Automate regression tests

IBM Rational Functional Tester makes regression testing a snap

Skill Level: Introductory

Brian Bryson
Author
IBM

21 Feb 2005

This tutorial introduces you to IBM Rational Functional Tester, an automated regression testing tool that lets you test Java, .NET, terminal-based and web applications running on Windows and Linux platforms. The tutorial will help you create, execute and edit a Functional Tester regression test. Along the way you'll explore the challenges of regression testing and the solutions offered by Functional Tester to address these challenges.

Section 1. Introduction

Overview

This tutorial introduces the first-time user to IBM® Rational® Functional Tester. IBM Rational Functional Tester is an automated regression testing tool that lets users test Java™, .NET, terminal-based, and Web applications running on Windows® and Linux platforms.

In this tutorial, you'll learn how to create, execute and edit your first Functional Tester regression test. Along the way, you'll explore the challenges of regression testing and the solutions offered by Functional Tester to address these challenges.

You will learn how to do the following:

- Record a Functional Tester regression test

- Become familiar with Functional Tester's scripting languages: the Java language and Visual Basic.NET
- Use data-driven testing techniques to increase the ROI of testing
- Use ScriptAssure technology to make tests resilient to application changes
- Validate static and dynamic application content

You'll achieve all of this without typing a single line of code. Ready?

Who should take this tutorial?

This tutorial is designed to introduce testers to automated regression testing with IBM Rational Functional Tester. No previous experience with IBM Rational Functional Tester or any other automated regression testing tool is required. You'll start at the beginning and quickly become competent to create tests against your application.

Prerequisites

If you don't already own a copy of IBM Rational Functional Tester, you can download a fully functional trial version . The installation process is described in Install Functional Tester. No special configuration or installation options are necessary for this tutorial.

Functional Tester supports two scripting languages: the Java language and Visual Basic.NET. If you want to craft tests using the Java language, Functional Tester includes the Eclipse open source development environment. No further installation steps are required. If you want to use the Visual Basic.NET scripting language, Visual Studio.NET must be installed on your system before you install IBM Rational Functional Tester. Visual Studio.NET is available for purchase from Microsoft.

Section 2. About regression testing

The challenges of regression testing

Eventually, in any software system, users will uncover defects or bugs in the program. Typically, these bugs are fixed, the fixes are tested, and the updated software is released back to the user community. However, due to the interconnected nature of software, even the smallest change can wreak unpredictable havoc when the implications of that change are not properly

understood. Any software change, even one that corrects a known defect, can affect a system in an unforeseen manner and potentially cause problems that are worse than those that the change was originally trying to address.

Regression testing is the practice of retesting of a software system that has been modified to ensure that no previously-working functions have failed as a result of defect reparations or newly added functionality. Comprehensive regression testing fully ensures that a software system is functioning as designed. Comprehensive regression testing, however, is rarely feasible, given the time and resource constraints placed on the typical software development team. As a software system expands and evolves, it becomes more and more difficult to test every piece of functionality. This problem is compounded by the frequency of software builds. In an environment where software builds are done on a nightly basis, comprehensive regression testing of every build is essentially impossible. Typically, in these environments, the testing of previous functionality is foregone to allow for testing of new fixes and new functionality. This leaves open the possibility that the software team will release software with undiscovered defects.

Rational Functional Tester is an automated regression testing tool that addresses these challenges. With Functional Tester, test scripts are created for the purpose of retesting system functionality. With every build of a new system, the scripts are executed to validate previously working functionality. Scripts are generally executed in an unattended mode, freeing up the quality assurance team to test the most recent fixes and functionality. As the software expands and evolves, so does the library of test scripts, ensuring that every build of the application is adequately tested from beginning to end.

Is Rational Functional Tester the tool for you?

Ask yourself the following question to determine whether or not Functional Tester is the tool for you:

Am I testing a Java, .NET, Web, or browser-based application?

If you answered *Yes*, then Functional Tester could be the tool for you! Functional Tester was designed to test Java, .NET, and Web/browser based applications exclusively. If you are testing Visual Basic, C++, or other Windows 32-bit applications, Functional Tester is not the tool for you. For users operating in these environments, IBM Rational offers Rational Robot. Similar in concept to Functional Tester, Rational Robot enables automated regression testing of Windows-based applications.

Now, consider these questions:

- Are you forced to forgo some tests due to time constraints?
- Are you spending too much time on test script maintenance?
- Are you testing a complex application that is hostile to record and

playback?

If you answered *Yes* to any of the above, then you need to take a good look at what Functional Tester can do for you. Continue on through this tutorial and explore the tool. In about an hour, you'll go through the process of recording, editing, executing, and analyzing your first Functional Tester script. Once complete, you should be in a good position to understand the value that Functional Tester can bring to your software development process.

Section 3. Get started

Overview

Getting started with Functional Tester involves three steps:

1. Installing Functional Tester
2. Installing and/or configuring the Java Runtime Environment
3. Creating a test project

In this section, you'll walk through those steps.

Install Functional Tester

If you haven't already, install Functional Tester now. A fully functional trial version of the software is available from the IBM Web site; see Resources for a link.

The Functional Tester trial is delivered in two pieces. The first is the C81MWML.BIN file, which contains the compressed software; the second is an application called the *extractor*. Download both and run the extractor application. The extractor processes the .BIN file, expands it into the application installation files, and launches the setup for you. On the setup wizard, select **Install IBM Rational Functional Tester v6.1**.

During the Functional Tester installation, select any installation directory you like and accept all defaults. Note again that if you intend to use Visual Basic.NET as a scripting language, Visual Studio.NET must be installed on your system prior to installing Functional Tester.

Install and/or configure the Java Runtime Environment

Functional Tester creates logs to inform you of what has transpired during test

execution. By default, these logs are HTML documents. However, to provide additional information, Functional Tester launches an application called the *comparator* to highlight any differences between expected and actual results. For Functional Tester to run the comparator, the Java Runtime Environment must be installed and configured.

Is the Java Runtime Environment already installed on my machine?

It's quite possible that you already have a Java Runtime Environment (JRE) on your machine. If you do, you'll find a Java configuration icon in your control panel. Select **Start > Settings > Control Panel.** If you see a Java Plug-in option in the list of applications, then the JRE is installed on your machine. You do not need to install a second JRE.

Installing the Java Runtime Environment

The Java Runtime Environment is available for download from Sun (see Resources for a link). Select the appropriate link for your environment. Most users select the Windows (US English Only) JRE link.

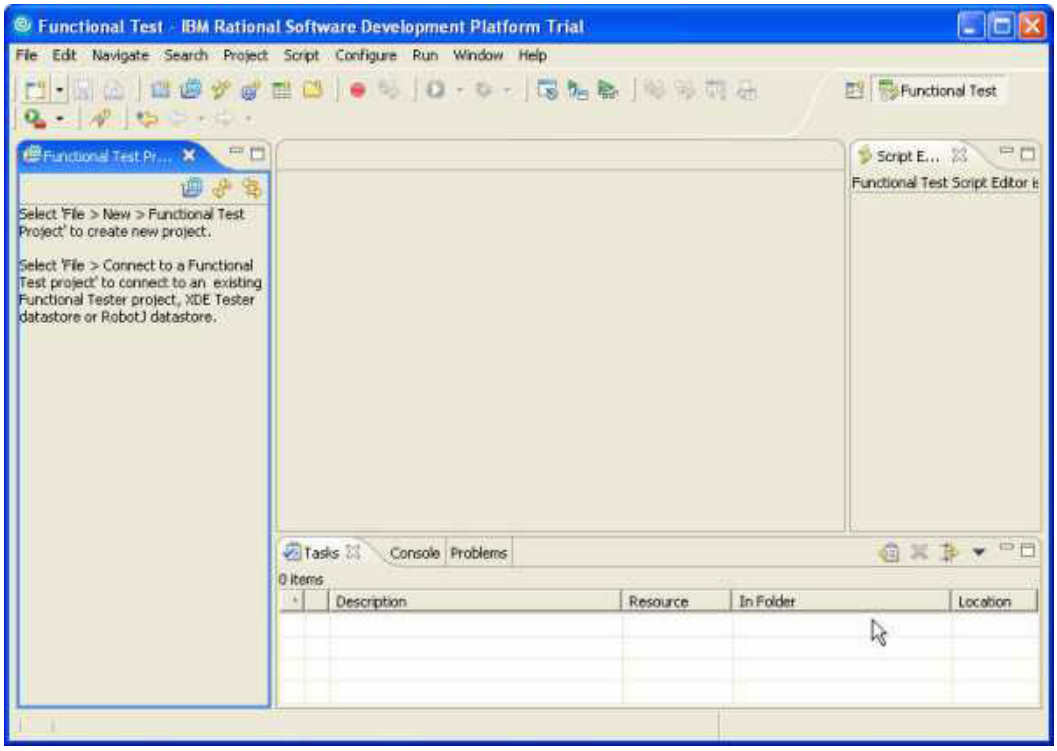
Create a test project

A *test project* is the location where Functional Tester keeps all of your scripts, along with expected and actual results. The format of the project is dictated by the environment you are using -- either Eclipse for Java scripting or Visual Studio.NET for Visual Basic.NET scripting. This is the only step in this document that is different for the two environments. If you're using the Java language, proceed to Create a Java test project. If you're using Visual Basic.NET, proceed to Create a .NET test project.

Create a Java test project

Complete these steps only if you are going to use the Java scripting environment.

1. Start Functional Tester by selecting **Start > Program > IBM Rational > IBM Rational Functional Tester v6.1 > Java Scripting.**
2. When the Workspace Launcher window opens, accept or change the default location for your workspace. This location is where Functional Tester will store all of your assets. Click **OK** to confirm your choice.
3. When the Rational Software Development Platform opens, close the "Welcome" tab by clicking on the X.
4. You are now in the main editing environment for Functional Tester.



5. Select **File > New > Functional Test Project** to create a new project.
6. Name the project `RFT Tutorial` and click **Finish** to create your project.
7. If you are using the evaluation version of Functional Tester, a window appears telling you how many days remain on your trial license. Click **OK** to acknowledge the message.

Create a .NET test project

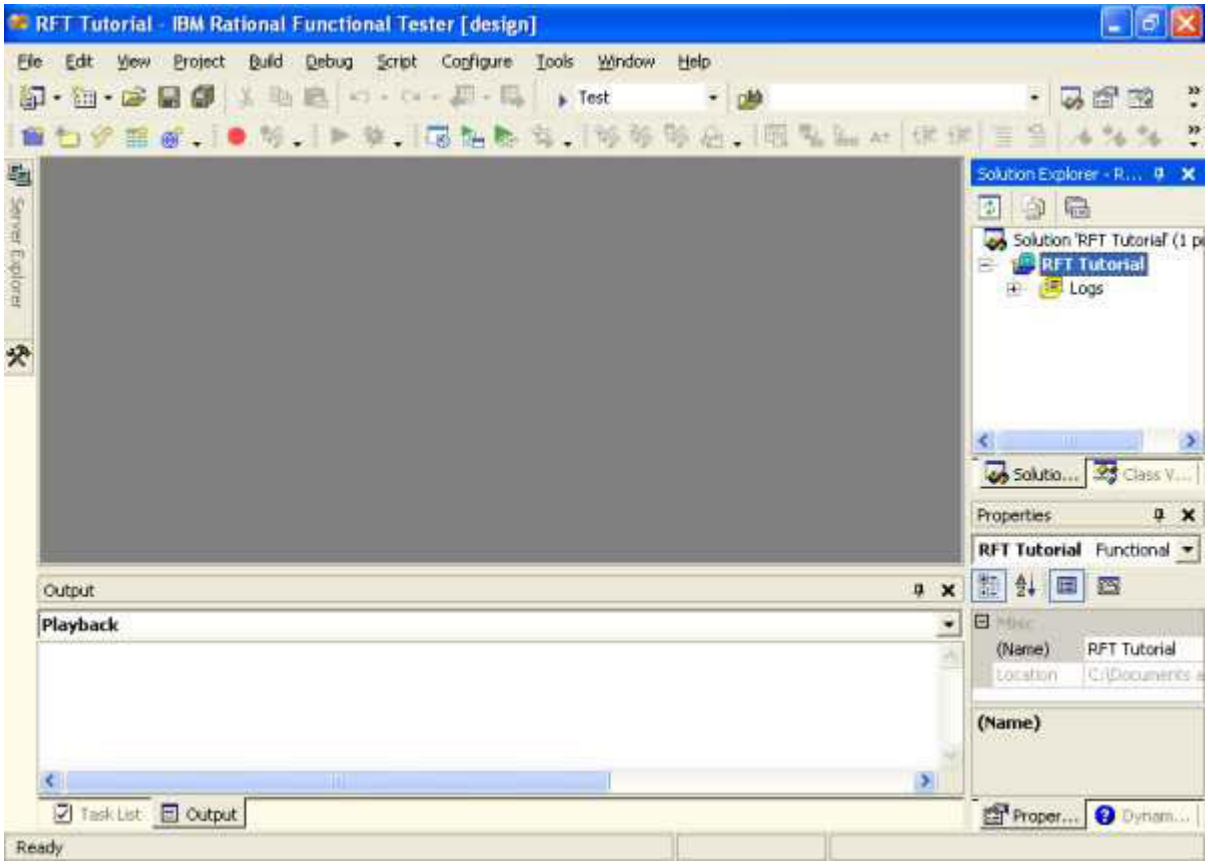
Complete these steps only if you are going to use the VB.NET scripting environment for this tutorial.

1. Start Functional Tester by selecting **Start > Program > IBM Rational > IBM Rational Functional Tester v6.1 > VB.NET Scripting**
2. If this is your first time opening Visual Studio, a start page appears. Close this page by clicking the X icon in the top right corner.
3. In the Visual Studio.NET environment, select **File > New > Functional Test Project** to create a new project.
4. Name the project `RFT Tutorial` and click **OK** to create your project.
5. If you are using the evaluation version of Functional Tester, a window appears indicating how many days remain on your trial license. Click **OK**

to acknowledge the message.

6. A Source Control window appears with the message, "To ensure optimal source control integration with Visual Studio .NET, check with your source control provider for compatibility and update information." Select the **Don't show this window again** check box and then click **OK** to close this window.

You are now in the main Functional Tester with VB.NET scripting environment.



Section 4. Create your test

Overview

You're about to record, edit, and play back your first Functional Tester script. You'll be using the Classics Online sample application that was installed with Functional Tester. This application allows users to browse and place orders for music CDs. There are two versions of the application, Build A and Build B. Having two builds will allow you to perform true regression testing. You'll record your test against Build A

and execute it against Build B.

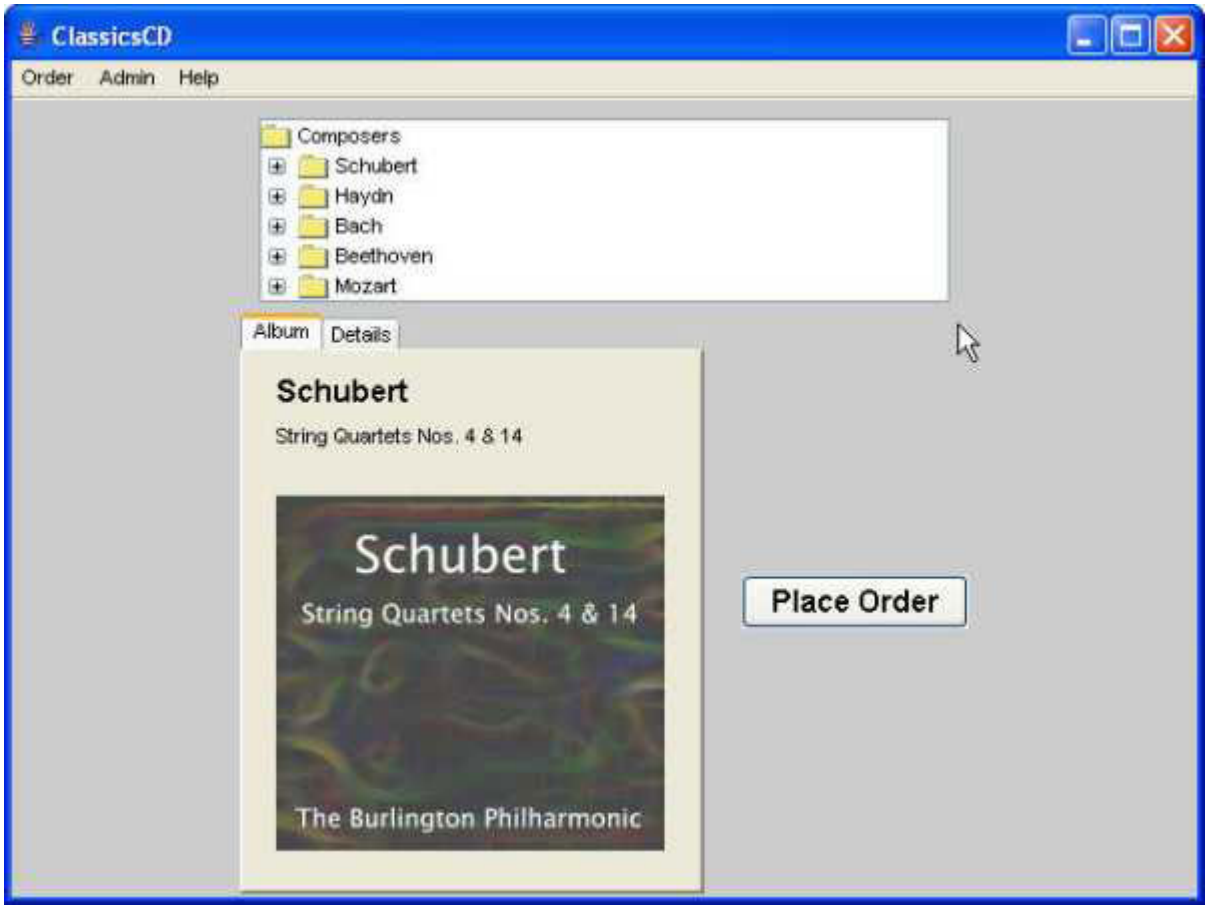
Start recording and launch the application

When you record scripts against a given build of an application, you're investing in future testing efforts. As soon as the next build becomes available, you can play back your scripts and test the new build in a fraction of the time that it would take to perform the tests manually.

In this section, you'll record a script that places an order for a CD. Your script will validate both static and dynamic data. It will also be *data driven,* meaning that it will be able to repeat the same steps for multiple test data sets. Once complete, you'll have a working test script that you can edit and execute.

1. Start your recording by selecting **Script > Add Script Using Recorder...** You can also click on the red record button on the toolbar.
2. When prompted for the script name, enter `PlaceOrder`.
3. Click **Finish** to start recording. The scripting environment minimizes and the recording toolbar appears.

4. Click **Start Application** on the recording toolbar. It is the third icon from the left, and looks like a window behind a green triangle.
5. The Start Application window appears. Select **ClassicsJavaA - java** and click **OK** to launch the application.
6. The ClassicsCD application opens. You should see a tree view containing CDs available for purchase, and a tab view below to display the selected CD. Schubert is the default CD. The application should look like this:



Place an order for a CD

To place an order for Bach's Violin Concertos CD:

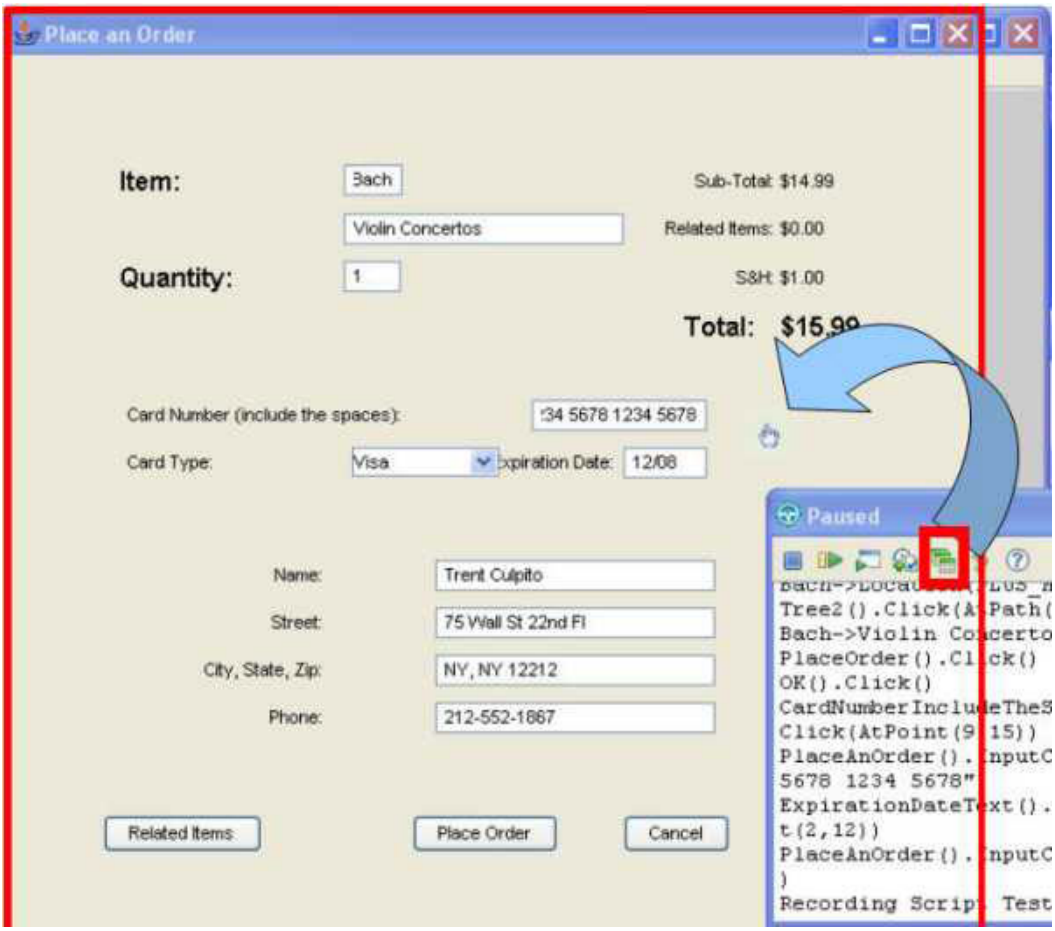
1. Click to expand the Bach folder.
2. Select the **Violin Concertos** CD.
3. Click **Place Order** to order the Violin Concertos CD.
4. On the Member Logon window, click **OK** to log in as the default customer, Trent Culpito.
5. When the Place an Order window comes up, enter the credit card number `1234 5678 1234 5678`.
6. Leave the card type as Visa and enter `12/08` as the expiration date.

Identify fields for data-driven testing

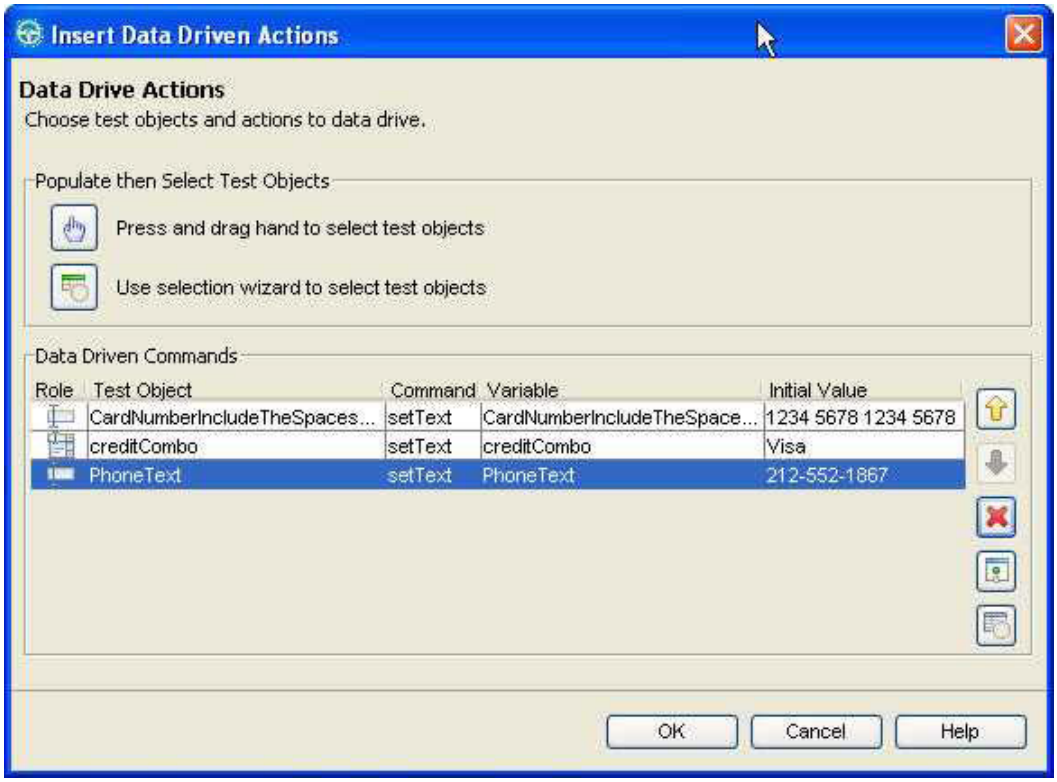
As you may have surmised, over the last few panels Functional Tester has been

capturing all your keystrokes and mouse actions into your script. The test automation best practice of data-driven testing separates keystrokes, or test data, from mouse actions. In this way, you can use one script for multiple data sets, as you'll see in this panel.

1. From the Recording toolbar, drag the Insert Data Driven Commands icon over the order form, so that the entire form is encased in a red square.



2. The Insert Data Driven Actions window opens. By default, Functional Tester assumes that you want to data drive, or separate, all test objects or fields. In this case, you just want to data drive the credit card. That allows you to use the same script to run tests with multiple credit card numbers. To delete the unnecessary test objects, highlight each one and click on the red X icon to delete. Do this for all values *except* CardNumberIncludeTheSpaces, creditCombo and ExpirationDateText.



3. Click **OK** to confirm your choices.

Perform a verification point on dynamic data

In this section, you'll confirm your order and you'll receive an order confirmation number. This number is dynamic, meaning that it will be different for every order placed. How will Functional Tester be able to validate this data if it cannot know beforehand what a valid response will be?

The answer is *patterns.* Using pattern matching technology, you'll perform a dynamic data validation that will ensure that a valid order confirmation message is received.

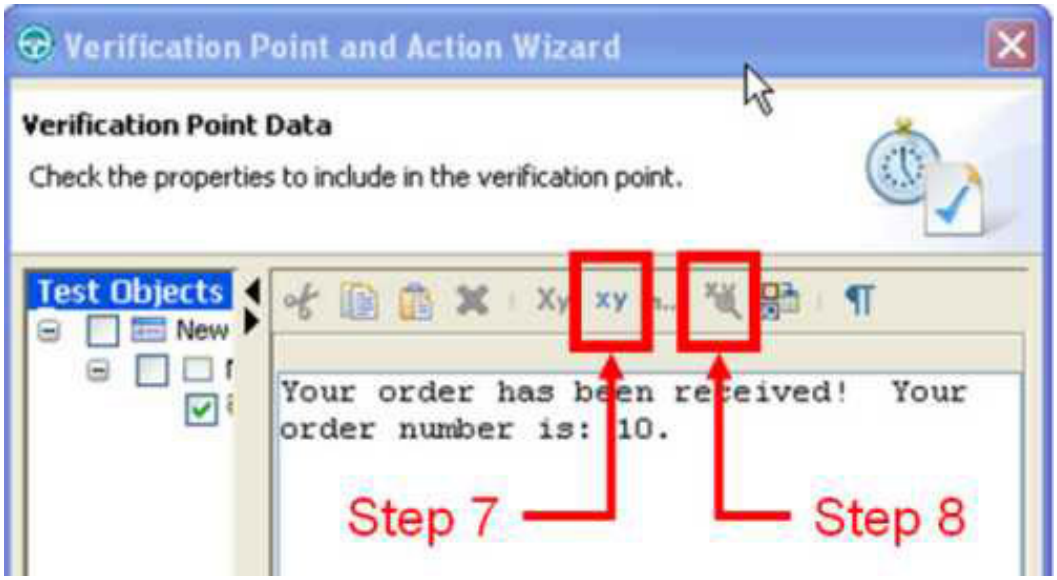
1. Start by clicking **Place Order** to place your order.
2. You receive an order confirmation window that reads: "Your order has been received! Your order number is 10." (Note that your order number is likely to be different than 10.)
3. Drag your Verification Point and Action Wizard from the recording toolbar -- it's the fourth icon from the left -- onto the message so that you highlight the message text. You should see something like the following figure:

4. Release the mouse button. This brings up the VerificationPoint and Action Wizard.

5. Click **Next** to accept the default action, **Perform a Data Verification point.**

6. Again, click **Next** to accept the default data value, **Label Visible Text.**

7. Click the Convert Value to Regular Expression icon, which is the xy icon above the message text.

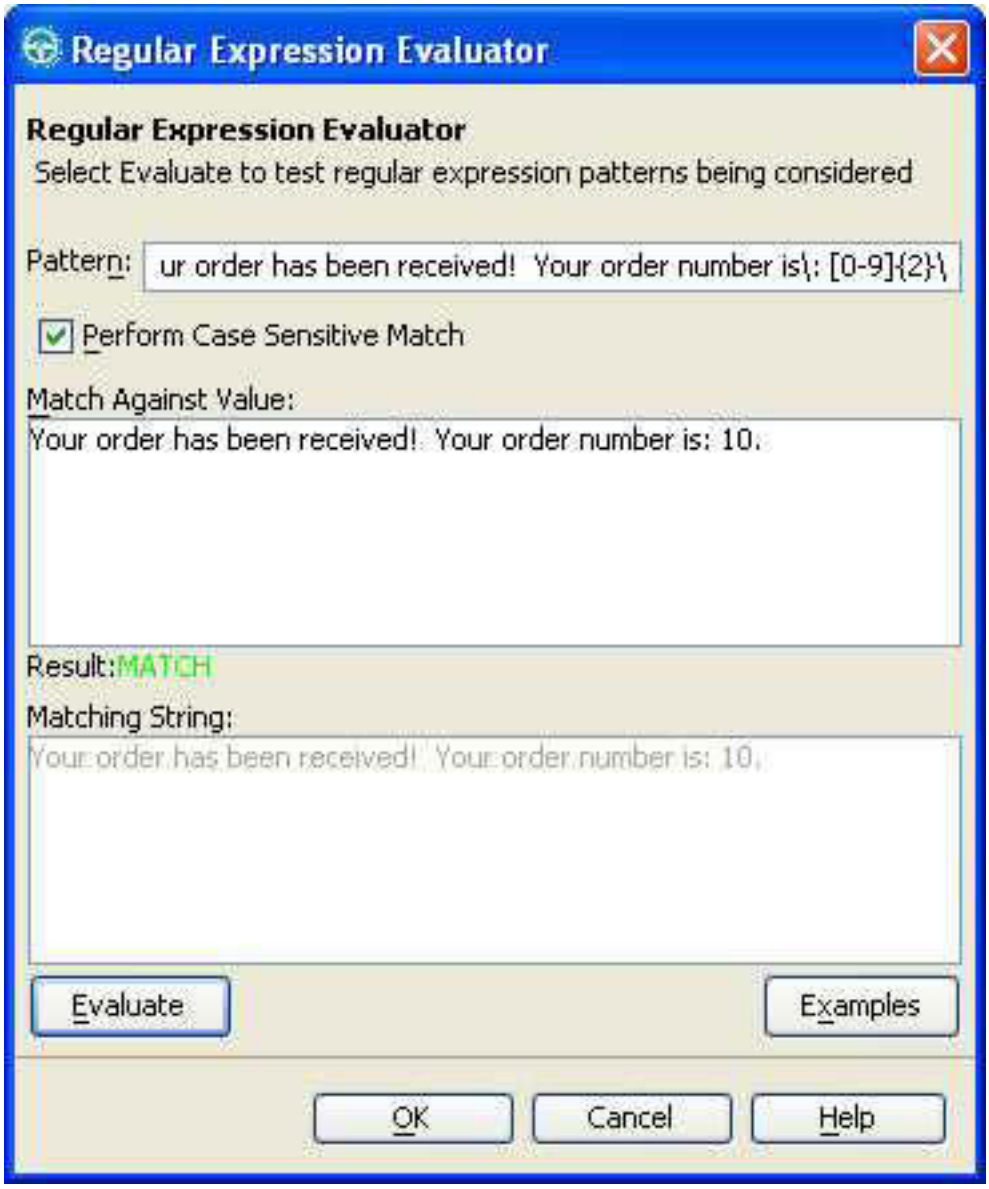


8. Click the Evaluate Regular Expression icon. This is the xy icon with the magnifying glass.

9. Regular expressions are a universal notation for pattern matching. The regular expression evaluator lets you test out your regular expressions to make sure you've got it right. In the Pattern field of the Regular Expression evaluator, replace the number 10 with the regular expression `[0-9]{2}`. Translated into plain English, this regular expression reads, "Match any two-digit combination of zeros and nines, or the range 00-99."

10. Click **Evaluate** to test your regular expression. You should see the text

Match in green show up in the Result field, as illustrated below:

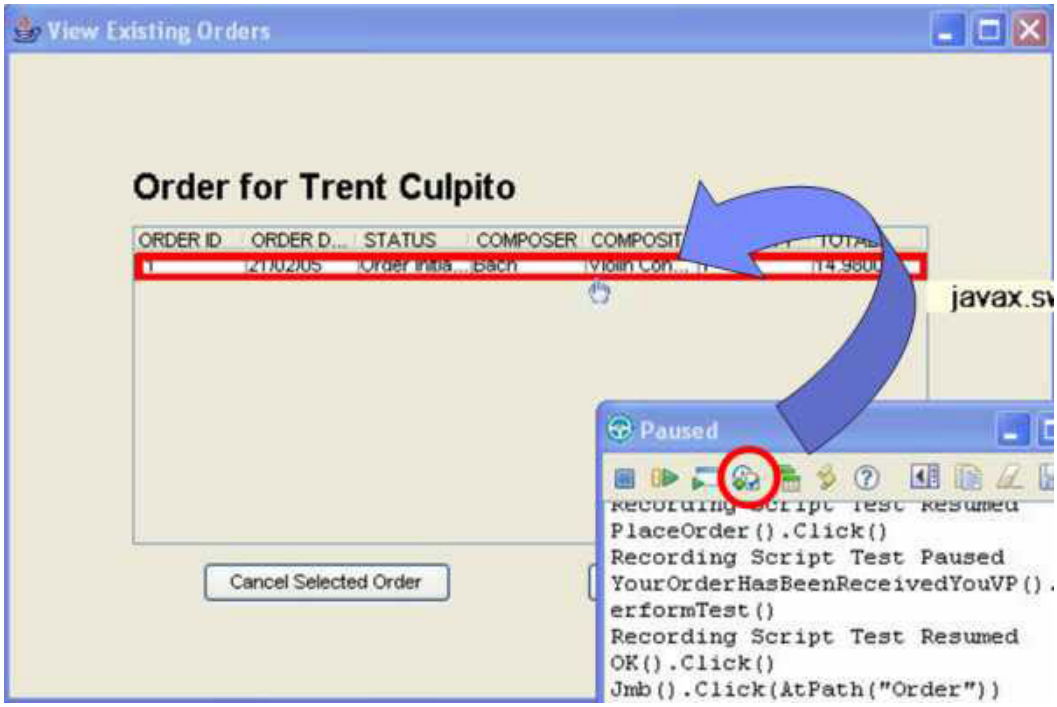


11. To further test your regular expression, change the value 10 in the Match Against Value section to any other number and click **Evaluate.** If you have done things correctly, you should get a match for any number between 00 and 99, but not a match for any number outside that range.
12. When you're done testing, click **OK** to close this window.
13. Click **Finish** to close the Verification Point and Action Wizard.
14. Click **OK** on the message window with the order confirmation.

Perform a verification point on static data

Your last step is to actually confirm that the order went into the database. To do this, bring up a table of orders and perform a static verification point on the table using the following steps:

1. Select **Order > View Existing Order Status...**.
2. On the View Order Status window, click **OK** to log in as Trent Culpito again.
3. With the order displayed, click and drag the verification point and Action Wizard onto the order information, so that the order is encased in a red square, as illustrated below.



4. On the first panel of the Verification Point and Action Wizard window, click **Next** to perform the default data verification point.
5. In the Data Value drop-down menu on the next panel of the wizard, select **Table Contents** and click **Next.**
6. Finally, review the data in the wizard and click **Finish** to store this information as a baseline result. At playback time, Functional Tester will compare what the application presents with this baseline and highlight any discrepancies in the test log.

Shut down the application and stop recording

To shut down the application:

1. Click **Close** to close the View Existing Orders window.
2. Close the Classics application by clicking on the X in the top right corner of the application window.
3. Click the stop recording button on the Recording toolbar. It is the leftmost button and looks like a square.

Congratulations -- you have just created your first automated test script!

Section 5. Review the script

Overview

So far you've created a test that you'll be able to re-use against all subsequent builds of the application. With every new build of the application and every run of the test, you'll be increasing your ROI on the test. Your test not only validates static and dynamic data, but has been built for data-driven testing. Your ROI will further increase with every dataset that you add to the script.

A test is comprised of three major components: the script, the object map and the datapool. Each of these components is examined in this section.

Examine the script

If everything has gone according to plan, you should now be looking at a script like this:

```
public void testMain(Object[] args)
{
    startApp("ClassicsJavaB");

    // Frame: ClassicsCD
    tree2().click(atPath("Composers->Bach->Location(PLUS_MINUS)"));
    tree2().click(atPath("Composers->Bach->Violin Concertos"));
    placeOrder().click();

    // Frame: Member Logon
    ok().click();

    // Frame: Place an Order
    cardNumberIncludeTheSpacesText().click(atPoint(35,11));
    placeAnOrder().inputChars("1234 5678 1234 5678");
    expirationDateText().setText(dpString("ExpirationDateText"));
    placeOrder2().click();
    YourOrderHasBeenReceivedYouVP().performTest();

    //
    ok2().click();

    // Frame: ClassicsCD
    jmb().click(atPath("Order"));
    jmb().click(atPath("Order->View Existing Order Status..."));

    // Frame: View Order Status
    ok3().click();
    existingTable_contentsVP().performTest();

    // Frame: View Existing Orders
    close().click();

    // Frame: ClassicsCD
    classicsJava2(ANY,MAY_EXIT).close();
}
```

Java

```
Public Function TestMain(ByVal args() As Object)
    StartApp("ClassicsJavaA")

    ' Frame: ClassicsCD
    Tree2().Click(AtPath("Composers->Bach->Location(PLUS_MINUS)"))
    Tree2().Click(AtPath("Composers->Bach->Violin Concertos"))
    PlaceOrder().Click()

    ' Frame: Member Logon
    OK().Click()

    ' Frame: Place an Order
    CardNumberIncludeTheSpacesText().Click(AtPoint(6, 14))
    PlaceAnOrder().InputChars("1234 5678 1234 5678")
    ExpirationDateText().Drag(AtPoint(22, 12), AtPoint(21, 12))
    PlaceAnOrder().InputChars("12/08")
    ' Data Driven Code inserted on Feb 21, 2005
    CardNumberIncludeTheSpacesText().SetText(DpString("CardNumberIncludeTheSpacesText"))
    CreditCombo().SetText(DpString("creditCombo"))
    ExpirationDateText().SetText(DpString("ExpirationDateText"))
    YourOrderHasBeenReceivedYouVP().PerformTest()

    '
    OK2().Click()

    ' Frame: ClassicsCD
    Jmb().Click(AtPath("Order"))
    Jmb().Click(AtPath("Order->View Existing Order Status..."))

    ' Frame: View Order Status
    OK3().Click()
    existingTable_contentsVP().PerformTest()

    ' Frame: View Existing Orders
    Close().Click()

    ' Frame: ClassicsCD
    ClassicsJava2(ANY, MAY_EXIT).Close()
End Function
End Class
```

Visual Basic.NET

Your script is standard Java or Visual Basic.NET code. If you look closely, you'll see that the scripts are remarkably similar. In fact, the main difference between a Java and a .NET script is that the Java statements end with semicolons, while the .NET statements do not.

Your script follows an `Object.Action` format. Let's examine a couple of statements:

```
PlaceOrder().Click()
```

This statement tells Functional Tester to perform a Click action on the PlaceOrder button.

```
PlaceAnOrder().InputChars("1234 5678 1234 5678")
```

This is the statement that tells Functional Tester to type the credit card number into the PlaceAnOrder screen.

```
YourOrderHasBeenReceivedYouVP().PerformTest()
```

This statement is the one that tells Functional Tester to perform the dynamic verification point on the order confirmation window.

Hopefully this panel has demonstrated that, while the script is in fact code, you don't need a PhD in engineering to be able to read and understand that code.

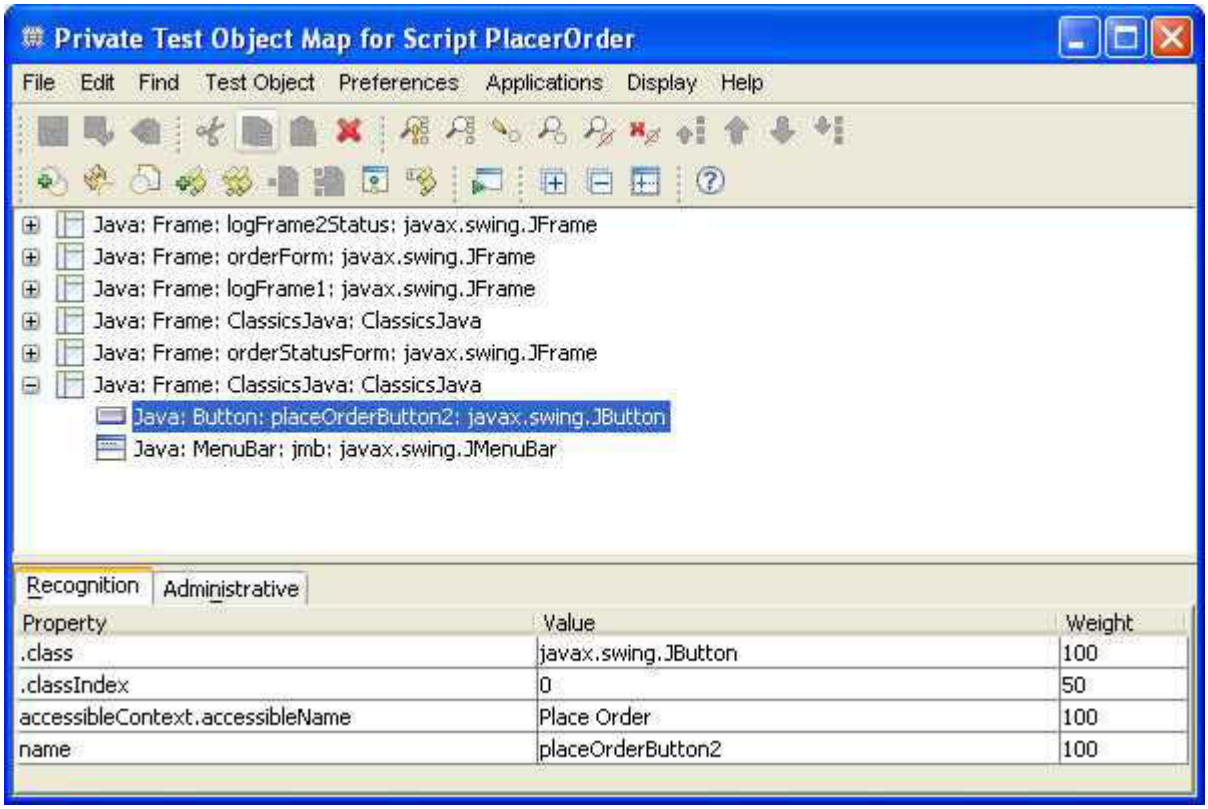
Examine the object map

To this point, the object map has not been discussed. The object map is where Functional Tester stores all of the information on the objects on the screen. Previously, you saw that the following command tells Functional Tester to click the PlaceOrder button:

```
PlaceOrder().Click()
```

How does Functional Tester know which button is the PlaceOrder button? It consults the object map. You can too!

To see what information Functional Tester has on the PlaceOrder button, double-click the PlaceOrder button. It is located in the **Test Objects** branch of the Script Explorer.



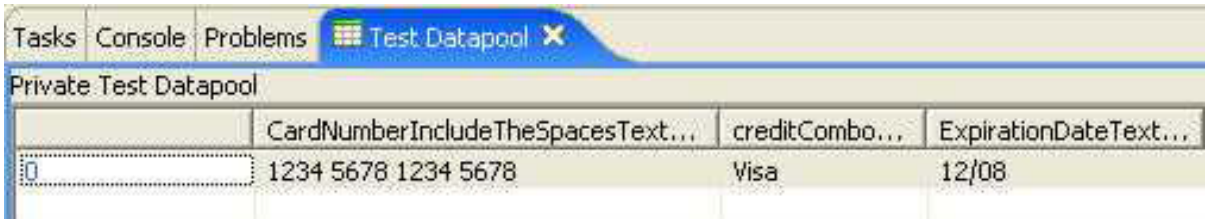
You can see that Functional Tester keeps four pieces of information to help it identify the PlaceOrder button at playback time: the class, classIndex, accessibleName, and name properties.

What's important here is that Functional Tester does not need to match all of these in order for it to identify the PlaceOrder button. Let's say the developers decide to change the name from placeOrderButton2 to placeOrder. For most automated testing tools, this change prevents the script from executing to completion. The tool is not able to find the object due to the name change and crashes. With Functional Tester, this is not the case. Functional Tester looks at all the properties, and no single property change disrupts playback. Functional Tester notices that the name has changed and logs that fact, but is able to continue, as it has matched the other properties and satisfied itself that the object now called placeOrder is close enough to the placeOrderButton2 object it is trying to find.

This technology is called *ScriptAssure,* and it ensures reliable script playback by making Functional Tester resilient to application changes. No single change to any object will prevent a Functional Tester script from running to completion.

Examine the datapool

The final component is the *datapool*. This is where Functional Tester stores all of the data to be used by the test. The datapool is located in the view under the script, and looks like this:



Many tests require the tester to perform the same repetitive steps, varying only the test data. Data-driven testing makes this possible, and Functional Tester makes it easy.

Datapools separate test data from test script code. You can use the same script code for multiple sets of test data, increasing the ROI on your script.

Let's do some data-driven testing:

1. Right-click anywhere within the test datapool and select **Add Record...**
2. In the Add Row window, select **Add After** and click **OK**.
3. A second row of data will appear for your test. Double-click the credit card number field and enter the card number `1111 2222 3333 4444`.
4. Leave the credit card type as Visa.
5. In the expiration date field, enter the date `06/07`.

Your datapool should now look like this:

You have now doubled the value of your script. By adding a new credit card, you've been able to use the same script code to test two separate credit cards. With a few more rows of data, you can get even more mileage. You can add a test for an expired card, an invalid card, an American Express card, a MasterCard, a stolen card ... you get the idea. Data-driven testing significantly increases the power of your scripts -- and you've been able to do it without writing a single line of code.

Section 6. Run your test

Overview

You've done a lot. You recorded a script that validates static and dynamic data. You made your script data driven, and included two datasets, thereby doubling your testing ROI on this script. It's now time to reap some benefits.

Execute the test

The real benefit of automation testing comes when you perform regression testing. Regression testing means executing your test against a subsequent build of your application, ensuring that what used to work, still works. That's what you're going to do here. To modify your script to run against Build B of the application and watch it run:

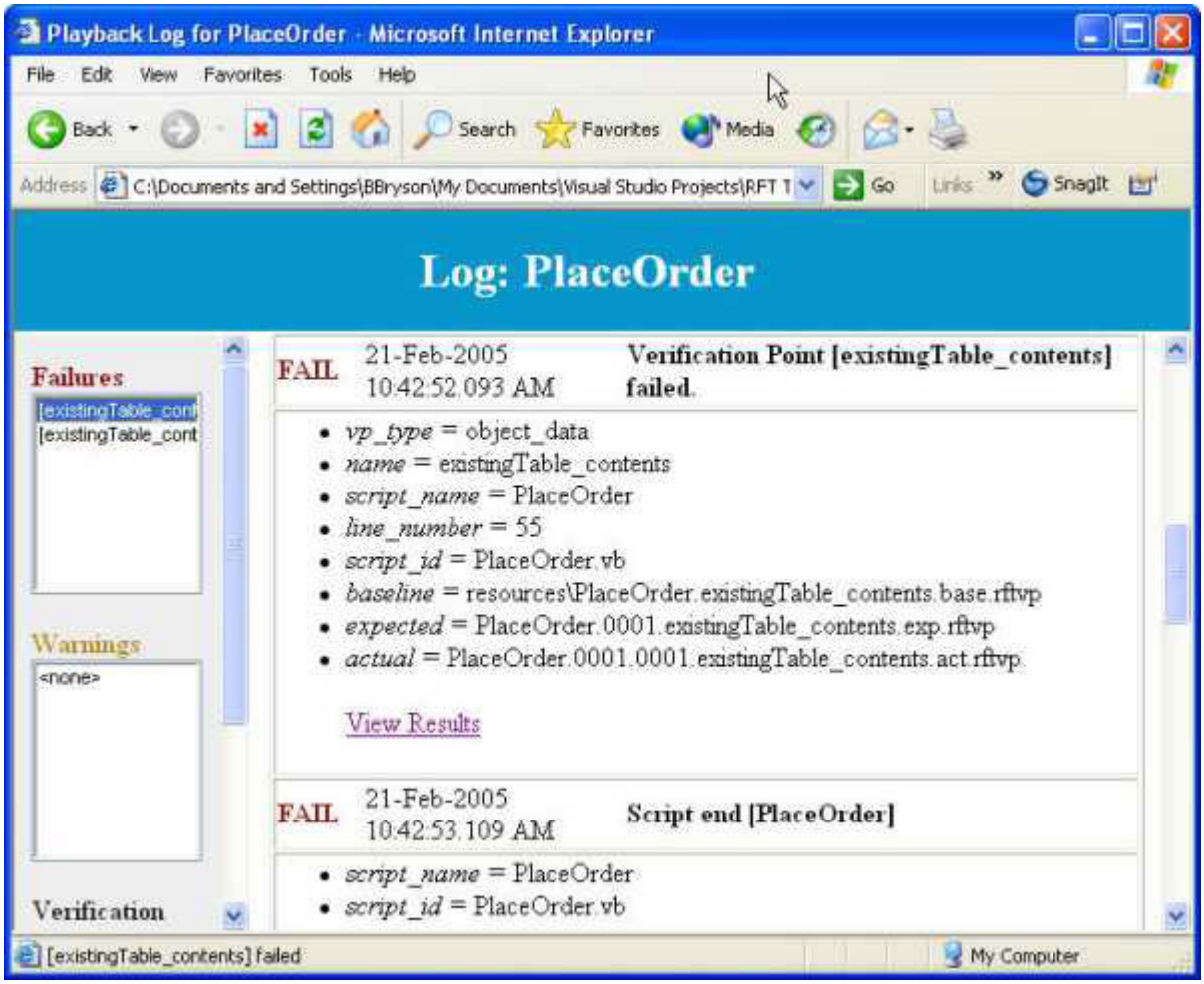
1. Modify the first line of your script, which reads `startApp("ClassicsJavaA");`. Change the A to a B, so that the line reads `startApp("ClassicsJavaB");`.
2. Select **Script > Run** from the menu bar to run your script.
3. On the IBM Rational Functional Tester window, accept the default log name of PlaceOrder and click **Next.**
4. In the Datapool Iteration Count field, select **Iterate Until Done.**
5. Click **Finish** to run your script.

As the script runs, take note of a couple things:

- The application layout has changed, but the script plays back. Layout changes do not impair Functional Tester's ability to execute a script.
- The credit card information is entered twice. The first entry simply plays back the numbers you typed when you recorded the script; the second is the entry from the datapool. If you want to, you can edit your script to remove the commands that enter the data you typed, as they are made redundant by the datapool data.

Examine the log

If all went well, you are now staring at a log of all the events that happened during playback. The log notes major events, such as the script and application launch and termination, as well as all verification points. If you have any script errors, or if any objects in the application are now different from their entries in the object map, those events are noted as well.



Note the links to the most important parts of the log on the left side of the screen. In the image above, I clicked on the first failure notification, which scrolled the log to show the first failure as the first item.

Click the **View Results** link for that entry to bring up the Verification Point Comparator.

The Verification Point Comparator highlights the differences between the expected and actual results. Here you can see that you expected to order Bach's Violin Concertos CD for $14.98, but instead got Schubert String Quartets Nos. 4 & 14 for $18.98.

You've found a regression error! Despite the fact that you most certainly ordered Bach's Violin Concertos -- the steps to reproduce are right there in the script -- you got Schubert's String Quartets by mistake. It's time to file a defect -- maybe even include the script, so that the developers will have the ability to reproduce the problem!

Section 7. Wrap up

Summary

You accomplished a lot in this tutorial:

- You recorded a data-driven script that can run multiple data sets through the same script code, increasing the ROI on your test script.
- You recorded a script that is resilient to application changes -- a script that ran without modification on two different builds of the same application.
- You validated static data using Functional Tester's verification point and

comparators.

- You validated dynamic data using pattern matching regular expressions.

And you accomplished all of this without writing a single line of code!

This is only the beginning. What you've done only skims the surface of what Functional Tester can do. With a little more work and a little imagination -- there's nothing Functional Tester can't do.

You now have the skills. Go give Functional Tester a try on your own application code, in your environment. You can do it!

Resources

Learn

- Check out the Rational Functional Tester page at IBM developerWorks.
- Learn more at the Rational Functional Tester page.
- Now that you've gotten started, you can improve your skills with additional Rational tutorials.

Get products and technologies

- Download a fully functional trial version of IBM Rational Functional Tester.
- The Java Runtime Environment is available as a free download from Sun.

Discuss

- Participate in the discussion forum for this content.
- Connect with other Rational users in the Rational Software Global User Group Community.

About the author

Brian Bryson
Brian Bryson joined IBM Rational in 1995 after having spent several years in software development. Since joining IBM Rational, he has held various positions supporting software quality tools, from consultant to technical marketing. He is currently the automated software quality evangelist for IBM Rational and spends his days speaking to customers, partners and analysts on all matters pertaining to software quality. Brian has spoken at numerous conferences and published many articles, the most recent on Patterns of Success in Test Automation in the December 2004 issue of Software Test and Performance.