests. Decisions are made about using a manual or automated approach and test cases are created based on the chosen test strategy. During this phase, the goal is to test the entire application to ensure that when all the separate components that make up the application are assembled, the application functions as described in the requirements documents.

Maintenance – On completion and release of an application, there is often a need for corrective maintenance to handle defects that were undiscovered in the development process, and preventive maintenance to mitigate issues that might arise from external threats. Additionally there is often a desire to upgrade certain functionality for the application. For deployed applications such as your bank's web application, there is also a need to support to users and manage configurations of the software system. The maintenance cycle describes how this process should be handled.

From the above, you may notice that the SDLC process is originally intended to be a linear process with one phase completing before the commencement of the next. You must have requirements before you know how to design your database structure, and likewise you need this database structure to know what tables and fields to reference in your programming code.

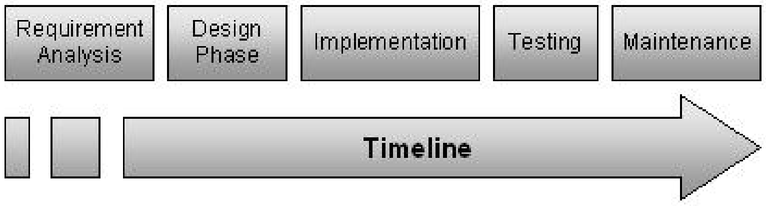


Figure 1.3: SDLC Timeline

Figure 1.3 shows the SDLC process implemented linearly. However, as time is always a major factor in software development with many companies seeking to reduce the time-tomarket for new products, methodologies have been created to optimize SDLC by allowing certain parts of it to function in parallel. These methodologies are processes designed to maximize the efficiency of the SDLC. Some of these methodologies include Scrum, Agile, Extreme Programming (XP) and the Rational Unified Process (RUP). The underlying goal of each these methodologies are to optimize the software development timeline while ensuring the quality of the resulting product.

By staggering the starting time of the subsequent phases, it is often possible to achieve a level of parallel development. As shown in Figure 1.4, this has a net effect of saving the project time by having multiple things done simultaneously. Note that the difference in the methodology being used will impact how the phases are staggered.

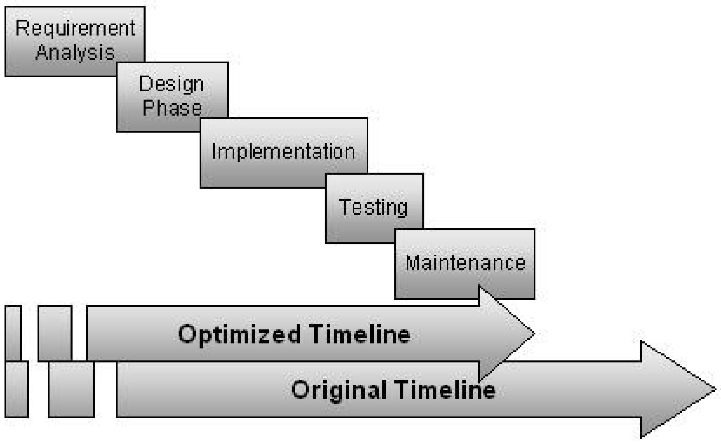


Figure 1.4: Optimized SDLC Timeline showing concurrent phases.

ASK THE EXPERTS

Q: Should the functional tester understand the programming language the application was created in?

A: This is not necessary, during functional testing we are only exposed to the completed application and not to the source code. Functional testing ignores the internal mechanisms of an application and focuses on the output provided for a given input.

Q: What is the best software development methodology?

A: This question is analogous to asking "what is the best programming language?" Each methodology has its strong suit and expectedly its weaknesses. All of them however, seek to make it easier for a project to be completed successfully on-schedule, and on-budget.

SDLC Testing Phase

The testing phase as a part of the SDLC process governs the preparation for testing, the management of testing resources and the creation of test cases. It also manages the execution of these tests as well the creation and tracking of defects found within the software application.

Although there is less consensus about the phases of the software testing process than there is about the SDLC process, you can generally find that the testing software systems is broken down in to the following phases:

Test Planning - The goal of this phase is to create a process that specifies the details as well as the logistics involved with the testing process. It takes into account the tools that will be used to test the application, the testing environment etc. Usually, a test plan document results from this phase, describing in details such crucial items as:

- Roles and responsibilities of everyone involved in the testing process
- Process of setting up the test environment
- Obtaining and managing test data
- Standards for test creation and execution
- Preliminary schedule for testing
- Test management process
- The defect-tracking process
- Software tools that would be used through the testing process

Test Design - Involves the breakdown of the functional and performance requirements for the application into a series of test cases. These test cases are documents that describe how a specific functionality is going to be tested. In order to successfully test any functionality, it is important that the test case completely identifies what needs to be done. Also, the test case needs to specify the sequential process that needs to be followed, any results expected and include all other relevant pieces of information. Table 1.1 contains a list of useful information that would be found in a test case.

Item	Description
Test Name	The name that would be given to the test script. This would allow a correlation between the test case that describes what is being tested and the test script that performs the test.
Purpose	The feature that is being tested.
Pre-Condition	The initial condition that must be met for the feature being tested to be properly tested.
User action	A sequential step-wise listing of actions that the user is expected to perform in order to appropriately test the desired functionality.
Expected result of each action	The result that is expected after each action performed by the user.
Post-Condition	The condition in which the test environment should be returned to on the completion of this test.
Valid Test Data	Source and location where valid test data can be found to test the functionality being tested. This data will be either hard-coded into the test script, or preferably, entered into the data table and parameterized into the test

Possible Exceptions	List of possible exceptions that can occur during this test. This information is used for creating recovery scenarios.

Table 1.1: Useful data needed in a test case

We have provided below in Listing 1.1, a test case that contains all this useful information. You should note that using tools such as Mercury Interactive's Test Director, it is easy to capture and track this crucial information.

```
Test Name:
      Add Expense

Purpose:
      Verify that when expense records are added
      they display within the application and are
      added to the database

Pre-Conditon:
      Application is launched, with the main page
      displayed.
      Note the number of expense records displayed
```

Step	User Action	Expected Result
1	Click the Today's Date button	Correct system date appears in the available textbox
2	Enter in additional details for an expense record	Entered details are displayed
3	Click the save button	a. Expense is displayed in the grid b. Totals' textbox is update c. Expense record is saved to database

```
Post-Condition:
      Close the application
```

```
Valid Test Data:
      Category: Movies
      Description: LOTR Return of the King at Lennox
      Amount: 11.50

Possible Exceptions:
      N/A
```

Listing 1.1: Add Expense Test Case

Test Development - This phase is only necessary for software applications that will be tested using automated software testing tools. In this phase, the test cases created in the previous stage, are converted into executable scripts. These scripts will then be executed using an automated testing tool. This step is unnecessary for applications that are being manually tested because manual testing involves the performance of the contents in the test case by individuals.

We will provide more discussion on manual vs. automated testing in the next chapter.

Summary

In this chapter you have learned:

What software testing is, and why we test software applications.

The software development lifecycle, and where testing fits into this process.

How to shorten the software development lifecycle by making certain phases run concurrently.

The different parts of the software testing process.

CHAPTER 16

RAPIDTEST SCRIPT WIZARD

Sometimes you may want to start your test creation activity with the shortest possible amount of preparation. This may be because you are doing some ad-hoc testing or probably under a deadline to deliver some results. It may also be that you want to quickly complete the process of GUI Map creation. Whatever the case may be, for this jumpstart approach, WinRunner provides a tool that can be used to quickly create a GUI Map and some automated tests. This tool is known as the RapidTest Script Wizard.

The RapidTest Script Wizard is a wizard tool. Similar in nature to some of the wizards we have discussed in this book such as the DataDriver Wizard. Wizards are helper tools that make performing complex tasks easier. A wizard gathers details from you about the operation you want to perform using a series of steps. After gathering all relevant information, the wizard performs the needed operation.

The RapidTest Script Wizard performs exactly like this. It displays a series of pages to the user, with each page asking for some information. Once this information gathering is complete, the wizard creates products based on the answers that were provided.

The WinRunner RapidTest Script Wizard can perform the following tasks:

1. Automatically learn all the objects that exist in an application and save this into a GUI Map file.
2. Create 3 different types of tests.

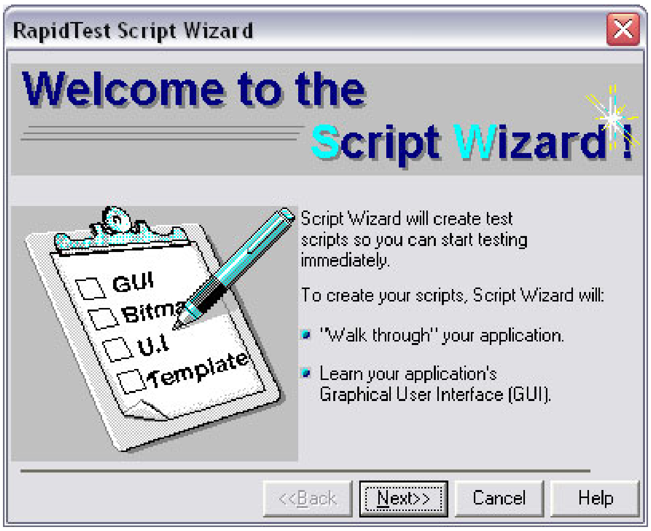
3. Generate a test template that can be used for future tests.
4. Automatically configure the WinRunner startup to preload certain resources.

Every item that is created by this wizard can be manually created. I have shown you how to do this in the previous chapters. The reason you may want to use the wizard is that it simplifies the test creation process and could be an efficient starting point for working with WinRunner.

Using the RapidTest Script Wizard

The RapidTest Script wizard is available through the Insert menu. You can launch the tool by clicking Insert ▶ RapidTest Script Wizard. Figure 16.1 displays the screen your are presented with on launching the wizard.

NOTE: The RapidTest Script Wizard is not available when certain add-ins are loaded. It is also not available when you are working in GUI Map File per Test mode.



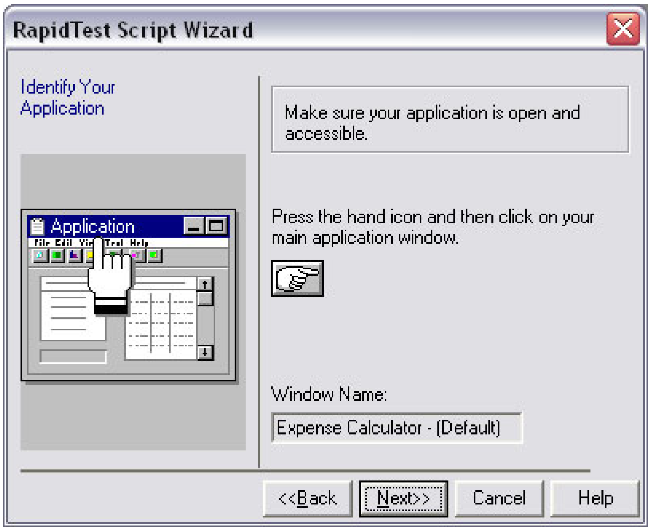
Script Wizard will create test scripts so you can start testing immediately.

To create your scripts, Script Wizard will:

- "Walk through" your application.
- Learn your application's Graphical User Interface (GUI).

Figure 16.1: Welcome page of the RapidTest Script Wizard.

This first page of the wizard simply welcomes you to the tool and states what functionality the wizard will provide. Click Next and the first interactive screen, shown in Figure 16.2 is displayed.


RapidTest Script Wizard

Identify Your Application

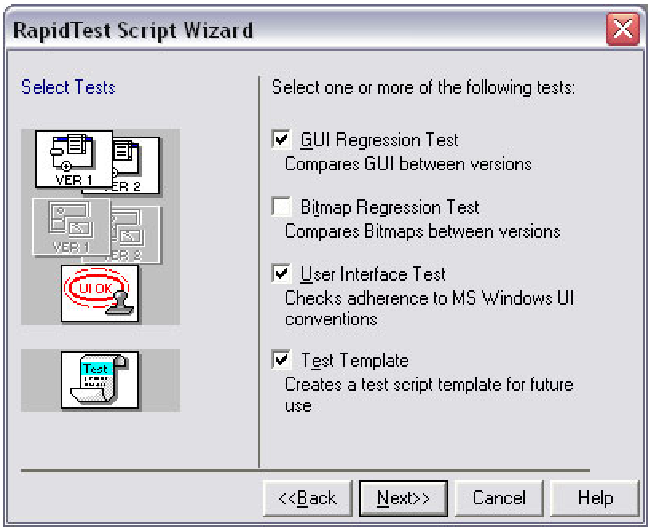
Make sure your application is open and accessible.

Press the hand icon and then click on your main application window.

Window Name: Expense Calculator - (Default)

Figure 16.2: The Identify Your Application page.

Using the pointer choose a window in your AUT. The name of this window is immediately displayed in the Window Name field. If you chose the wrong object, simply click on the pointer and try again. Once you have chosen the correct window, click Next.


RapidTest Script Wizard

Select Tests

Select one or more of the following tests:

[✓] GUI Regression Test
Compares GUI between versions

[ ] Bitmap Regression Test
Compares Bitmaps between versions

[✓] User Interface Test
Checks adherence to MS Windows UI conventions

[✓] Test Template
Creates a test script template for future use

Figure 16.3: The Select Tests page.

The next screen of the wizard is shown in Figure 16.3. In this screen, the wizard prompts you to choose what type of tests it should generate. There are 4 types of tests that can be chosen at this point.

GUI Regression Test: Compares differences in the application GUI between 2 versions of the AUT. This test is used to identify any changes that have been made to objects that exist within the AUT.

Bitmap Regression Test: Similar to a GUI Regression Test, but uses bitmap checkpoints instead. It compares visual differences between two versions of your AUT. Remember that regression tests simply identify differences and that differences are not necessarily defects.

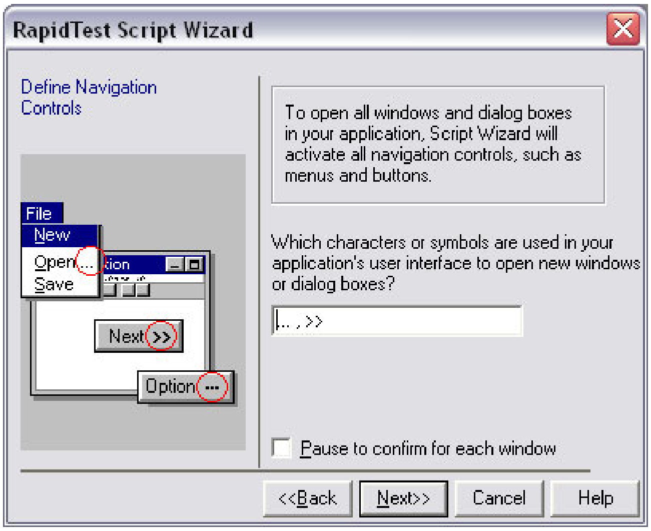
User Interface Test: Tests the conformance of your AUT to specifications provided by Microsoft for application design. While this is a good test to run, remember that for your specific application, the requirements documents determine what is a defect and what isn't.

Test Template: A test script template for use with future tests. It contains TSL code that manages navigation to all the windows in the AUT and can be used as a starting point for any test script.

NOTE: The GUI Regression Test and Bitmap Regression Test are mutually exclusive. This means you can only choose one of the two.

As shown, I have chosen the GUI Regression Test, User Interface Test, and Test Template. Click Next.

TIP: If you are using the wizard simply as a tool to automate the creation of your GUI Map file, you may choose not to select any tests at this step.


RapidTest Script Wizard

Define Navigation Controls

To open all windows and dialog boxes in your application, Script Wizard will activate all navigation controls, such as menus and buttons.

Which characters or symbols are used in your application's user interface to open new windows or dialog boxes?

..., >>

☐ Pause to confirm for each window

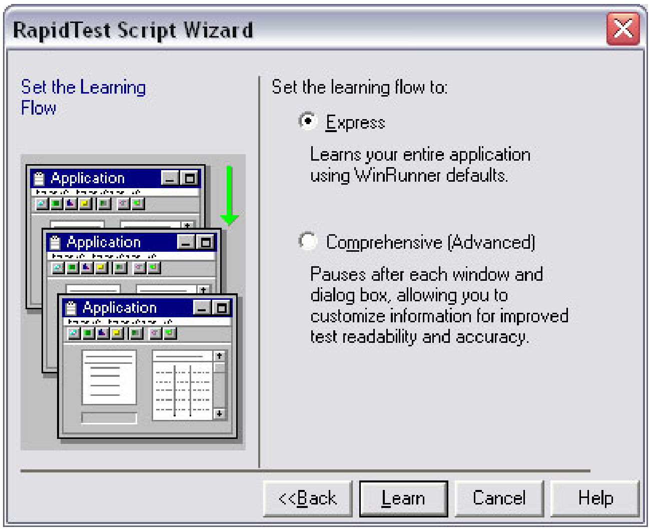
Figure 16.4: The Define Navigation Controls page.

The next step requires you to specify navigational markers that are used to open windows and dialogs in your AUT. By convention, in the Windows environment, programmers use an ellipsis (...) to denote an operation that launches a new window. You may have seen this when you look in the File menu of an application like WinRunner. Additionally, >> is used to specify that a dialog will be opened. As shown in Figure 16.4 even the RapidTest Script Wizard also employs the use of these. (<< is often used to re-open a previously opened dialog so it does not need to be included here).

If your application uses other navigational controls, then you want to include those in this step. For the Expense Calculator application, several menu items launch new dialogs. These include File ▸ New Account, File ▸ Open Account, Actions ▸ View Summary and Help ▸ About. To ensure that the wizard opens all windows, the following values were provided in this field:

```
..., New Account, Open Account, View Summary
```

These are the text values on all objects that open a different dialog in Expense Calculator. Notice that the values are separated by commas. Click Next.


RapidTest Script Wizard

Set the Learning Flow

Set the learning flow to:

[● Express]

Learns your entire application using WinRunner defaults.

○ Comprehensive (Advanced)

Pauses after each window and dialog box, allowing you to customize information for improved test readability and accuracy.

Figure 16.5: The Set the Learning Flow page.

The next dialog requires you to specify the learning flow. The choices are:

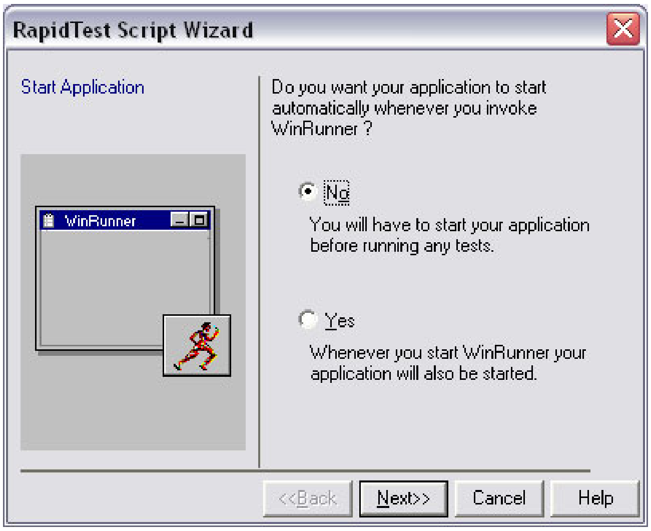
Express - WinRunner learns the entire set of objects within your AUT without pausing.

Comprehensive - WinRunner learns all the objects within your AUT and pauses after learning each object to allow you make changes to the recorded properties.

While both choices achieve the same result of creating a GUI Map file, the Comprehensive learning flow provider a greater level of control because it allows you to make changes to the GUI Map file being recorded. Recommended changes include editing the logical names that WinRunner provides for these items. In my opinion, this makes it a better choice. But it is also slower.

As shown in Figure 16.5, I have selected the Express mode for this book. Use this mode if you are new to this wizard or doing casual testing. This mode is considerably faster. Now, we are ready to have the wizard create the GUI Map. Click Learn and just sit back.

The RapidTest Script Wizard now begins to walk through the entire application. As it does so it opens up each of the windows in the AUT, learns the object and repeats the process on the next window. You may be asked to show the tool how to perform a certain operation such as closing a window, if the wizard gets stuck in a process. The entire process is exceedingly fast, and on completion, the dialog in Figure 16.6 is displayed.


RapidTest Script Wizard

Start Application

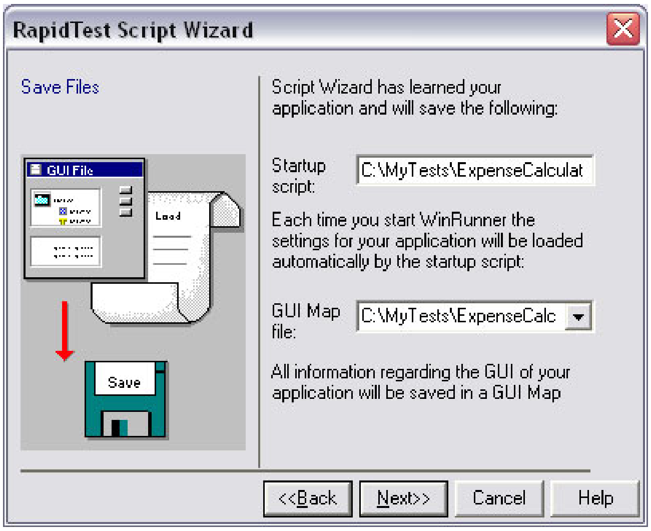
Do you want your application to start automatically whenever you invoke WinRunner ?

[✓ No]
You will have to start your application before running any tests.

Yes
Whenever you start WinRunner your application will also be started.

Figure 16.6: The Start Application page.

This dialog asks if you want WinRunner to automatically launch your AUT whenever WinRunner is launched. I typically enter No in this screen because I test a variety of applications and having my AUT pop-up every time I run WinRunner would be a nuisance. Click Next.


RapidTest Script Wizard

Save Files

Script Wizard has learned your application and will save the following:

Startup script: C:\MyTests\ExpenseCalculat

Each time you start WinRunner the settings for your application will be loaded automatically by the startup script:

GUI Map file: C:\MyTests\ExpenseCalc

All information regarding the GUI of your application will be saved in a GUI Map

Figure 16.7: The Save Files page.

In the dialog shown in Figure 16.7, WinRunner requests you to enter 2 pieces of information.

Startup Script - The name and location of a script that will be executed whenever WinRunner is started. This script contains relevant functions for running the tests created by WinRunner. Listing 16.1 shows the contents of this startup script.

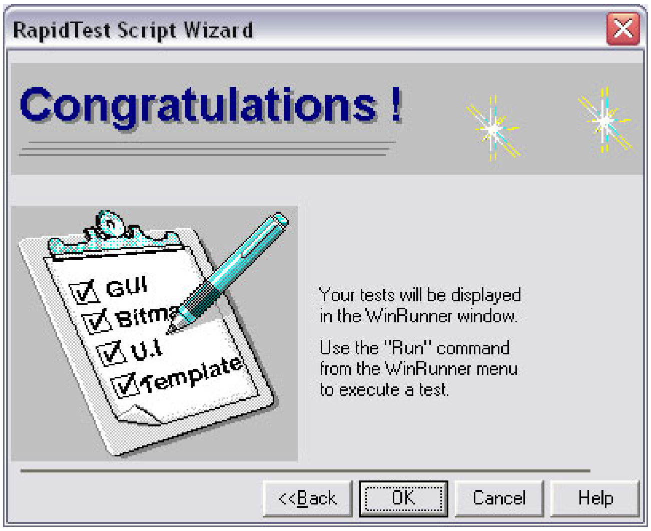
The GUI Map File name - This is the name that the GUI Map file should be saved as and the location on the filesystem for the file.

```
GUI_load("C:\\MyTests\\ExpenseCalculatorRapidTest.GUI")
```

Listing 16.1: The startup script created.

An additional dialog is also displayed prompting you for a name for each of the tests you instructed the wizard to create.

Click Next from this dialog and the dialog shown in Figure 16.8 is displayed. This dialog merely congratulates you for successfully using the RapidTest Script Wizard for your testing needs.



Your tests will be displayed in the WinRunner window.

Use the "Run" command from the WinRunner menu to execute a test.

Figure 16.8: The Congratulations page.

Congratulations indeed! You have successfully used the RapidTest Script Wizard. Now that the tests are created, you will notice that several tests have been added to your test window. Listing 16.2 shows the test template that was created by the wizard.

```
time_out=getvar("timeout");
win_open("Expense Calculator - (Default)",time_out);
#Enter code for window "Expense Calculator - (Default)" here

win_open("About Expense Calculator",time_out);
#Enter code for window "About Expense Calculator" here

type("<kEsc>");
win_open("Expense Summary",time_out);
#Enter code for window "Expense Summary" here


win_close("Expense Summary");
win_open("New Account",time_out);
#Enter code for window "New Account" here

win_close("New Account");
win_open("Open Account",time_out);
#Enter code for window "Open Account" here
```

```
win_close("Open Account");
```

Listing 16.2: The test template create for Expense Calculator.

Notice that all the test template does is open your AUT's windows by using the win_open function. It also closes these windows after use using the win_close function. For your tests, you could choose to record your actions into this TSL test template. Don't forget to remove sections that are irrelevant to your current test case.

STARTUP SCRIPTS

Whenever WinRunner is launched, it executes a script file that sets certain variables and performs some operations. This script file is named **tslinit** and is found with the **dat** folder of the WinRunner installation.

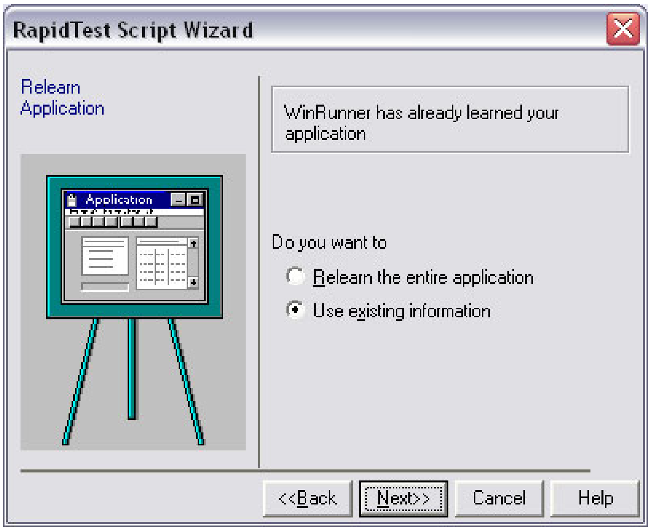
Unless you are a WinRunner expert, it is not advisable to edit this script. If you want additional operations to be performed by WinRunner at startup, you should create your own WinRunner script and specify this as the WinRunner startup test. WinRunner will execute this on startup after executing the **tslinit** script.

To inform WinRunner to run your script, navigate to Tools ▶ General Options ▶ General ▶ Startup and specify your test script in the startup test field.

Repeating the RapidTest Script Wizard

If you launch the RapidTest Script Wizard to create scripts for a application that already has a GUI Map file created and loaded into WinRunner, WinRunner immediately detects this and generates a different screen to verify what you want to do.

TIP: Always keep a backup of your GUI Map file and avoid making changes to this file late in the testing process. It might prevent your existing tests from working properly.


RapidTest Script Wizard

Relearn Application

WinRunner has already learned your application

Do you want to

Relearn the entire application [✓ Use existing information]

Figure 16.9: The Relearn Application page.

Shown in Figure 16.9, this screen informs you that the objects in your application are already known to WinRunner and prompts you to choose one of the following actions:

Relearn the entire application – create a new GUI Map for the application and use this for all tests.

Use existing information – use existing information from the loaded GUI Map file to create the tests.

Summary

In this chapter you have learned:

- What the RapidTest Script Wizard is and how to use it.
- The different items that can be created by the RapidTest Script Wizard and what they are used for.
- How to set a startup test in WinRunner to run whenever you launch the tool.