Introduction to IBM Rational Functional Tester

Student Manual

RFT Version 9.5

Terms of Usage

Preface

This manual has been compiled from various sources as a companion to the course "Introduction to IBM Rational Functional Tester" provided to the IRS by ProTech Training on Jul 29-30, 2019.

This manual is not intended as a comprehensive guide to the Rational Functional Tester (RFT) product but is intended to supplement the instructor presentation and the hands on exercises provided during the course.

Much of the content found in this manual can be found at various IBM websites and in assorted IBM documents on line; however, since the classroom environment, in particular the laptops provided for the class, does not have Internet access, those materials are consolidated in this manual.

Course Objectives

The overriding goal in in presenting this course is to follow the principle of pragmatics: specifically, whatever you learn in this course should either be directly applicable to the work you do as a tester or analyst, or give you some insights or understanding that improve your competence and performance in either of those roles.

When teaching a “how-to” course for a specific technology product, this translates into some specific objectives:

1. Having you master the product in terms of becoming familiar with its functionality and its interface.
2. Understanding how the product can and should be used, as well as the kinds of tasks that it is intended to automate. The two ways we can fail to get a product to work effectively for us is (a) either not use it for tasks it is intended to automate, or (b) trying to use it for tasks that it was never intended for.
3. Learn how to problem solve using the tool, specifically how to use it in a novel or new situation or task.
4. Learn now to trouble-shoot when the product does not seem to be working the way you expect it to.
5. Know where to get more help and information about the product when you need it.

What doesn't meet these objectives is having you sit and follow a few canned examples step by step with out really understanding what you are doing. What does work is trial and error or engaging the product and learning how to bend it to your will. This means a lot of figuring things out as you you, the same way you learn to to anything, you read the instructions (well, sometimes) and then try it out by just using it.

As a result, the way we will do the class is follow the following sequence of activities for the hands on portions.

1. The instructor will demonstrate how to perform specific tasks with you following along. This may be done several times. During this phase, it is important for you to understand not just what the instructor is doing but also why they am doing it, and what is happening in the tool.

2. The instructor will repeat step one several times, stopping to answer questions whenever someone has one.

3. You will try to replicate what the instructor did on your own with the instructor monitoring everyone's progress to see how you are doing and clarify anything you are not clear on.

4. The basic exercise will be documented in the lab section of this manual but the instructor may suggest a related new problem, or variation on the demonstration, for you to try on your own, using the tool to solve the problem. Again, the instructor will monitor everyone to see how people are doing and answer any questions you may have.

5. If requested, the instructor will walk through a solution to the problem that you can compare to your own solutions and have a discussion as to why they may be different.

In addition, there will be the standard step by step sorts of exercises common in this course – taken from the IBM tutorials for the product – for those who feel more comfortable with this approach. These can also be used after the course as follow up exercises or refreshers.

One the measures of success for this course is not how much material is presented, but instead is how much material you learn, and how much of that is usable when you leave this class and head back into the real world.

How well we do in accomplishing the second of those can be defined in modest terms. What could be considered success is what benefits you get personally from the course. If this course can help you save 20 minutes a day by showing you a faster or easier way to do something, or help you eliminate a stressful task by giving you alternatives, or just in general makes the little day to day things you have to do go faster, smoother or better, then this course is a success.

And Your Responsibilities

There are a number of ways that you can make sure this class has value for you:

1. *Be present.* Just being physically in class is not enough, you need to be engaged with the material and the activities. If you are trying to do two things at once – being in class and working on other things, neither will be done well. Choose one and commit to it; either be 100% in class or leave and go take care of the more pressing matters.
2. *Contribute.* Think of this as a seminar. If something is not clear, or you disagree with the instructor on some point, speak up. Ask questions. Add your own observations and insights. The class will be better when we can use the wealth of knowledge and experience that the students bring to the class.
3. *Be courteous and respect the learning environment for your fellow students. B*e prompt coming to class so your peers are not waiting for you to show up. If you know you are going to be late or absent, let the instructor know so they won't hold up the class waiting for you. You are responsible for you own comings and goings but please do it in a manner that is discrete and unobtrusive.
4. *Make sure that you have any prerequisites done before you come to class.* I will be assuming that you have met certain criteria to be here in class and will not be doing any remedial or catch up sessions.
5. *Enjoy yourself.* Learning takes place most efficiently when you are having fun during the learning process. I plan to have fun so I encourage you to do the same.

Most of this is common sense but I want to make sure that you have every opportunity for not just a productive and useful class, but also for a good fun week that you will enjoy and remember positively.

Introductions

At the start of the class, the instructor will ask you for three items of information.

1. *What your job is here at the IRS.* We don't need a job description but just a quick sentence or two about your job so that the instructor can get a sense of how this material will relate to what you do back in the real world.

2. *You background and experience in software testing and working with requirements.* This gives the instructor a quick measure of what sort of skill level everyone in starting off with so they pitch the level of presentation accordingly.

3. *What you want this class to do for you.* You may have some very specific things you want to learn in this class. Knowing this in advance allows the instructor to ensure that when we get to any part of the course that is relevant to that need, they are aware of what you are looking for and can make sure to meet your learning objectives. If you don't have any specific needs, then that is perfectly okay, but if at any time during the course you do realize you have some thing specific you want to cover, please feel free to let the instructor know.

What the Course is About

This is an entry level course designed to introduce IBM Rational Functional Tester (RFT) to both those who have used the tool in the past and those who may not have had any exposure to the tool in the past. RFT is an automated functional testing and regression testing tool designed to supported automated test- ing for functional, regression, GUI, and data-driven testing.

Interface testing is one of the most tedious and error prone forms of testing, as well as being quite labour intensive. RFT allows the automation of testing scenarios by being able to record test scripts and tests that can be rerun to allow identical executions of the same testing sequence of steps. The tool is highly customizable to allow for configuration over many variations of the testing process and test scripts. Especially useful in regression testing, the tool allows identical test scenarios to be run on the before and after versions of an interface or GUI, a critical aspect of maintaining quality control in a regression test environment. At least two thirds of the class consists of hands on walk throughs, demonstrations and lab activities.

The course starts with an overview of the sorts of problems that RFT is intended to address, and looks at the risks and costs of manually performing those tasks. This followed by a walk through of the RFT interface and work-along demonstrations of the main RFT functionality which is then followed by hands on labs where students learn to master specific features of the tool. This cycle of walk through, follow along demo and then lab is repeated for all the other features of RFT.

Course Objectives:

• Describe the function and purpose of Rational Functional Tester

• Navigate the Rational Functional Tester interface

- Record automated scripts
- Play back automated scripts
- View and analyze results
- Modify scripts to extend the capability to test the application
- Use test object maps
- Control object recognition
- Create data-driven tests and use datapools

Main Topics Covered

- Record and playback process - Creating resilient scripts with Simplified scripting.
- Test object maps, Object recognition, Recognition scores, Pattern-based recognition
- IBM Eclipse interface (perspectives, script debugging, and editing)
- Performing user actions with a script (including verification points)
- Extending scripts with script support features
- Layout and structure of a script
- Logs and logging options
- Datapools and external data sources

Introduction to Rational Functional Tester

Module One

1 Introducing Rational Functional Tester

Rational Functional Tester (RFT) is designed to fill a specific automation role in the testing process; specifically, RFT gives testers the capability to automate interactions through a GUI that that mimic the actions of a human user.

The man role of RFT is to allow software quality teams to perform reliable and accurate automated regression testing and configuration testing.

Versions

At the time this material was prepared in July of 2019, the current operational version of RFT is 9.5, although version 10 is due for release.

RFT originally started as a GUI test tool that was acquired by Rational Software in 2002 and became known as "Rational Robot." Several years later when Rational Software was acquired by IBM, the tool went through a substantial revision and was released as Rational Functional Tester.

RFT has continued to evolve with major changes through the various releases (starting with version 6.0 in 2005) with a series of enhancements and improvements in the code base, the product functionality and stability, as well as its ability to integrate with other IBM and non IBM products (such as Jenkins and HP Quality Center).

Three of the major changes made to the product were; the migration of the proprietary scripting language to Java and Visual Basic based scripting, a new model for referencing GUI objects and the migration of RFT to the Eclipse platform.

Integration

Currently, RFT is integrated with two other tools, the Rational Integration Tester and the Rational Performance Tester as part of the Rational Test Workbench. However, in this course will only be looking at RFT and will leave the other two tools to be dealt with in other courses.

RFT is also intended to integrate with the IBM Rational Team Concert Platform, specifically the Rational Quality Manager which is used to organize tests and test assets; however, integrating RFT with any other platforms or products is optional.

For this class, we will be using RFT in a standalone environment.

1.1 The RFT Architecture

RFT has three different modes of operation:

1. **Recording Mode:** The manual operations performed by the user are recorded as a script.
2. **Playback Mode:** A previously executed script is played and executed.
3. **Edit Mode:** Where modifications can be made o the script or environment – this is the normal mode of the tool.

We will explore working in each of these modes in the following sections. There are two kinds of scripting available: Java Scripting and Visual Basic scripting. This course will only focus on Java scripting to avoid being overly complex, but the ideas and techniques covered in this class apply to both scripting environments.

1.1.1 Test Assets

RFT is a standalone application where all test assets are stored in local files, although they can be stored in various version control systems, such as git and ClearCase, as well. Maintaining assets in a version control system is considered a best practice but is out of scope of this course.

Because we are using Java scripting, our RFT projects are a special type of Java Eclipse project. If we were using Visual Basic scripting, then the RFT project would be stored as a special type of Visual Basic project in Microsoft Visual Studio. The type and nature of the assets in both environments are almost identical.

Because all of the test assets are created, edited and maintained through the Eclipse perspective, no knowledge of the structure or format of the test asset files is needed to use the product effectively.

1.1.2 Test Results

Test results are stored in test logs which can kept in a variety of different formats. Since the choice of test logs depends on how we want to manage and store the results, usually in a test management system, we will stick with simple log formats in this course. Some of the test log formats available are:

1. HTML
2. Text
3. Test and Performance Tools Platform (TPTP)
4. XML
5. Rational Quality Manager
6. Json

1.1.3 Understanding Scripts

Scripts in RTF are created in RFT by using the test recorder. We will create a simple script and play it back in lab #1.

Test are created by putting RFT into recording mode and starting an application. Every time a key press or any action with a mouse (excluding moving a mouse pointer) is executed by the user, it is recorded as a step in the test script.

To ensure robustness, RFT does not record itself nor does it record any actions on applications that are not enabled. Even making mistakes, like typos, and then correcting them will be recorded as part of the script.

Test script are created on the fly during recording and can be viewed as they are being created. Various information needed about the test script environment such as test objects, verification points, and test data are stored as additional asset files but are accessible as test assets in the script. When a script is being recorded, RFT creates something called an "object map" which records certain kinds of information about which objects the user is interacting with.

The actual test scripts are not pure Java, but are extensions a special ft.script Java package which means they shhould be executed through RFT so that the required Java packages and libraries can be set up and initialized properly.

1.1.4 Executing Scripts

Tests are executed by using the playback mode of RFT. In playback mode, all of the keyboard and mouse actions that are recorded in the script are executed. However, playback mode does not disable the keyboard or mouse which means it is possible to add spurious events into the playback that might corrupt or cause the script to crash. However, being able to interact with tests can be useful in diagnosing and debugging problems with script playback.

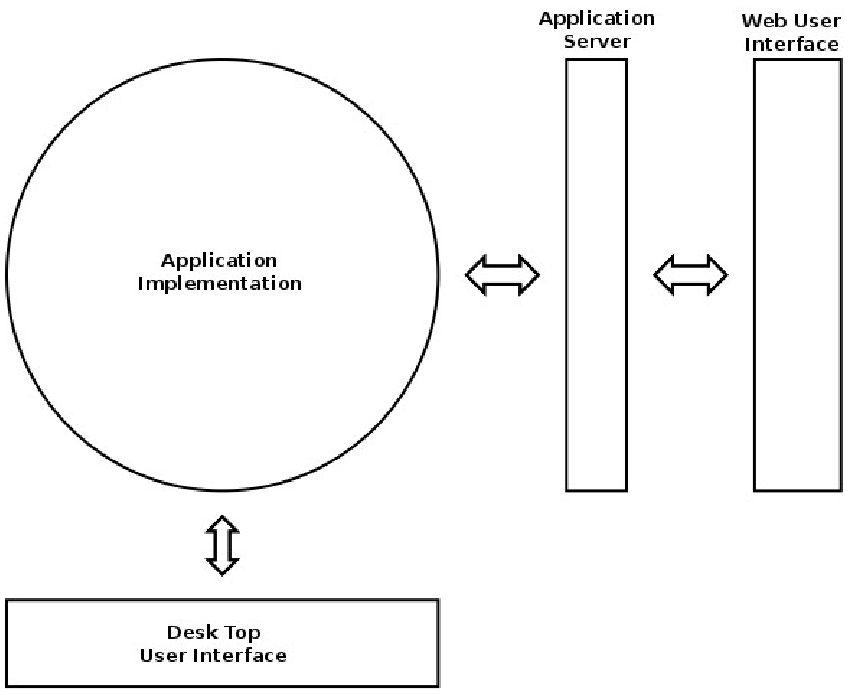
When playing back a script, RFT has to find each object it has to operate on by matching the descriptions of objects in the script's saved test object map against the actual objects that are present at run-time. If RFT cannot find a close enough match, it logs an error and either attempts to continue or aborts. If a matching object is found, RFT performs the action specified in the script on the object.

Actions generally refer to any measurable interaction a user might have with the application interface via either the mouse of keyboard; however the action may be something added to the test to be performed on the object, such as comparing an expected value with an actual value at a defined verification point.

While the script itself executes as if a user were interacting with an application, the important feedback for a test are the results of verifying and validating via test points that the playback is proceeding as it should.

1.2 Defining the Problem

RFT is designed to fill a specific automation role in the testing process. In order to understand this role, we need to develop a few background concepts. The design of RFT assumes that we are following some sort of standard application architecture that looks like the following:



The basic parts of this architecture are:

1. *The Application Implementation:* This is where the application logic takes place – the actual application that does the work. It may be on a main frame or running as module executing on some host, a web application or any of a number of other forms.
2. *The User Interface:* For each application, there may be a number of different interfaces through which the use interacts with the application. The interfaces by be through a web page, a mobile application or some sort of desk top application.
3. *A Connector:* This is an optional step and is some sort of component that allows a user interface to talk to the application, a web application server is an example of such a connector.

Users do not access the implementation directly, instead they interact with a UI which in turn accesses the application, sometimes through some sort of connector application. For the rest of this course we will assume that the connector application can be considered part of the user interface, just to to make the rest of this discussion simpler – this assumption will not have any impact on what we are covering in terms of our course objectives.

Essentials of IBM Rational Functional Tester

Rational Functional Tester can be used to test two different things:

1. The correctness of the functionality of the underlying application when accessed through a user interface. This is usually called *acceptance testing.*
2. The correctness of the interface itself. This is normally called *user interface testing.*

Any testing activity should clearly identify which of these is the test objective. The actual mechanics of using RFT are similar for both kinds of testing, but the test plan should clearly identify exactly what the test purpose is and what the test is expected to reveal.

No test execution should be planned to test both the interface and underlying application unless it is a final integration test.

While the same script can be used in multiple testing plans for different for different test objectives, the purpose of the test execution should be clearly delineated. We will not deal with this topic further since the planning process is beyond the scope of this course.

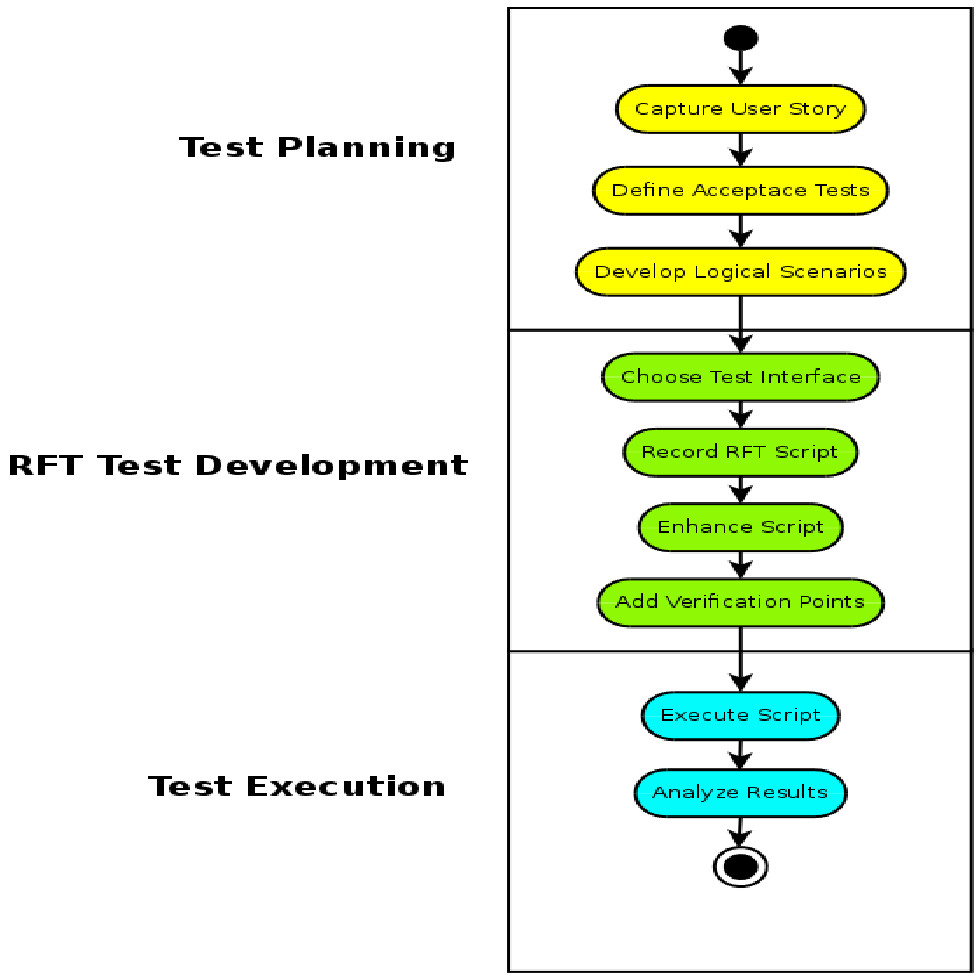
1.3 Integration with Testing Protocols

RDT automates part our testing process. This cannot be emphasized enough – if we have no testing process in place, then the benefits of using RFT are questionable. In this section, we will take a quick overview on when to use RFT and what should be done before we start working with RFT scripts.

"A fool with a tool is still a fool" - Grady Booch

We never develop RFT scripts on the fly as part of testing project (unless we are engaged in exploratory testing, but again, that is outside the scope of this course). Instead, script development uses the standard testing protocols that we use else where in our testing efforts. RFT does not replace any of our testing activities, it merely automates some of the tasks.

In the work flow diagram on the next page, we give a generic sense of the flow of what needs to be done before we develop an Rational Functional Tester script.



Suppose that we are developing an online CD ordering system for a music store (the Classics examples provided with the product) and the ordering application exists as a desktop Java application that can be accessed on their desktop. We will assume that there is some back end piece that we won't worry about that connects to our store ordering system.

If we were in a development process, we would normally start thinking about test cases in the requirements phase. I am using the term "requirements" here in a very generic term rather than as a reference to a specific formal phase in a development methodology. Requirements could refer to getting a formal business process description or to an Agile collaborative face-to-face discussion between the developer and the product owner.

Notice in the diagram I have divided the workflow into three different phases representing tasks which would be done at different times during the project.

We will not mention this again for the rest of the course but will focus instead on using RFT.

1.4 Automation

There are a number of excellent reasons for automating various kinds of tests. In fact, the benefits of automated testing are so significant that it is considered a professional software testing best practice to automate tests whenever possible. Some of the major reasons are:

1. *Automation enables continuous testing*. Instead of expending resources in setting up and manually running tests, they can be run whenever immediate feedback is needed and, once have the tests are automated, the resource cost of running the tests is negligible. This frees up a tester's time to do other tasks, like exploratory testing for example, since the developer just has to press a button to run the automated unit or acceptance tests.

2. *Automated tests remove a the opportunities for human error*. Automated testing is highly reliable in the sense that they are run exactly the same way every time. As well, if the reporting is also automated, then we remove the possibility of human error when examining the results of test execution. The problem with running large number of similar tests manually is that people get bored, stop paying attention or become inconsistent or sloppy in what they do – any number of situations that lead to human error.

3. *Automated tests reduce systemic testing errors*. Every software tester knows that we have to follow strict testing protocols in order to avoid systemic errors that can bias our testing efforts. However, automated testing removes the opportunity for these systemic errors to occur.

1.4.1 Pitfalls for Automation

Of course, like anything else, there are benefits and liabilities to using automation. In this section, some of the potential pitfalls testers can fall into when doing automated testing are mentioned below. These are not arguments against automated testing but rather are concerns that have to be addressed in order to do automated testing effectively and efficiently.

For example, automated UI tests provides a high level of confidence, but they are slow to execute, fragile to maintain and expensive to build if they are approached in a a poorly planed manner. Test automation should be treated as a "project" which is managed and subject to quality testing like any other project is. Automation may not significantly improve test productivity unless the testers know how to effectively and efficiently use and maintain the automated tests and tools.

1. Test Script Errors

Adding automation tools introduces a new source of testing error. With manual tests, we know that a test result depends on the quality of the test and whether or not an underlying fault existed. By maintaining strict quality for the test development and execution, we are able to achieve a high degree of confidence that test results are due to defects.

But what if there is a defect in the code that runs the test? Perhaps some of the tests are not being run or there is a logic error in the code that evaluates the result. Generally we want to perform the same sort of testing on the scripts and testing framework code that run our tests that we do on the actual code under development.

2. Unstable Tests

Tests are unstable when the feature description or acceptance criteria is unstable, often because the stakeholders have not fully settled on what that feature should do. What we want to try and avoid automating these tests too soon so that we avoid the situation were we constantly have to go back and rewrite the automated test, and then validate and verify the rewritten test.

3. Interface Dependencies

Very often automation tools operate through an interface, normally a web or other GUI interface. This means the automation script has to interact with the interface, so if the interface changes, then the script may have to be rewritten or else it might start returning incorrect results.

One way to mitigate against this is to create a special minimal testing interface through which all tests are run to avoid complications due to changes in the actual application interfaces. This is the approach generally recommended for acceptance testing but it does require going through the effort of setting up a testing interface.

4. Complacency

There can be a tendency to just forget about what we are actually testing in the automated tests, which means that our testing effort may lose validity in the sense that we think we are testing something when in fact we are not. For example, not all of the tests may be running, or the test suite no longer provides test coverage, or the requirements have changed but neither the test cases not the automation of the test cases have been updated. Since the tests drive development, this could be a critical problem.

We mitigate this by testing the tests. This might involve a couple or test quality activities.

1. Regular test automation reviews where the tests being run are examined to ensure that they are actually being run and that the correct ones are being run.
2. Develop test data for the test cases. If we have tests that are supposed to pick up a certain defect, the we develop a mocked up version of the product (we will look at mocks in module seven) which manifests a specific defect and only that defect. Running our automated tests against the mock should cause the test we are testing to fail and none of the others. In addition, we would have a mock that manifests no defects to ensure all the tests pass in the absence of any defects.

5. Set Up and Tear Down

The results of a test case depend on two factors, the test input and the state of the system when the test is run. It is possible that an automation may run the tests correctly but may either not put the system in the correct state (set up) for the test or return the test environment to the state it was in before the test was run (tear down).

We generally mitigate this by running configuration tests on our test automation script to ensure they do the set up and tear down correctly.

6. Test Metrics

A often used test quality metric is something like how many test cases a day are executed. While this might be an acceptable metric in a manual environment, it is meaningless in an automated environment. Instead we have to think about what sort of metrics are meaningful to us from a quality perspective.

7. "That's odd.."

Almost every tester has some story about how they notices some unexpected behaviour during a test that led to the discover of a previously unsuspected defect. In my own case, I noticed that when running a set of tests, that one of the tests ran many times slower than the rest, although they all should have taken about the same amount of time. All the tests passed, but the odd behaviour of that one test led to developing a set of exploratory tests that identified a subtle bug in the code that managed the database queries which actually caused a random momentary disconnection from the database.

In automated testing, these serendipitous observations may no longer occur because the test automation code never thinks to itself "That's odd..." We mitigate this by spending more time doing exploratory testing and various sorts of analyses of metrics of our testing process. For example, it might not be particularly difficult to track average execution times for a set of tests and then identify test runs where those execution time show a significant deviation from the expected value.

For example, if a test that has been running consistently in 1ms, then after a new build it starts taking 120ms to run, that may suggest that some change in the code may have had an unexpected impact in how other code executes.

2 The RFT Environment

As mentioned, RFT is build on two different existing platforms. For this class we will focus only on the Eclipse-Java environment since that is normally the default configuration. We will not explore the Visual Studio – Visual Basic platform in this course.

2.1 Eclipse

The RTF product is built on the Eclipse framework and works with the Eclipse architecture of perspectives and plugins.

In the 1990s, a company by the name of Rational Software, run by the same people who invented UML and RUP (Jim Rumbaugh, Grady Booch and Ivar Jacobson – collectively referred to as the three amigos) started buying up a variety of modeling products in an attempt automate the entire software development process from end to end. Unfortunately for them, they overextended themselves financially and were forced into bankruptcy. However fortunately for them, IBM then bought their company and re-branded their software tools as the IBM Rational Suite.

The IBM Rational suite consisted of a collection of various tools that all originated with different companies and organization and included:

1. *ClearCase:* An enterprise configuration management application.
2. *RequisitePro:* A requirements documentation and tracking tool.
3. *ClearQuest:* A change request management automation tool.
4. *Rational Rose:* A conceptual modeling tool for UML
5. *Test Manager:* An automation tool for test management
6. *Rational Robot:* An automated GUI testing tool that evolved into RFT
7. ...and lots more

Most of the these tools were adopted by the IRS and deployed across the whole organization over the several decades.

The insurmountable problem IBM faced was that each of these tools was made by a different companies as a stand-alone application, which meant that trying to integrate them all into a single suite was effectively impossible. The architecture, data formats, design principles, user interfaces and just about everything else about the products was just not “integrate-able”.

The eventual solution to this dilemma was to create a second generation of products, called Rational Team Concert that did three things:

1. Created a common back-end web application architecture called the Team Concert and Jazz Server so the various functional parts of the suite could actually work together, for example, requirements could be linked to test cases.

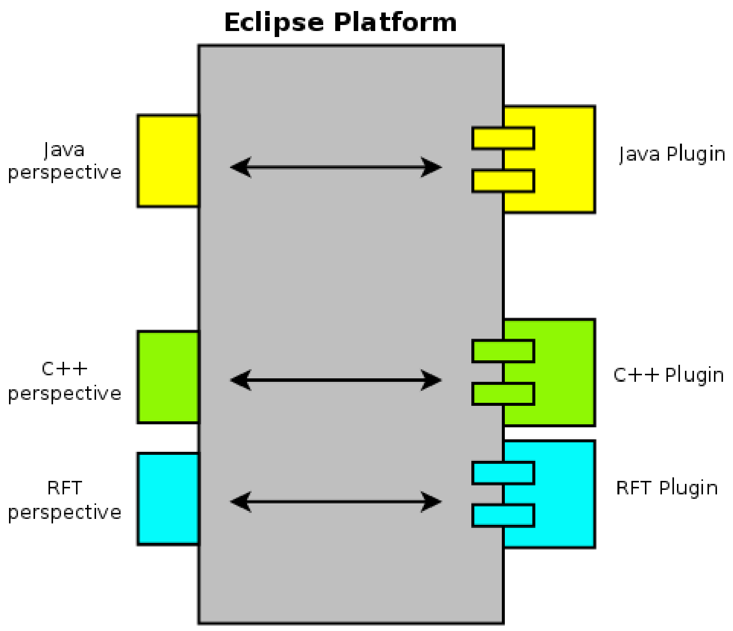
2. Rewrote the individual applications to have a common architecture and to have compatible artifacts. For example, RequisitePro evolved into Requirements Composer and Test Manager evolved into Quality Manager.

3. Implemented a common user interface using the Eclipse platform.

Since, as we mentioned earlier, there is no server component to RFT, the only one of these three innovations that concerns us is the Eclipse platform.

IBM had developed an integrated development environment called WebSphere and after keeping it proprietary for a while, open sources a version of the IDE under the name Eclipse and gave it to the Eclipse Foundation for distribution.

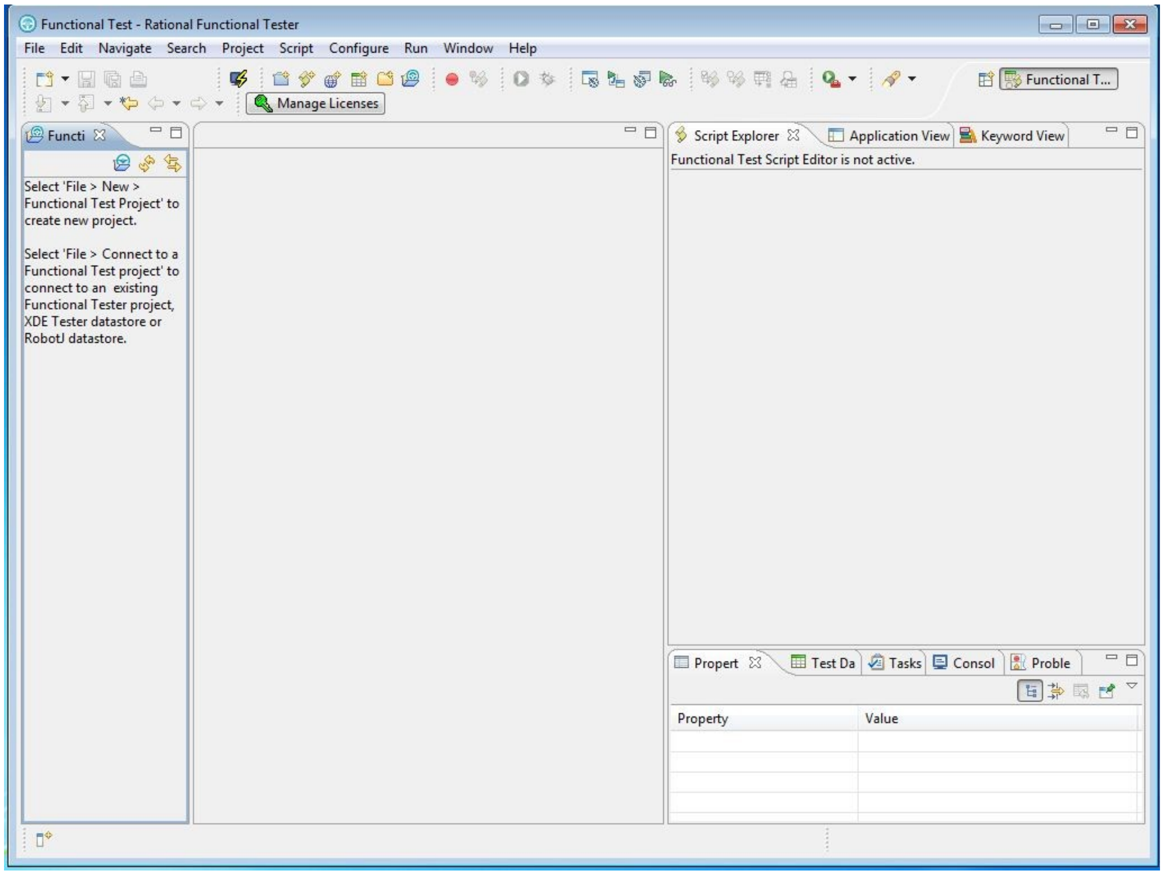
Eclipse is built around what is called a “plug-in” architecture. The basic idea is that instead of having one IDE for each development technology, for example, one for Java, another for C++ and another for Data Modeling, Eclipse provides a basic framework that would be common to all IDEs, sort of a generic IDE. We call this the “workbench”.



The specific functionality for each application is delivered via a plugin, for example, Java functionality is provided by a Java plug in. The plug-in provides the specific tools the IDE needs to do Java development. Similarly, a C++ plug-in provides the specific functionality that would be needed for C++ programming.

Each plug-in provides a unique interface called a perspective to access the capabilities of that plug in. Users can switch between plug-ins by choosing different perspectives.

For example, this is the RFT perspective.

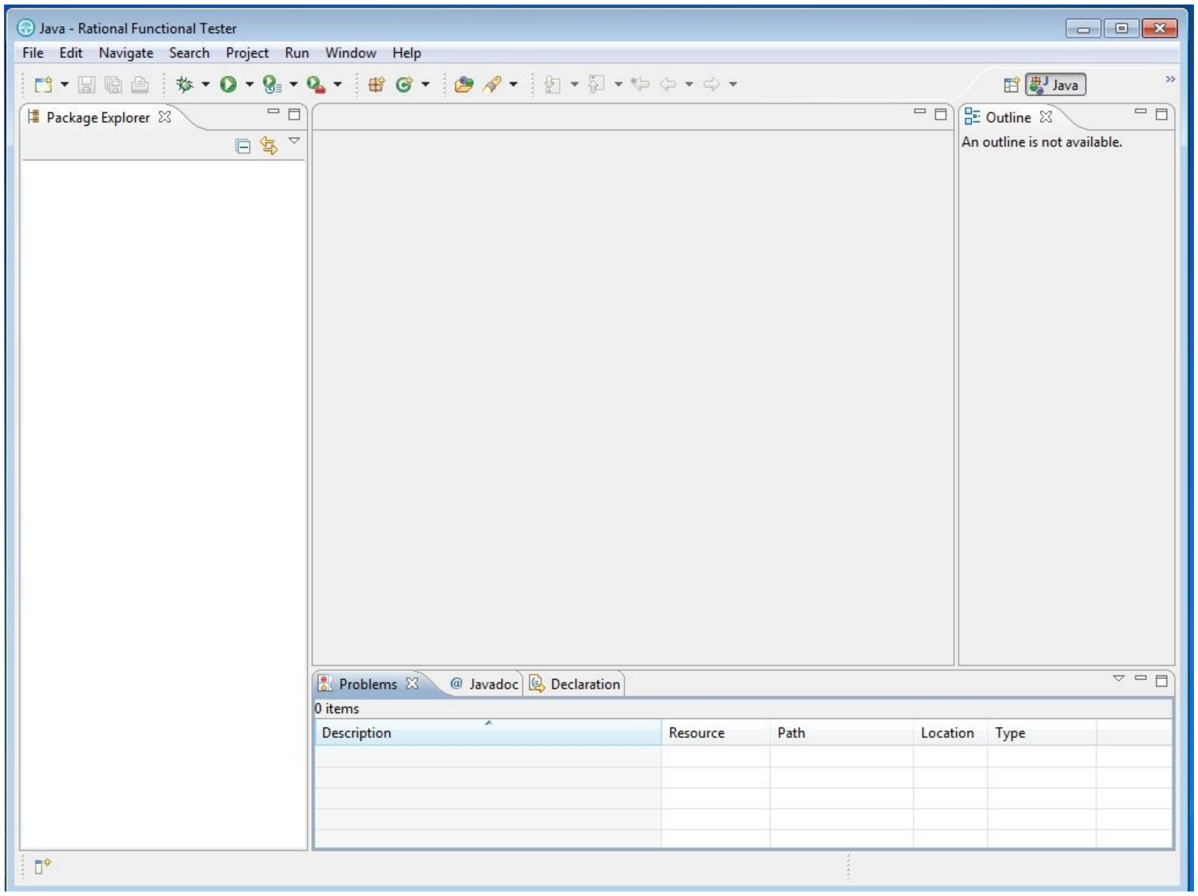


Select 'File > New > Functional Test Project' to create new project.

Select 'File > Connect to a Functional Test project' to connect to an existing Functional Tester project, XDE Tester datastore or RobotJ datastore.

Functional Test Script Editor is not active.

While on the next page, we see the Java Perspective.


Java - Rational Functional Tester
File Edit Navigate Search Project Run Window Help
Java
Package Explorer
Outline
An outline is not available.
Problems
Javadoc
Declaration
0 items
Description
Resource
Path
Location
Type

By selecting the perspective control at the top right of the interface, we can switch to any other perspective that is associated with an installed plug-in. For example, this is the Java Development perspective. (On the next page).

The use of this plug-in architecture is designed to provide a common look and feel for Rational applications as well as a standardization of the architectures of the various products.

The instructor will demonstrate how to navigate through the Eclipse interface, and provide explanations for the various generic Eclipse functionality. This is something you should follow along with and practice before beginning the first Lab.

2.2 Setting up a Project

The first thing that we have to do before we can start using RFT is to set up a project. We do this just by creating a new project from the menu and specifying optionally a place to store the project files. It was mentioned bit earlier that the project does not store the script but contains all the configuration and other data needed to create, edit and execute an RFT script

The instructor will demonstrate setting up an RFT project. There are also step by step instructions in Lab 2.

2.3 Configuration

Often, before we can create any test scripts, we have to do some preliminary configuration.

2.3.1 Enabling Test Environments

Because we can test a number of different applications, we need to enable the environments so that RFT can start the application. Note that we do not start the application ourselves, but we tell RFT how to start the application so that the application start is consistent across scripts.

Java Environments

Any Java application can run under a set of different Java run-time environments, and a given test environment may have multiple Java environments installed (for example, Java 8 and Java 10). Under the configuration menu, we can specify the Java environments we want to use.

Whenever a new Java environment is added, it should be tested. The JRE tester displays the JRE version, vendor, and a message that JRE successfully enabled if the environment is correctly configured.

Rational Functional Tester is shipped with a JRE (IBM Rational SDP JRE)that is automatically enabled during installation.

Web Browsers

Web Browsers that are to be used in testing HTML applications must be enabled before RFT can run an HTML application script. Depending on the operating system RFT can detect the browsers available. When running under Windows, the configuration tool consults the Windows registry to find installed browsers and automatically enables Internet Explorer if it is present. On *NIX systems, the configuration tool scans the hard drive(s) for any installed browsers.

Other browsers can be added manually since we may want to run scripts on different versions of browsers or non-standard browsers like Opera, Safari, IceCat and so on.

Similarly to the Java environments, each browser can be tested in the configuration panel to ensure it is configured correctly.

Eclipse Applications

The Eclipse enabler must be used to configure RFT to be able to test Eclipse based applications, or applications that use Eclipse as its user interface. The configuration tool scans hard drive(s) looking for any installed versions of Eclipse. Since we will not be using any of these applications and none are installed on the lab machines, this configuration feature is empty.

2.3.2 Configuring Applications

The first step in any RFT script is to start the application. Under the configuration menu, the "Configure Applications" allows us to configure how RFT should start the application.

For example, to start a Java Application, it has to be added to the applications menu and one of the enabled JVMs specified as the Java machine that will be used to start the application.

The instructor will demonstrate the configuration features of the RFT environment, however this topic is also covered in more detail in Lab 2.

2.3.3 Configuring Object Identification Properties

If you think about playing back a script, the question that should jump out at you is "how does the script know where to move the mouse or where to click?" When recording the script, it's obvious to you where the inputs should go because you are looking at the visual representation of the various widgets and components on the screen.

Very early versions of RFT actually saved the screen co-ordinates in terms of pixel as to where you had clicked on a mouse, or where the cursor was on the screen while you were typing.

The problem with this approach is that it was very brittle. If the application appeared at a different location on the screen, or if the screen resolution changed, then the script would break because the pixel co-ordinates no longer lined up with the buttons, text boxes and other widgets.

Instead, RFT uses an "Object Oriented" approach. All of the application types, such as Java applications and HTML applications are really made up of two layers:

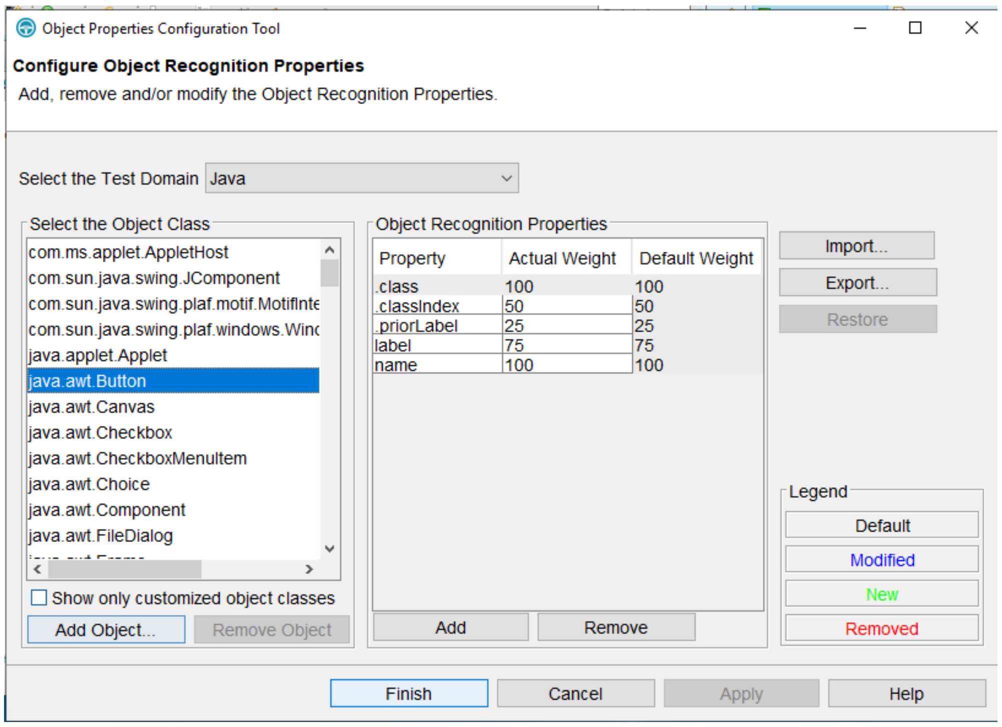
1. An underlying abstract code layer where program objects in code represent the various widgets and controls in a user interface. These can be Java classes or DOM objects in a browser.
2. A graphical rendering of the underlying program objects and a mapping from the underlying program object to that graphic depiction on the screen

At any given time, the underlying program object, a clickable button perhaps, has a number of properties that can be accessed via program code. For example, JavaScript is used to modify the display properties of the button to change its appearance (greying it out for example) or to specify what should happen when the user interacts with the graphical representation.

RFT maintains a catalogue of the various objects in the application *at the code level* in order to ensure that the script can access the appropriate widget no matter how it actually looks or is positioned on the user interface display.

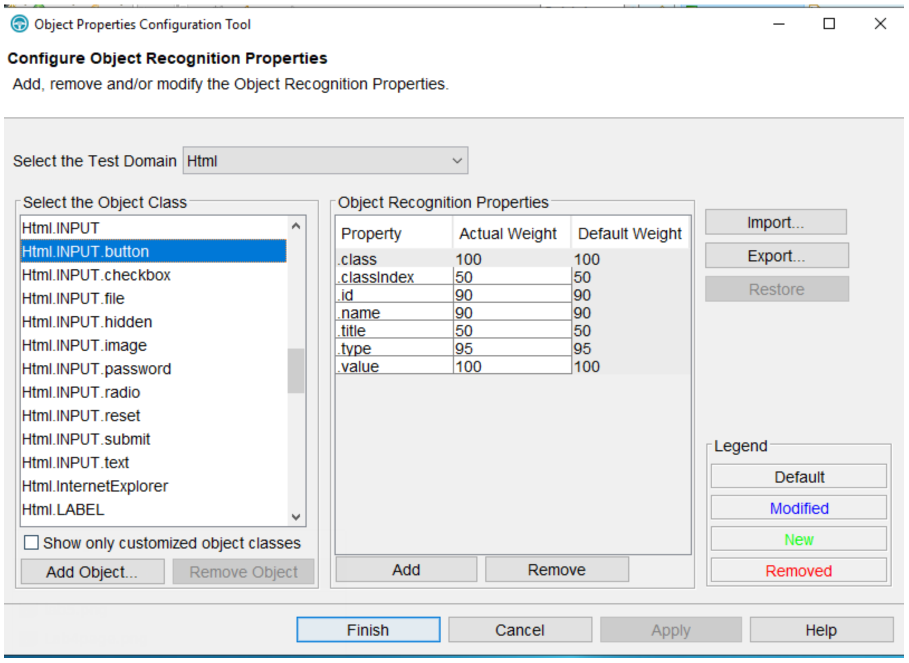
RFT enables users to configure which properties at the code level should be used to identify the various interface objects' recognition properties. In this class, we will stick with using the standard default options, but this can be very useful when introducing custom widgets or changing how existing widgets are located.

The screen shot on the next page shows the default properties for identifying a standard button in a Java application. In this case "java.awt.Button" is the name of the underlying Java class that is used to create buttons in the interface


Object Properties Configuration Tool
Configure Object Recognition Properties
Add, remove and/or modify the Object Recognition Properties.
Select the Test Domain
Java
Select the Object Class
com.ms.applet.AppletHost
com.sun.java.swing.JComponent
com.sun.java.swing.plaf.motif.MotifInte
com.sun.java.swing.plaf.windows.Winc
java.applet.Applet
java.awt.Button
java.awt.Canvas
java.awt.Checkbox
java.awt.CheckboxMenuItem
java.awt.Choice
java.awt.Component
java.awt.FileDialog
Show only customized object classes
Add Object...
Remove Object
Object Recognition Properties
Add
Remove
Import...
Export...
Restore
Legend
Default
Modified
New
Removed
Finish
Cancel
Apply
Help

Property	Actual Weight	Default Weight
.class	100	100
.classIndex	50	50
.priorLabel	25	25
label	75	75
name	100	100

Similarly, the following screenshot shows how RFT identifies a button on an HTML web page.



Property	Actual Weight	Default Weight
.class	100	100
.classIndex	50	50
.id	90	90
.name	90	90
.title	50	50
.type	95	95
.value	100	100

We will come back to object maps in a later sections.

To complete this module, you should work through Labs One and Two at the end of this manual.

Recording and Playing Simple Scripts

Module Two

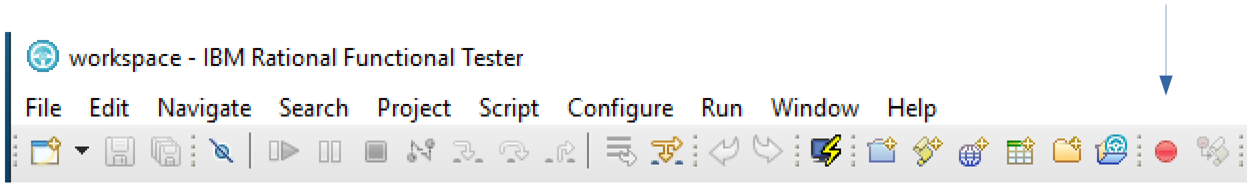
3 Recording a Script

The process of recording and playing back a script is documented in detail in Lab 3, however the instructor will demonstrate the process first. It is recommended that you follow along and try and duplicate what they are doing.

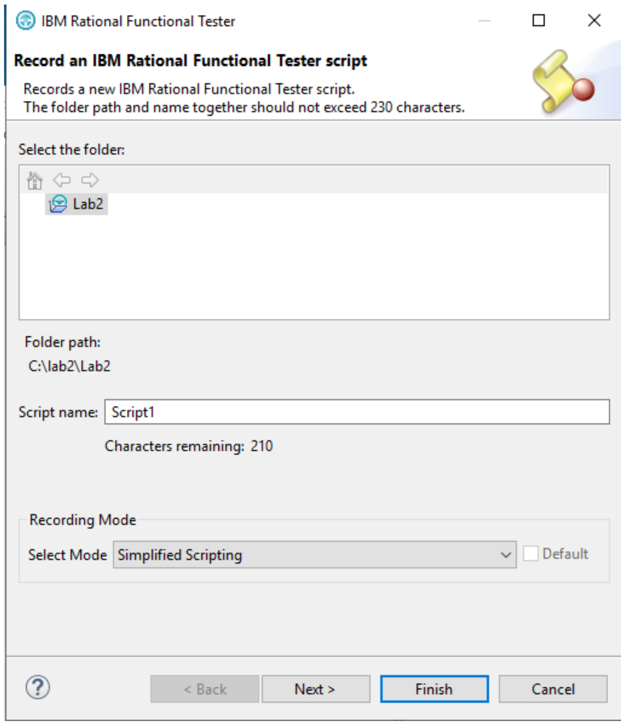
3.1 Starting the Recorder

Once the project has been set up, we can record a script. We can record any number of scripts in a given project.

We can start by selecting the record option (the red circle) form the task bar or the start recording option from the "Script" menu


workspace - IBM Rational Functional Tester
File Edit Navigate Search Project Script Configure Run Window Help

When started, we are prompted for a script name (we are being really creative here and using "Script1"). We also want to ensure that we are using "Simplified Scripting" in order to make the script easier to edit.


IBM Rational Functional Tester

Record an IBM Rational Functional Tester script

Records a new IBM Rational Functional Tester script.
The folder path and name together should not exceed 230 characters.

Select the folder:

Lab2

Folder path:
C:\lab2\Lab2

Script name: Script1

Characters remaining: 210

Recording Mode

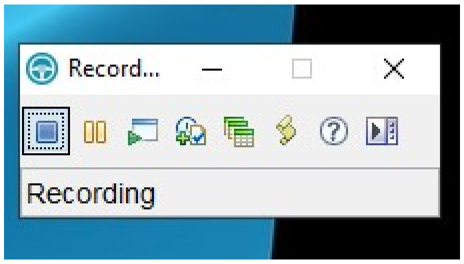
Select Mode: Simplified Scripting

Default

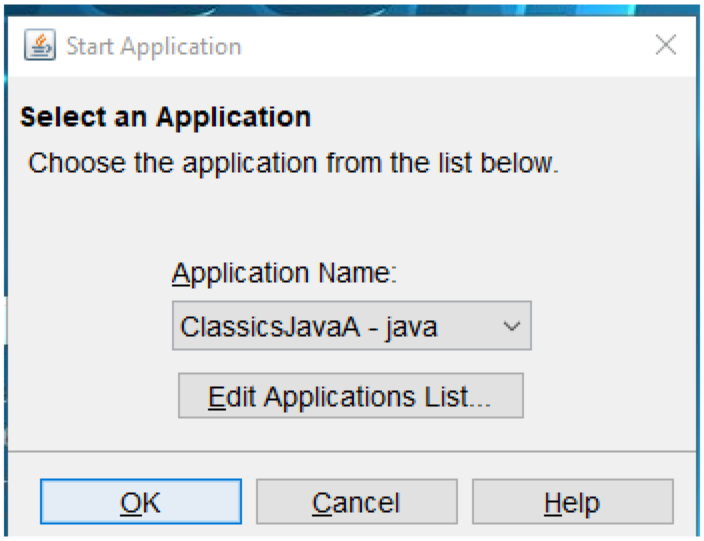
< Back | Next > | Finish | Cancel

Simplified scripting is a simplified presentation of the underlying Java code that is actually makes up the script.

Once "Finish" is selected, the Eclipse interface is minimized and the recorder dialogue box appears.



From this dialogue box, we select the "Start Application" button (third from the left). This brings up the "Select Application" dialogue box



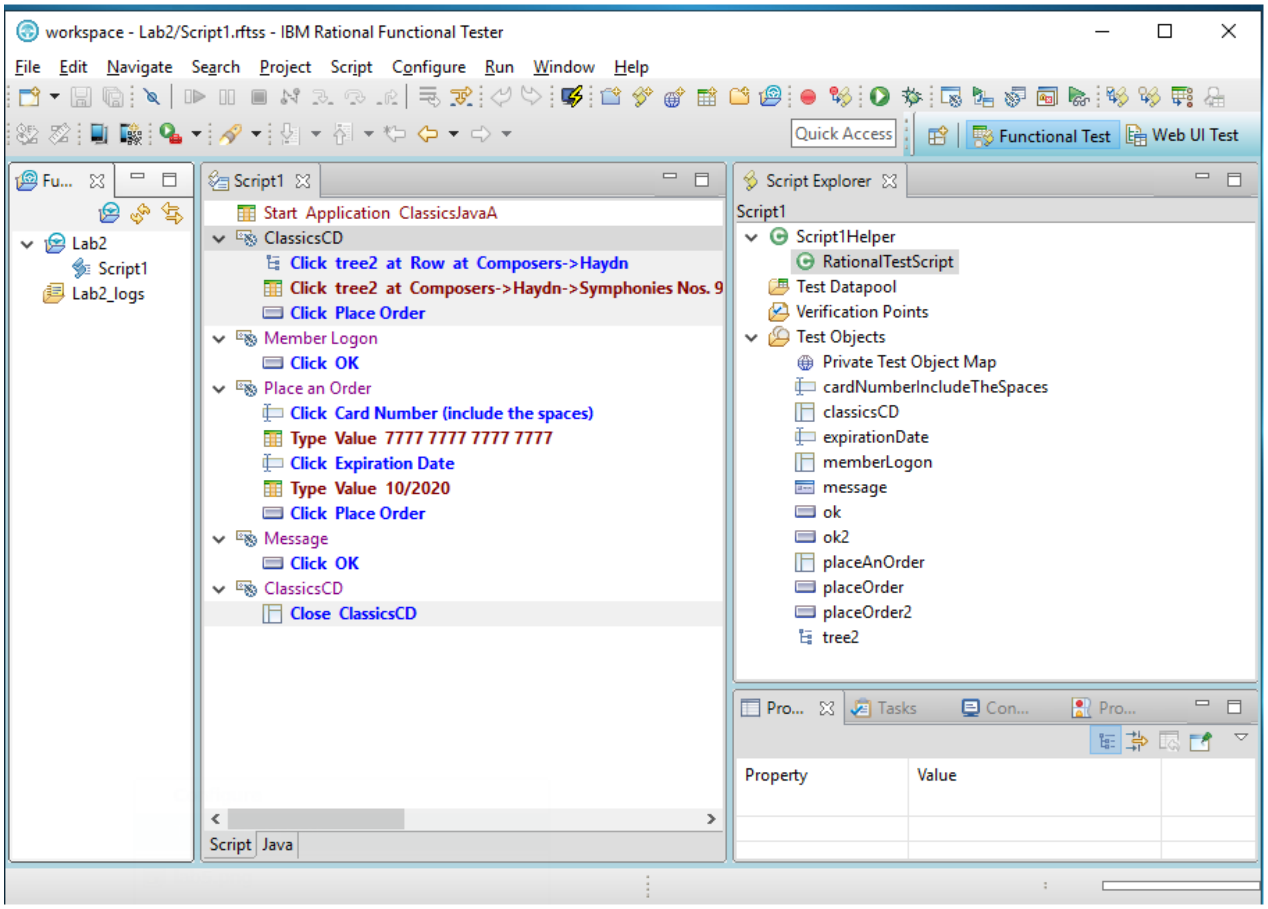
3.2 Recording the Script

To avoid replication, the script is recorded after selecting "OK" using the steps outlined in Lab 3

The instructor will demonstrate the steps involved in recording an RFT script.

3.3 Examining the Script

Once the script has been recorded, it is displayed in the project workspace.



```
Start Application ClassicsJavaA
ClassicsCD
    Click tree2 at Row at Composers->Haydn
    Click tree2 at Composers->Haydn->Symphonies Nos. 9
    Click Place Order
Member Logon
    Click OK
Place an Order
    Click Card Number (include the spaces)
    Type Value 7777 7777 7777 7777
    Click Expiration Date
    Type Value 10/2020
    Click Place Order
Message
    Click OK
ClassicsCD
    Close ClassicsCD
```

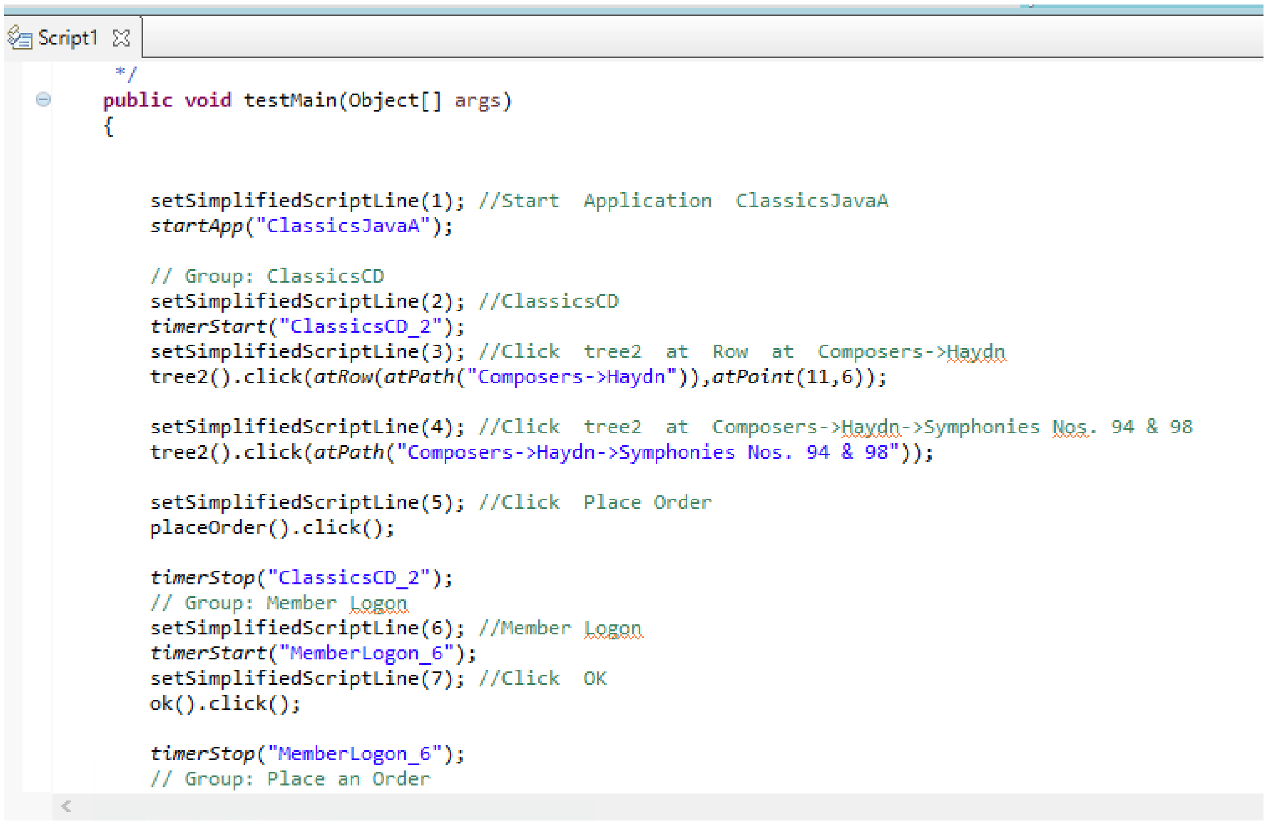
```
Script1
    Script1Helper
        RationalTestScript
    Test Datapool
    Verification Points
    Test Objects
        Private Test Object Map
        cardNumberIncludeTheSpaces
        classicsCD
        expirationDate
        memberLogon
        message
        ok
        ok2
        placeAnOrder
        placeOrder
        placeOrder2
        tree2
```

One thing to note is that we are looking at a simplified script. We can examine the actual executable Java code by selecting the Java tab at the bottom of the script window.

Notice that there are no logs yet because we generate logs only during script playback, not during the recording of the script.

If we want to examine the actual executable code, we can look at the actual Java code that is executed when we run the scrip. Just select the "Java" tab at the bottom of the Script1 window.

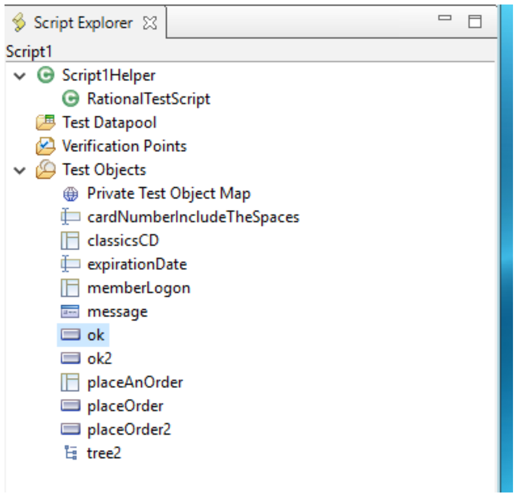
A sample of this script is shown below:



Script1

```
     */
    public void testMain(Object[] args)
    {


        setSimplifiedScriptLine(1); //Start  Application  ClassicsJavaA
        startApp("ClassicsJavaA");

        // Group: ClassicsCD
        setSimplifiedScriptLine(2); //ClassicsCD
        timerStart("ClassicsCD_2");
        setSimplifiedScriptLine(3); //Click  tree2  at  Row  at  Composers->Haydn
        tree2().click(atRow(atPath("Composers->Haydn")),atPoint(11,6));

        setSimplifiedScriptLine(4); //Click  tree2  at  Composers->Haydn->Symphonies Nos. 94 & 98
        tree2().click(atPath("Composers->Haydn->Symphonies Nos. 94 & 98"));

        setSimplifiedScriptLine(5); //Click  Place Order
        placeOrder().click();

        timerStop("ClassicsCD_2");
        // Group: Member Logon
        setSimplifiedScriptLine(6); //Member Logon
        timerStart("MemberLogon_6");
        setSimplifiedScriptLine(7); //Click  OK
        ok().click();

        timerStop("MemberLogon_6");
        // Group: Place an Order
```

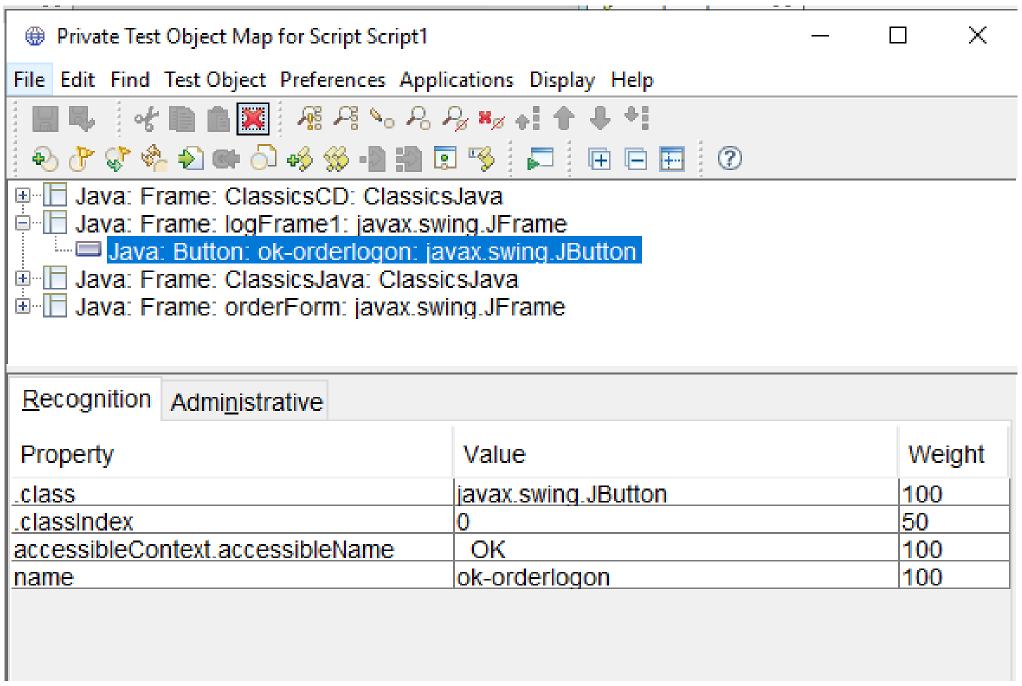
We can also examine the test assets that were created during the recording of the script. For example, if we look at the "OK" button



Script Explorer

Script1
- Script1Helper
  - RationalTestScript
- Test Datapool
- Verification Points
- Test Objects
  - Private Test Object Map
  - cardNumberIncludeTheSpaces
  - classicsCD
  - expirationDate
  - memberLogon
  - message
  - ok
  - ok2
  - placeAnOrder
  - placeOrder
  - placeOrder2
  - tree2

We can look at the profile for that underlying object that RFT generated. For example, this is identified as a javax.swing.JButton program object.



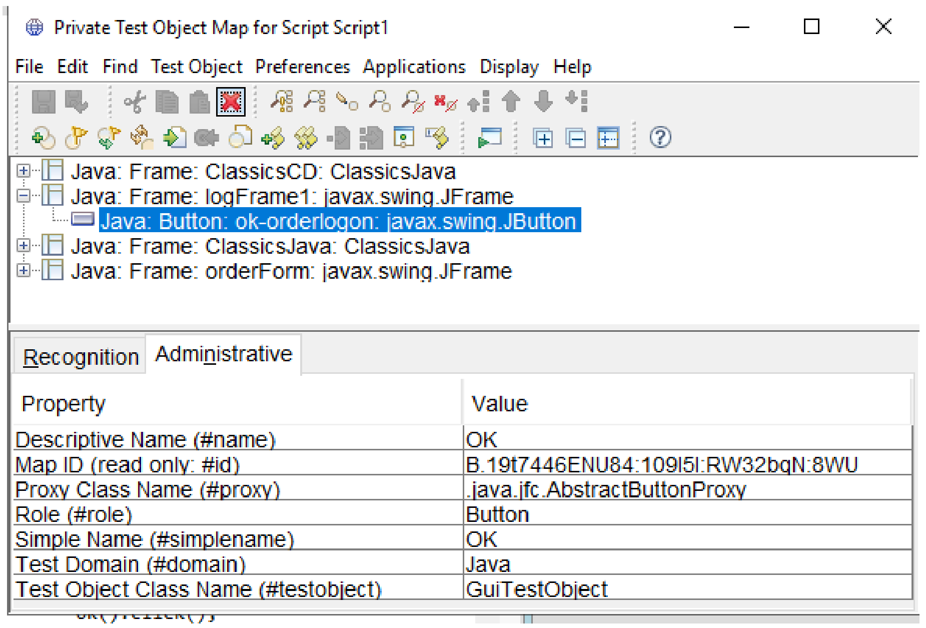
Private Test Object Map for Script Script1
File Edit Find Test Object Preferences Applications Display Help

Java: Frame: ClassicsCD: ClassicsJava
Java: Frame: logFrame1: javax.swing.JFrame
Java: Button: ok-orderlogon: javax.swing.JButton
Java: Frame: ClassicsJava: ClassicsJava
Java: Frame: orderForm: javax.swing.JFrame

Recognition | Administrative

Property	Value	Weight
.class	javax.swing.JButton	100
.classIndex	0	50
accessibleContext.accessibleName	OK	100
name	ok-orderlogon	100

And some of the properties that are associated with that object.



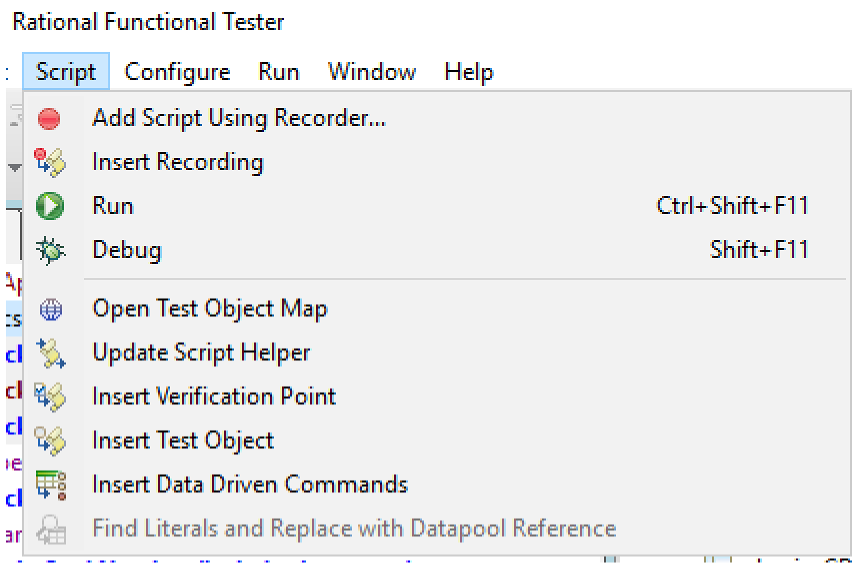
Private Test Object Map for Script Script1
File Edit Find Test Object Preferences Applications Display Help

Java: Frame: ClassicsCD: ClassicsJava
Java: Frame: logFrame1: javax.swing.JFrame
Java: Button: ok-orderlogon: javax.swing.JButton
Java: Frame: ClassicsJava: ClassicsJava
Java: Frame: orderForm: javax.swing.JFrame

Recognition | Administrative

Property	Value
Descriptive Name (#name)	OK
Map ID (read only: #id)	B.19t7446ENU84:109I5I:RW32bqN:8WU
Proxy Class Name (#proxy)	.java.jfc.AbstractButtonProxy
Role (#role)	Button
Simple Name (#simplename)	OK
Test Domain (#domain)	Java
Test Object Class Name (#testobject)	GuiTestObject

3.4 Running the Script


Rational Functional Tester
Script
Configure
Run
Window
Help
Add Script Using Recorder...
Insert Recording
Run Ctrl+Shift+F11
Debug Shift+F11
Open Test Object Map
Update Script Helper
Insert Verification Point
Insert Test Object
Insert Data Driven Commands
Find Literals and Replace with Datapool Reference

Now the script can be run with either the run option from the script menu or the "Run" button from the task bar.

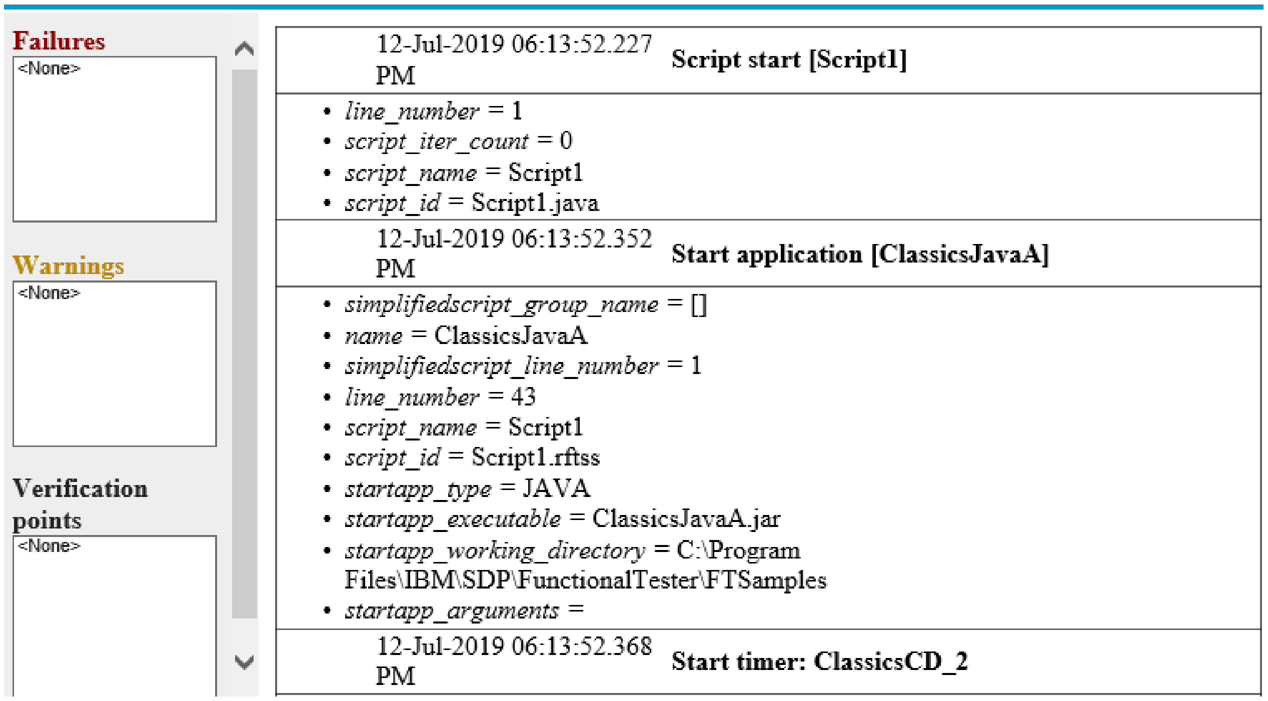
When running the script, RFT will ask for the name of a log file to store the results in. Once the script executes, the results of that test script will be displayed in a browser (since we chose HTML logging).

The instructor will demonstrate this. You should follow along and examine the various log files and other artifacts produced.

This is also described step by step in Lab 3.

3.4.1 Log Files

Since we chose HTML logging, the log file is shown in a browser as:



```
12-Jul-2019 06:13:52.227 PM    Script start [Script1]
• line_number = 1
• script_iter_count = 0
• script_name = Script1
• script_id = Script1.java
12-Jul-2019 06:13:52.352 PM    Start application [ClassicsJavaA]
• simplifiedscript_group_name = []
• name = ClassicsJavaA
• simplifiedscript_line_number = 1
• line_number = 43
• script_name = Script1
• script_id = Script1.rftss
• startapp_type = JAVA
• startapp_executable = ClassicsJavaA.jar
• startapp_working_directory = C:\Program
  Files\IBM\SDP\FunctionalTester\FTSamples
• startapp_arguments =
12-Jul-2019 06:13:52.368 PM    Start timer: ClassicsCD_2
```

Most of this information is low level detail that we are not too interested in, or at least until we have to start debugging. What we are interested in whether a test fails or passes its verification points, which is the topic of the next module.

3.5 Editing the Script

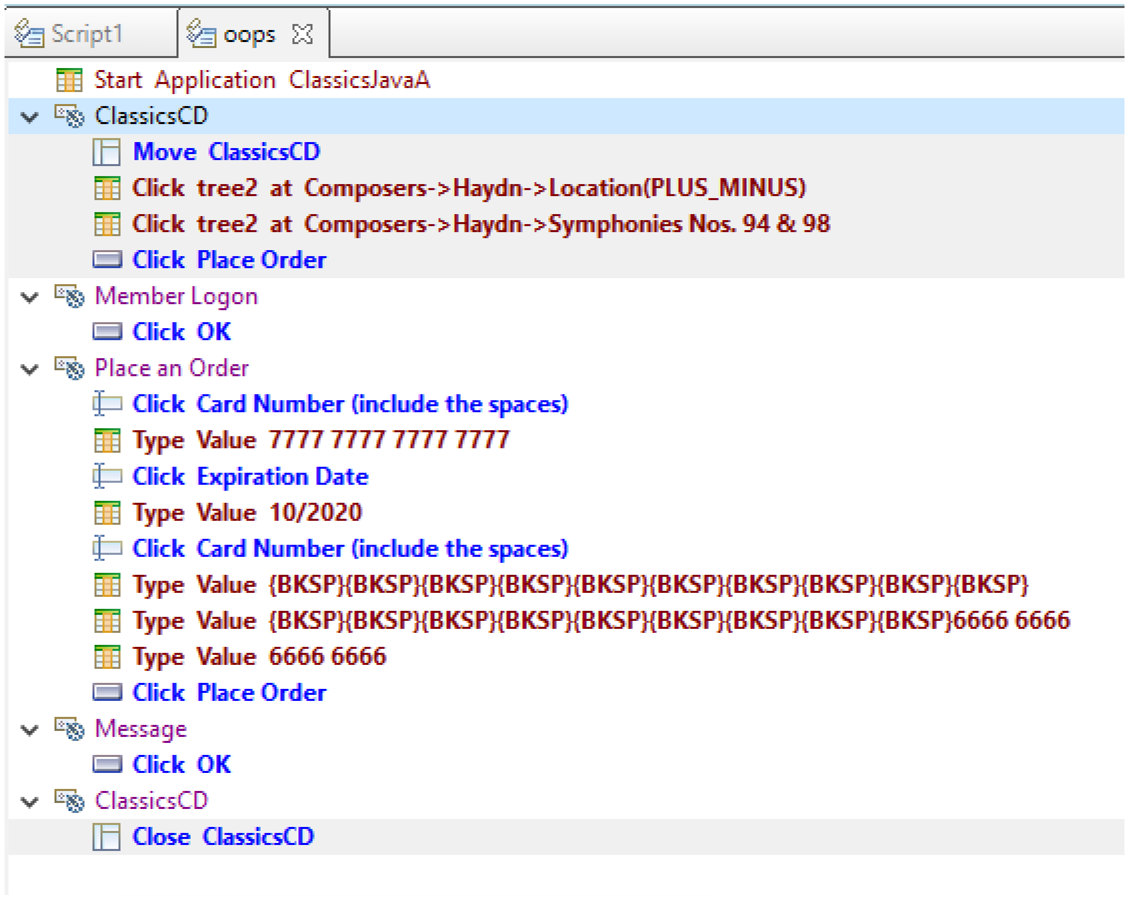
In the lab, you will edit the credit card number and the expiry date. Once the script has been generated, we do have the ability to modify the data values used in the script.

We can also modify other aspects of the user choices by working with what is selected; however, this is not recommended since it is very easy to make errors in editing references to selections other than entered data.

In the simplified script, we can disable actions, which has the effect of commenting out the corresponding Java code. We an also enable any action we have disabled. We can also do this directly in the Java code; however if you are not an experienced Java programmer, this is not recommended.

It is also possible to delete some actions. This is useful when we make mistakes in entering the data and want to edit out those mistakes from the script.

For example, if I had made a mistake in the entry of the credit card number then went back and corrected it like so:

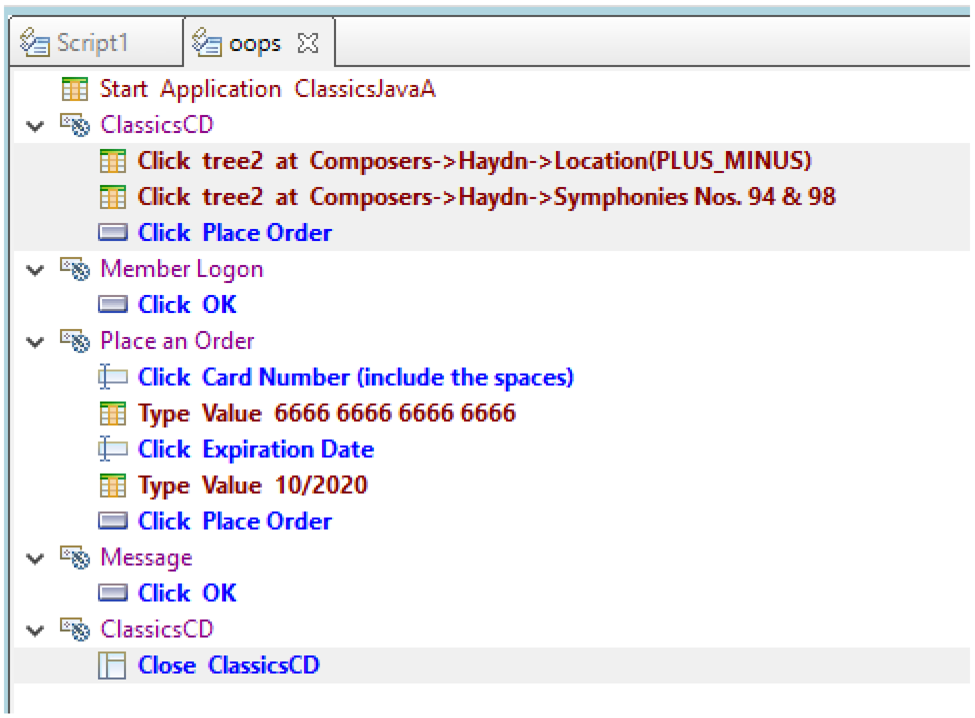


```
Start Application ClassicsJavaA
ClassicsCD
    Move ClassicsCD
    Click tree2 at Composers->Haydn->Location(PLUS_MINUS)
    Click tree2 at Composers->Haydn->Symphonies Nos. 94 & 98
    Click Place Order
Member Logon
    Click OK
Place an Order
    Click Card Number (include the spaces)
    Type Value 7777 7777 7777 7777
    Click Expiration Date
    Type Value 10/2020
    Click Card Number (include the spaces)
    Type Value {BKSP}{BKSP}{BKSP}{BKSP}{BKSP}{BKSP}{BKSP}{BKSP}{BKSP}{BKSP}
    Type Value {BKSP}{BKSP}{BKSP}{BKSP}{BKSP}{BKSP}{BKSP}{BKSP}{BKSP}6666 6666
    Type Value 6666 6666
    Click Place Order
Message
    Click OK
ClassicsCD
    Close ClassicsCD
```

I could edit out the lines where I made the correction and fix up the script. I noticed that I also moved the application at the start of the script from the top left corner. That was a mistake as well, so I can edit that action out.

Then the edited script now looks like the screen shot on the next page and works the way I originally intended.

You should always be careful about editing scripts and you should always be sure that you test your script after it has been edited so that you can confirm that you have not "broken" the script.

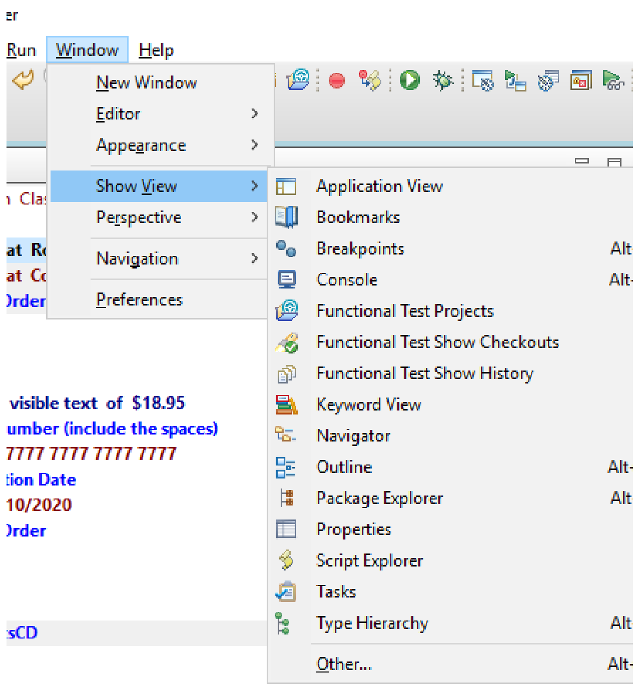


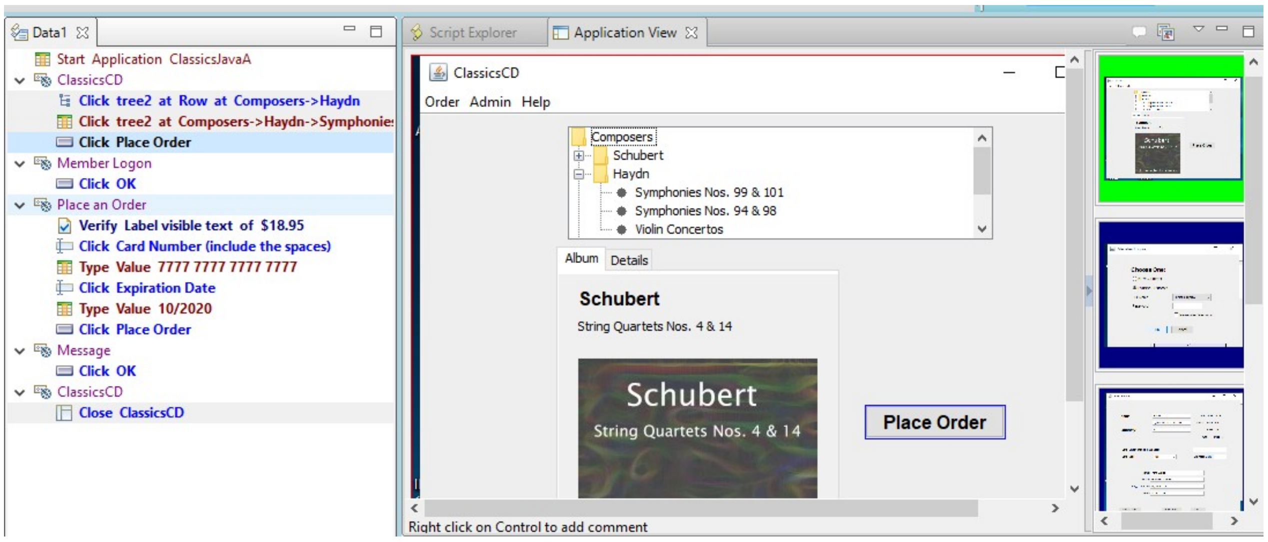
Script1 | oops

- Start Application ClassicsJavaA
- ClassicsCD
  - Click tree2 at Composers->Haydn->Location(PLUS_MINUS)
  - Click tree2 at Composers->Haydn->Symphonies Nos. 94 & 98
  - Click Place Order
- Member Logon
  - Click OK
- Place an Order
  - Click Card Number (include the spaces)
  - Type Value 6666 6666 6666 6666
  - Click Expiration Date
  - Type Value 10/2020
  - Click Place Order
- Message
  - Click OK
- ClassicsCD
  - Close ClassicsCD

3.6 Application View

While the application is running, RFT takes a screen shot of what is on the screen at each step of the script so that as you highlight each line of the script, the application view shows you what was visible at that moment. Application view can be displayed by selecting the following from the "Window -> Show View" menu.





```
Data1
  Start Application ClassicsJavaA
v ClassicsCD
    Click tree2 at Row at Composers->Haydn
    Click tree2 at Composers->Haydn->Symphonie:
    Click Place Order
v Member Logon
    Click OK
v Place an Order
    Verify Label visible text of $18.95
    Click Card Number (include the spaces)
    Type Value 7777 7777 7777 7777
    Click Expiration Date
    Type Value 10/2020
    Click Place Order
v Message
    Click OK
v ClassicsCD
    Close ClassicsCD
```

Right click on Control to add comment

The active windows and controls at that point in the script execution are highlighted. Stepping through the scrip using application view gives a story board of what was seen at each step of the execution.

Verification Points

Module Three

4 Verification Points

It is a fundamental axiom of software testing that every test case must have a predefined expected outcome that we use to compare against the actual computed value to determine if the test case has passed or failed.

Up until now, we have been constructing test scripts but they all pass because we have not had any expected values to compare any actual values to.

However, the other problem we have is that for a script, there is not just a single computation to be concerned with, but rather a series of points in the script where things can go wrong; not just wrong values but wrong text sizes, colours, images and more.

What we do to adapt our test case logic to handle a script by establishing a series of what are called *verification points.* These are specific places in the script where we check either the data contained in a widget (like the credit card number) or the properties of a widget (for example whether or not a check box has been selected).

In other words, we identify a number of places where our script could do something we don't expect, and for each of those places, we establish a verification point.

For example, we might also have image verification points that we use to determine if the right images are being seen at the right times.

At each verification point we define what we expect to see and when the script gets to that point, RFT does the comparison of what we told it should be there and what actually is there. If all the verification points match, then the script passes. If any of the verification points does not match, the script fails.

4.1 Establishing Verification Points

An important point is that RFT does not do testing for you but automates your tests – which means that you have to decide whether or not a piece of data should be verified and what the "correct" value should be defined to verify it against.

During recording, a verification point captures object information (our expected values) and stores it in a baseline data file. The information in this file becomes the baseline of the expected state of the object during subsequent playbacks of the script.

The verification point for an object is created by RFT at a specific point in the script. The object is a partial copy of the actual object that is selected by the user at the place in the scrip recording where a verification is desired.

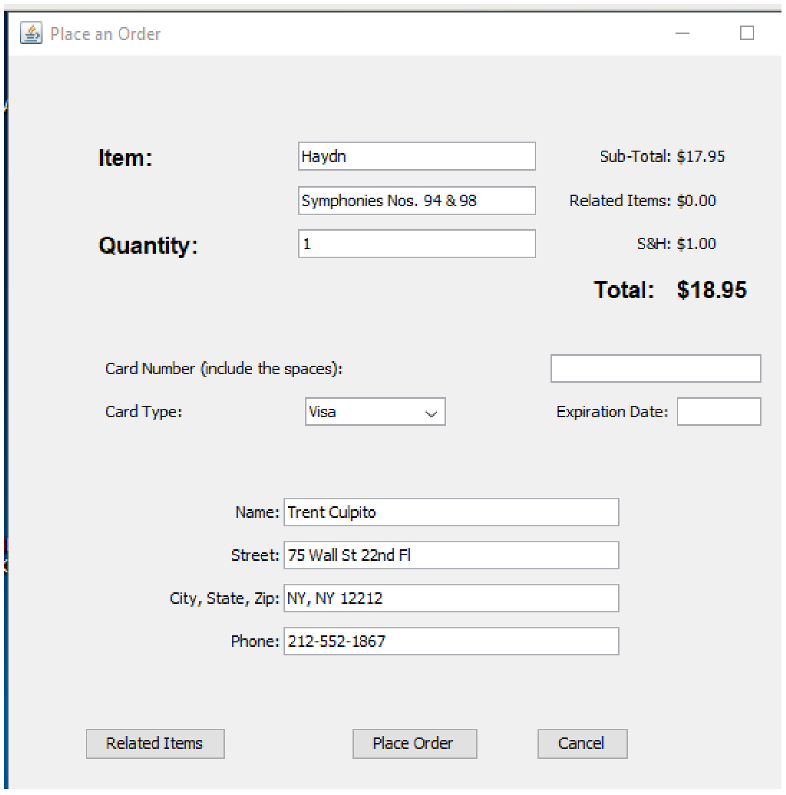
The extent of the copy made depends on the kind of verification point selected.

4.2 Creating Verification Points

Verification points are normally created during the recording process using the verification point creation tool. This is the easiest way to insert verification points, but it does require that we have done our testing planning so that we know exactly what data we want to verify in the script and what the values are that we expect to find.

Just to clarify, by "data we expect to find," we generally also mean that we see the options that should be available to the user, images, text or anything else that would be seen or could be manipulated by the user.

For example, if we wanted to create a simple data verification point to test the calculation of the total in ordering the CDs in the example we have been looking at, we would start recording the scrip as usual (here it is called data1) and when we get to the screen where the data we want to validate is displayed, we see something that looks like the screenshot below.


Place an Order

Item: Haydn

Symphonies Nos. 94 & 98

Quantity: 1

Sub-Total: $17.95

Related Items: $0.00

S&H: $1.00

Total: $18.95

Card Number (include the spaces):

Card Type: Visa

Expiration Date:

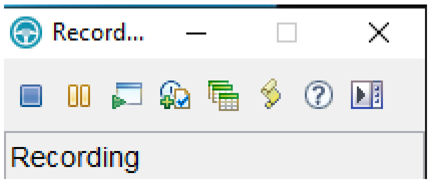
Name: Trent Culpito

Street: 75 Wall St 22nd Fl

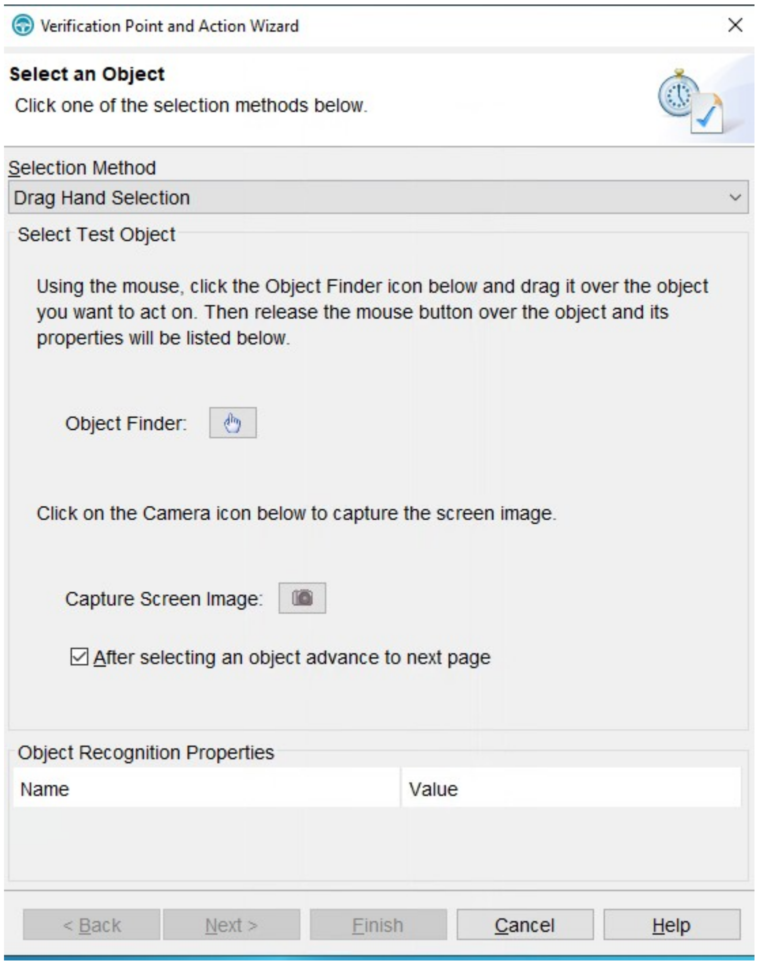
City, State, Zip: NY, NY 12212

Phone: 212-552-1867

The verification point button (4th from the left with the little green checkmark) on the record dialogue box is selected.



This brings up the Verification point wizard. This allows the user to select the object to be verified from the current application display.


Verification Point and Action Wizard

Select an Object

Click one of the selection methods below.

Selection Method: Drag Hand Selection

Select Test Object

Using the mouse, click the Object Finder icon below and drag it over the object you want to act on. Then release the mouse button over the object and its properties will be listed below.

Object Finder:

Click on the Camera icon below to capture the screen image.

Capture Screen Image:

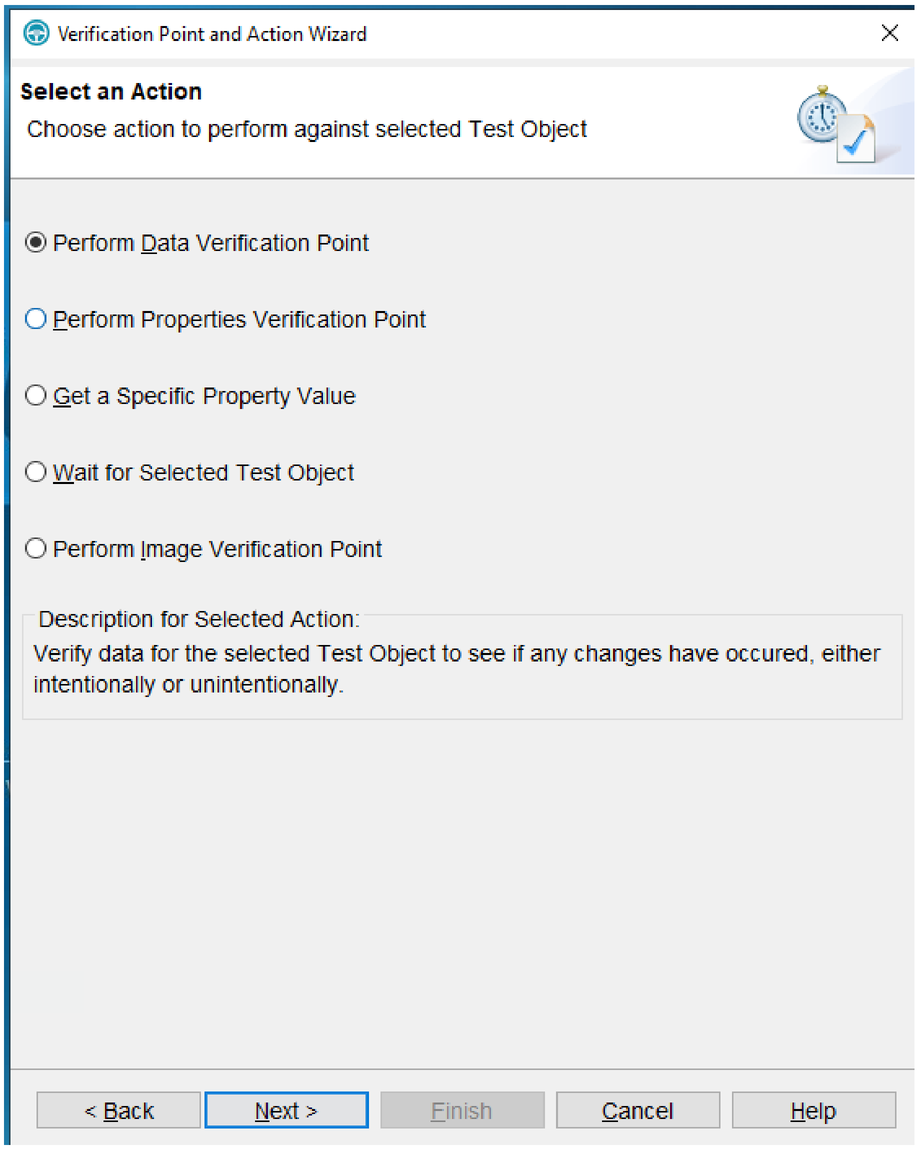
☑ After selecting an object advance to next page

Object Recognition Properties

Name	Value

Following the instructions and dragging the hand icon over to the $18.95 display brings up the following dialogue.

This dialogue shows us the types of verification points we can select for insertion. The subsequent dialogue boxes will depend on the type we choose here since the data captured for the verification point will depend on the type of verification point we have selected and the object we have selected.



Verification Point and Action Wizard

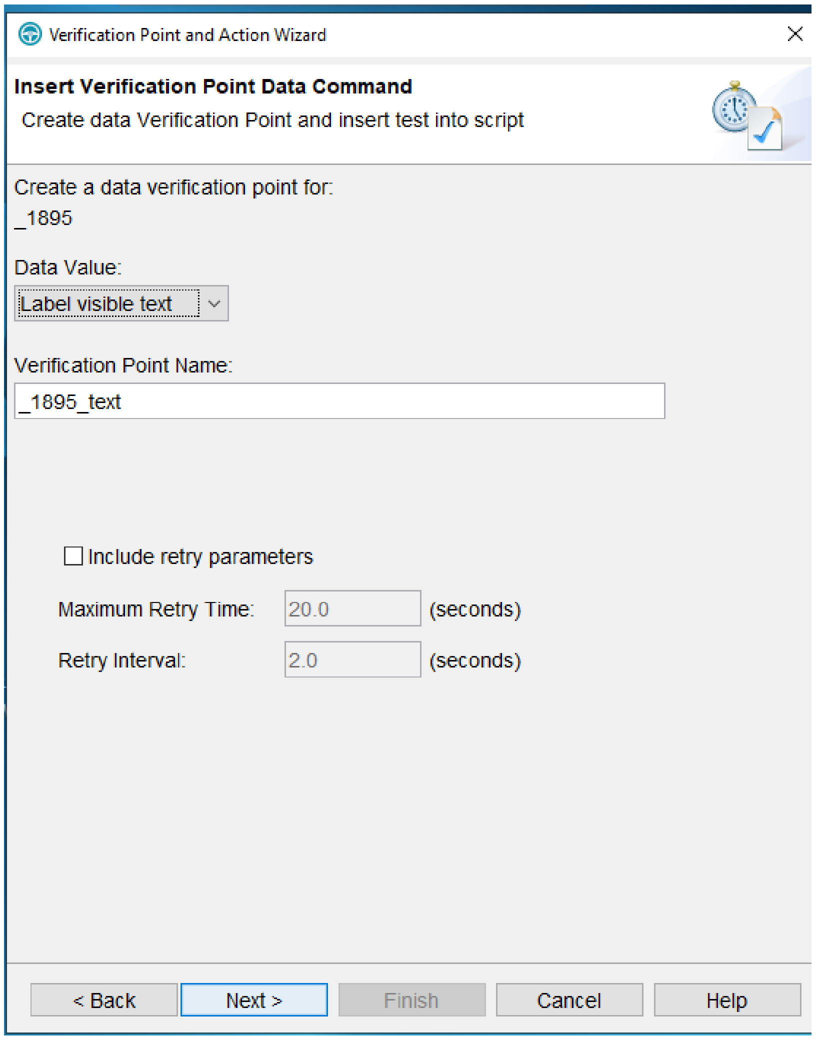
Select an Action

Choose action to perform against selected Test Object

- [✓ Perform Data Verification Point]
- Perform Properties Verification Point
- Get a Specific Property Value
- Wait for Selected Test Object
- Perform Image Verification Point

Description for Selected Action:
Verify data for the selected Test Object to see if any changes have occured, either intentionally or unintentionally.

In this case we will continue with a Data Verification Point and select "Next".



Verification Point and Action Wizard

Insert Verification Point Data Command

Create data Verification Point and insert test into script

Create a data verification point for:
_1895

Data Value:
Label visible text

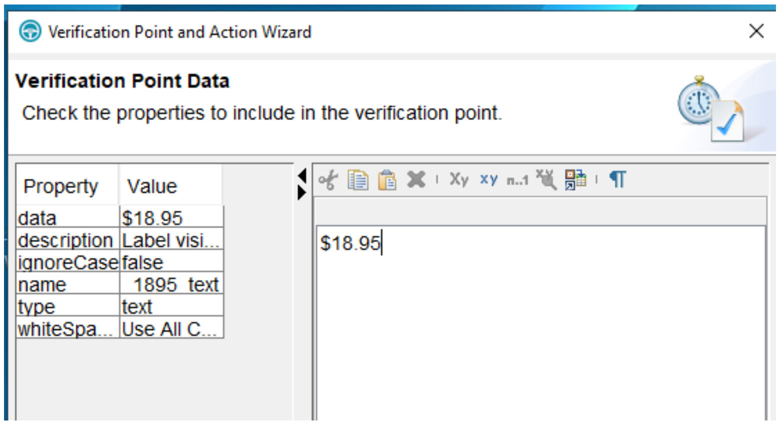
Verification Point Name:
_1895_text

☐ Include retry parameters

Maximum Retry Time:	20.0	(seconds)
Retry Interval:	2.0	(seconds)

We are validating the value in a label and data verification point is being created. It is a good idea to actually rename the point here, but for this first example, it will be left with the auto generated name. Selecting "Next" allows us to verify the data that will be used as the baseline value in the data verification point.

Although it is not in the cropped screen shot, selecting "Finish" allow us to then complete recording the script.


Verification Point and Action Wizard

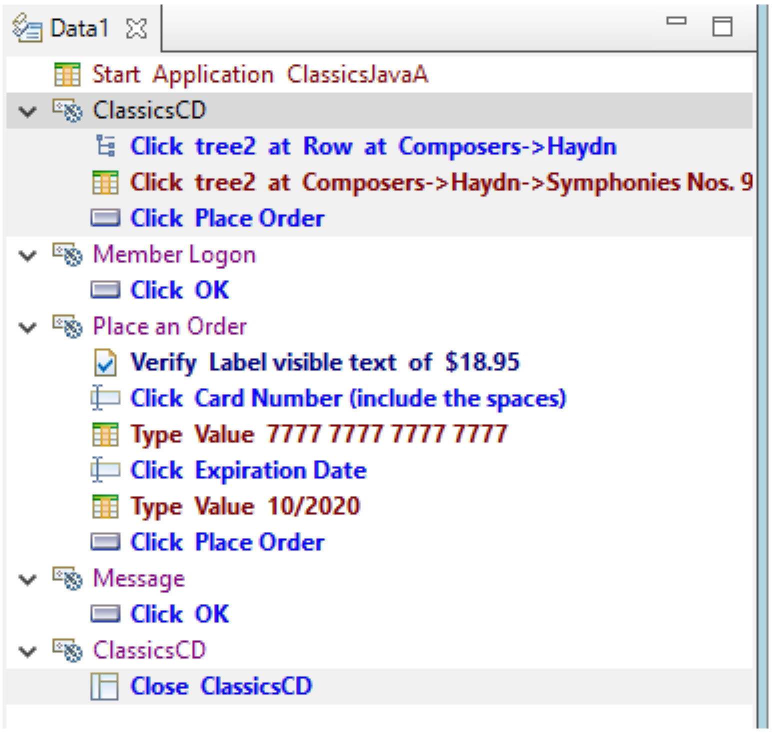
Verification Point Data

Check the properties to include in the verification point.

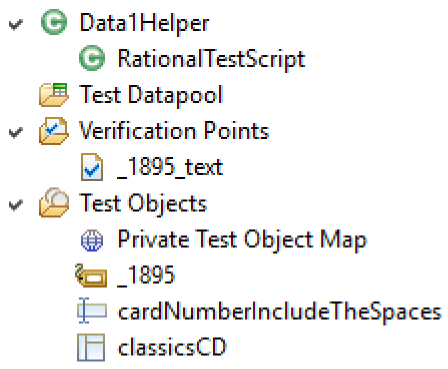
Property	Value
data	$18.95
description	Label visi...
ignoreCase	false
name	1895_text
type	text
whiteSpa...	Use All C...

$18.95

Now when we examine the script, we see that there is a verification point now inserted into the script.

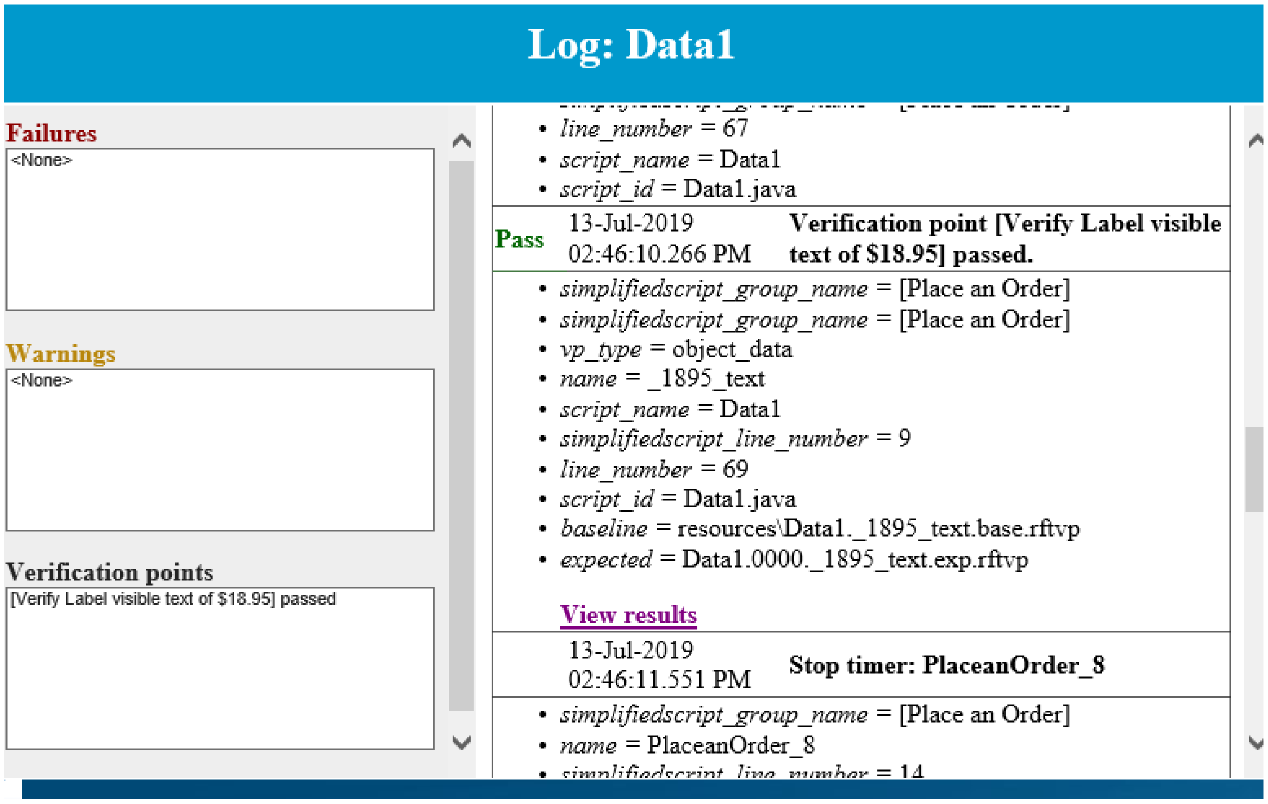

```
Data1
  Start Application ClassicsJavaA
  ClassicsCD
    Click tree2 at Row at Composers->Haydn
    Click tree2 at Composers->Haydn->Symphonies Nos. 9
    Click Place Order
  Member Logon
    Click OK
  Place an Order
    Verify Label visible text of $18.95
    Click Card Number (include the spaces)
    Type Value 7777 7777 7777 7777
    Click Expiration Date
    Type Value 10/2020
    Click Place Order
  Message
    Click OK
  ClassicsCD
    Close ClassicsCD
```

Looking at the Script Explorer, we can also see the verification point has been added to the script test assets.



Data1Helper
RationalTestScript
Test Datapool
Verification Points
_1895_text
Test Objects
Private Test Object Map
_1895
cardNumberIncludeTheSpaces
classicsCD

Replaying the script now produces the following log file output.



Log: Data1

Failures
<None>

Warnings
<None>

Verification points
[Verify Label visible text of $18.95] passed

- *line_number* = 67
- *script_name* = Data1
- *script_id* = Data1.java

Pass	13-Jul-2019 02:46:10.266 PM	**Verification point [Verify Label visible text of $18.95] passed.**

- *simplifiedscript_group_name* = [Place an Order]
- *simplifiedscript_group_name* = [Place an Order]
- *vp_type* = object_data
- *name* = _1895_text
- *script_name* = Data1
- *simplifiedscript_line_number* = 9
- *line_number* = 69
- *script_id* = Data1.java
- *baseline* = resources\Data1._1895_text.base.rftvp
- *expected* = Data1.0000._1895_text.exp.rftvp

View results

	13-Jul-2019 02:46:11.551 PM	**Stop timer: PlaceanOrder_8**

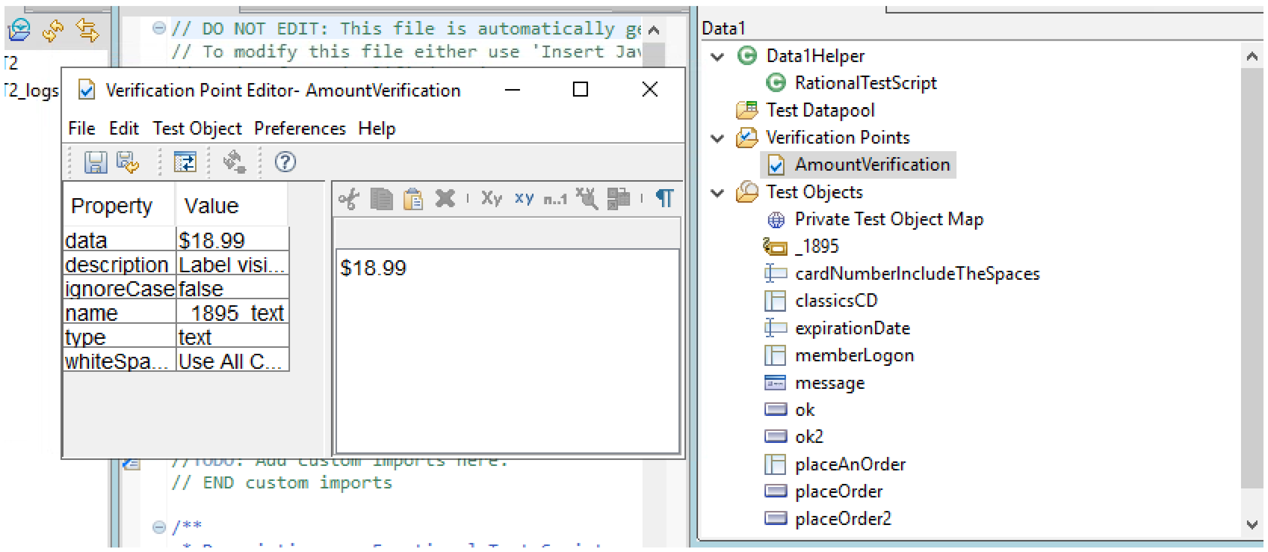
- *simplifiedscript_group_name* = [Place an Order]
- *name* = PlaceanOrder_8

In the summary on the left, the verification point was recorded as passing, and the log entry for the line where the verification point was located is also marked as having passed.

4.3 Editing Verification Points

To illustrate what happens when a verification point fails, the current example has been edited. First, the name of the verification point has been changed to "AmountVerification" which is a better name, but note this had to be done in the Java tab, not the simple script tab.

Choosing edit for this verification point brings up the same editor window seen when we created the point. In this case, we are changing the baseline value to be $18.99.

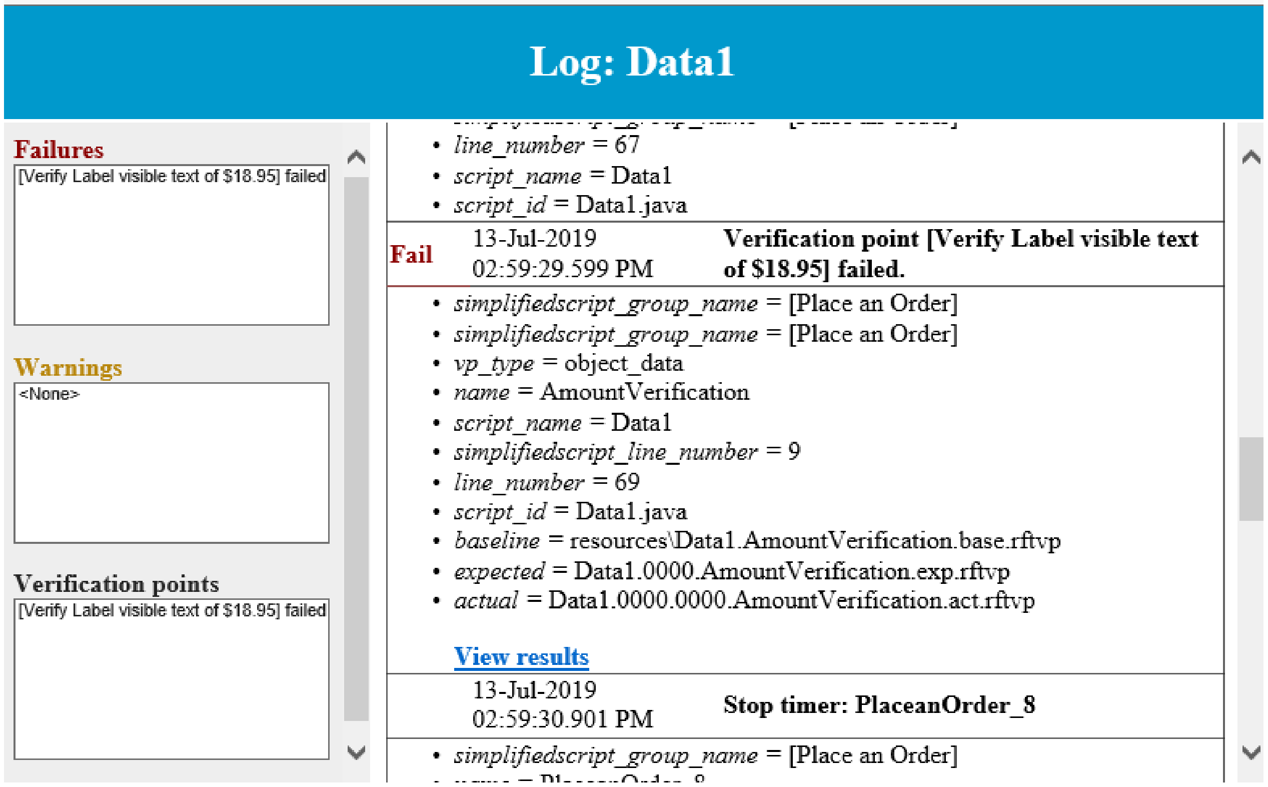


```
// DO NOT EDIT: This file is automatically ge
// To modify this file either use 'Insert Jav
// END custom imports
```

Property	Value
data	$18.99
description	Label visi...
ignoreCase	false
name	1895_text
type	text
whiteSpa...	Use All C...

$18.99

Saving the new value and rerunning the script produces the following log file output.



Log: Data1

Failures
[Verify Label visible text of $18.95] failed

Warnings
<None>

Verification points
[Verify Label visible text of $18.95] failed

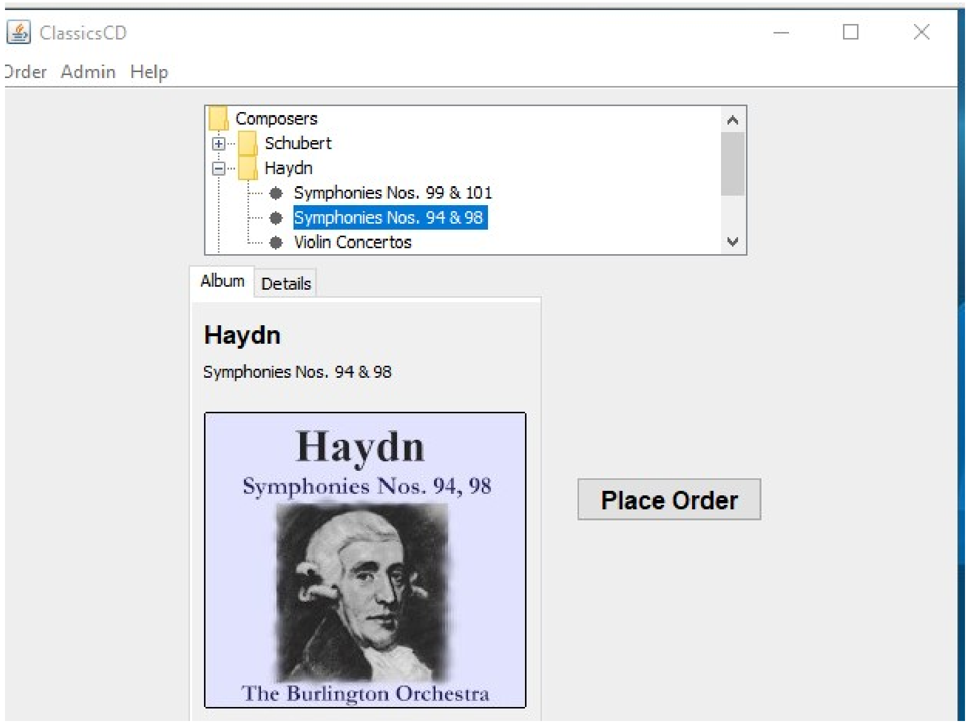
```
• line_number = 67
• script_name = Data1
• script_id = Data1.java
Fail  13-Jul-2019            Verification point [Verify Label visible text
      02:59:29.599 PM        of $18.95] failed.
• simplifiedscript_group_name = [Place an Order]
• simplifiedscript_group_name = [Place an Order]
• vp_type = object_data
• name = AmountVerification
• script_name = Data1
• simplifiedscript_line_number = 9
• line_number = 69
• script_id = Data1.java
• baseline = resources\Data1.AmountVerification.base.rftvp
• expected = Data1.0000.AmountVerification.exp.rftvp
• actual = Data1.0000.0000.AmountVerification.act.rftvp

View results
      13-Jul-2019            Stop timer: PlaceanOrder_8
      02:59:30.901 PM
• simplifiedscript_group_name = [Place an Order]
```

The verification point value of $18.95 is now a failure and is logged as such.

4.4 An Object Verification Point

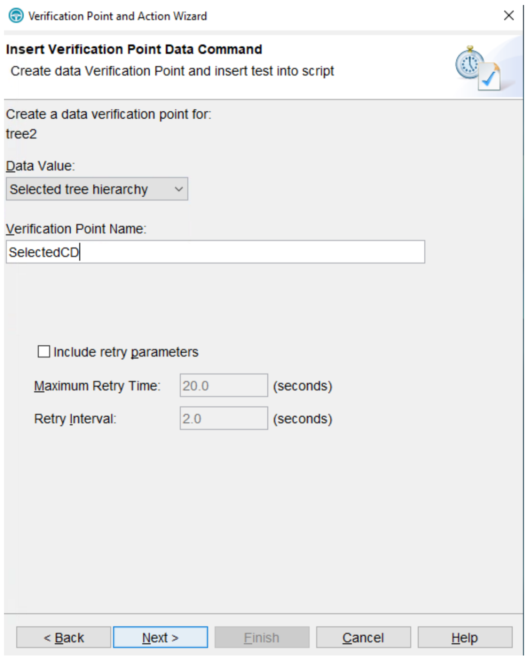
Object verification points are used to establish baselines for which objects are visible or selected or are in the correct state. The following is an example of a selection tree verification point.

The script starts in the usual manner, however, this time a verification point is created when the get to point in the script shown below.



And we want to verify that the correct CD is being chosen. We bring up the verification point dialogue and select a data verification. Remember that the wizard will then produce the correct options for the selected object.

In this case, the dialogue on the next page is shown.


Verification Point and Action Wizard

Insert Verification Point Data Command

Create data Verification Point and insert test into script

Create a data verification point for:
tree2

Data Value:
Selected tree hierarchy

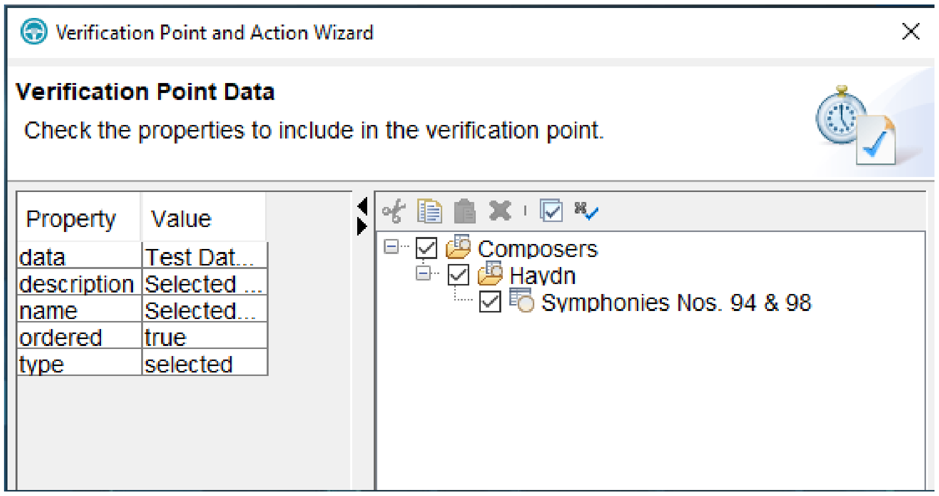
Verification Point Name:
SelectedCD

☐ Include retry parameters

Maximum Retry Time: 20.0 (seconds)

Retry Interval: 2.0 (seconds)

This time we are renaming the verification point with a meaningful name. Selecting "Next" shows me the base line data selection.


Verification Point and Action Wizard

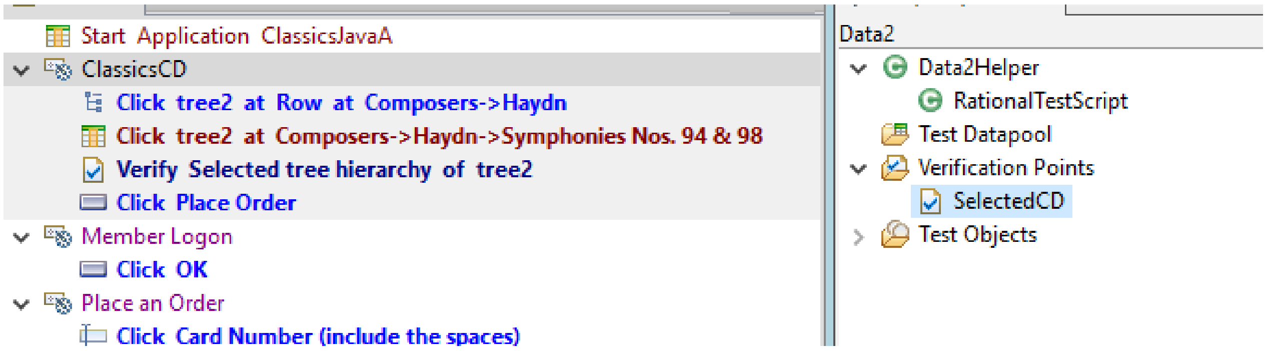
Verification Point Data

Check the properties to include in the verification point.

Property	Value
data	Test Dat...
description	Selected ...
name	Selected...
ordered	true
type	selected

☑ Composers
  ☑ Haydn
    ☑ Symphonies Nos. 94 & 98

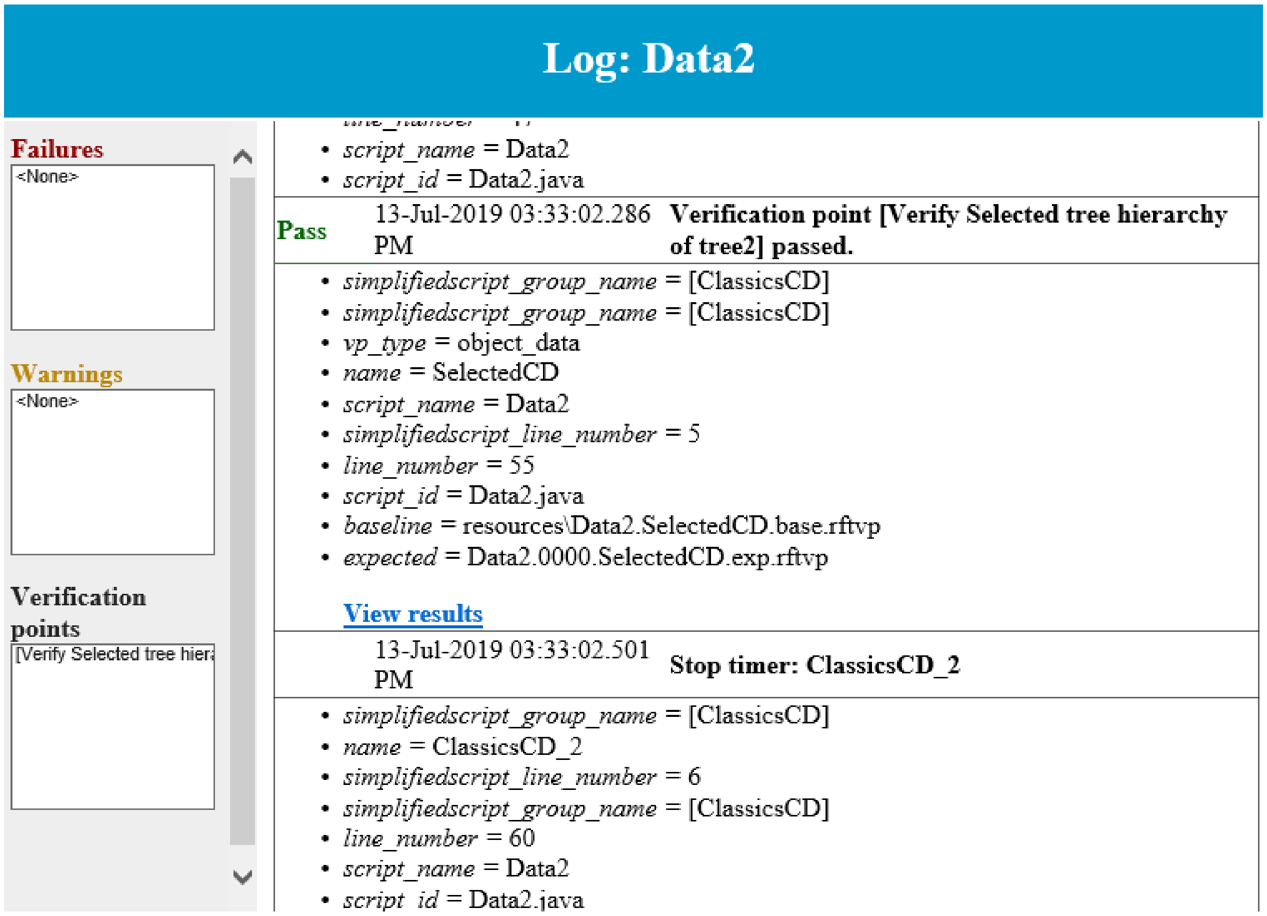
After the script is recorded, we can see the verification point recorded at the correct location.



```
Start Application ClassicsJavaA
ClassicsCD
    Click tree2 at Row at Composers->Haydn
    Click tree2 at Composers->Haydn->Symphonies Nos. 94 & 98
    Verify Selected tree hierarchy of tree2
    Click Place Order
Member Logon
    Click OK
Place an Order
    Click Card Number (include the spaces)
```

```
Data2
  Data2Helper
    RationalTestScript
  Test Datapool
  Verification Points
    SelectedCD
  Test Objects
```

And executing the script produces the following log output.



Log: Data2

Failures
<None>

Warnings
<None>

Verification points
[Verify Selected tree hiera

```
• script_name = Data2
• script_id = Data2.java
Pass   13-Jul-2019 03:33:02.286 PM   Verification point [Verify Selected tree hierarchy of tree2] passed.
• simplifiedscript_group_name = [ClassicsCD]
• simplifiedscript_group_name = [ClassicsCD]
• vp_type = object_data
• name = SelectedCD
• script_name = Data2
• simplifiedscript_line_number = 5
• line_number = 55
• script_id = Data2.java
• baseline = resources\Data2.SelectedCD.base.rftvp
• expected = Data2.0000.SelectedCD.exp.rftvp

View results
       13-Jul-2019 03:33:02.501 PM   Stop timer: ClassicsCD_2
• simplifiedscript_group_name = [ClassicsCD]
• name = ClassicsCD_2
• simplifiedscript_line_number = 6
• simplifiedscript_group_name = [ClassicsCD]
• line_number = 60
• script_name = Data2
• script_id = Data2.java
```

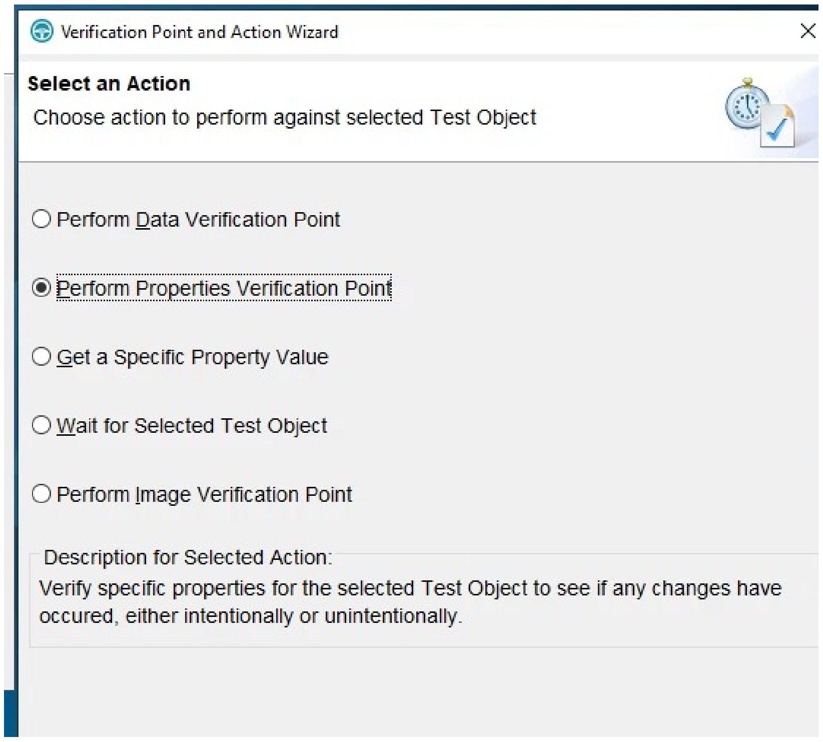
4.5 Property Verification

All graphical objects, whether Java widgets or HTML/DOM objects have properties associated with them. For example, objects that display text have properties that describe the colour of the text, the font of the test and the size of font, among others.

In regression testing, we want to often ensure that changes to our application or interface have not broken the properties of a given widget or object.

In this example, we are going to set a baseline property that the text being displayed in one of the labels should be in red. Since all of the text is actually in black, when we record the verification point, the baseline will be black but we will edit it later to make it red.

At the same place (the total amount) that we did the data verification, we insert a property verification point.



Select an Action

Choose action to perform against selected Test Object

○ Perform Data Verification Point

● Perform Properties Verification Point

○ Get a Specific Property Value

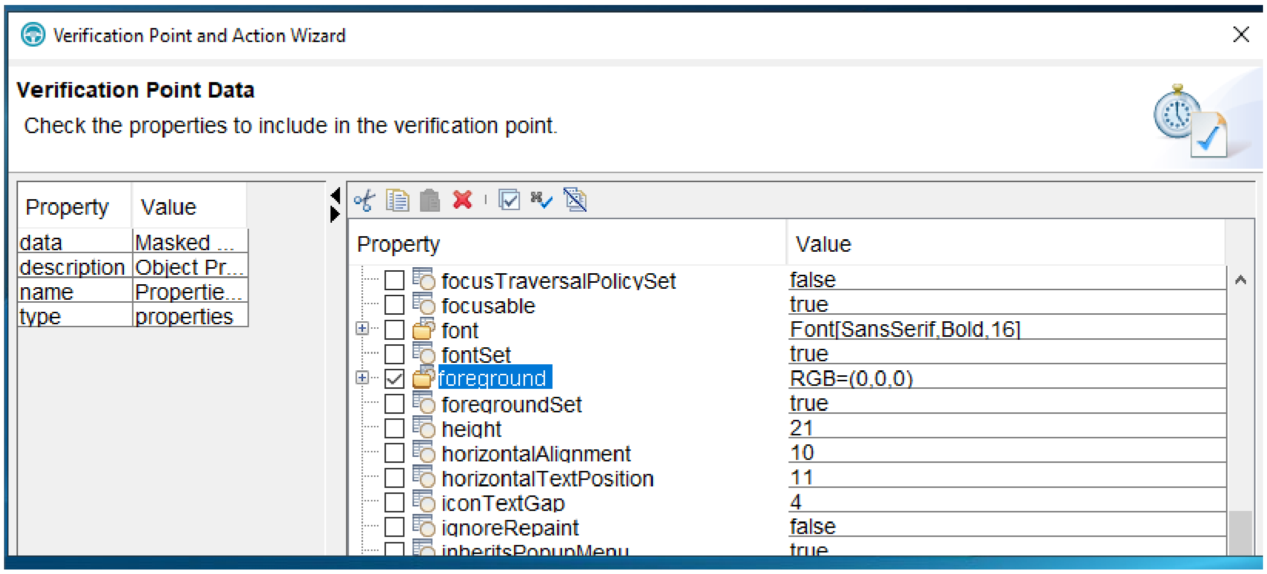
○ Wait for Selected Test Object

○ Perform Image Verification Point

Description for Selected Action:

Verify specific properties for the selected Test Object to see if any changes have occured, either intentionally or unintentionally.

Selecting "Next" gives us a list of all the properties for the amount text label. We select the "foreground" property since that is the colour of the text.


Verification Point and Action Wizard

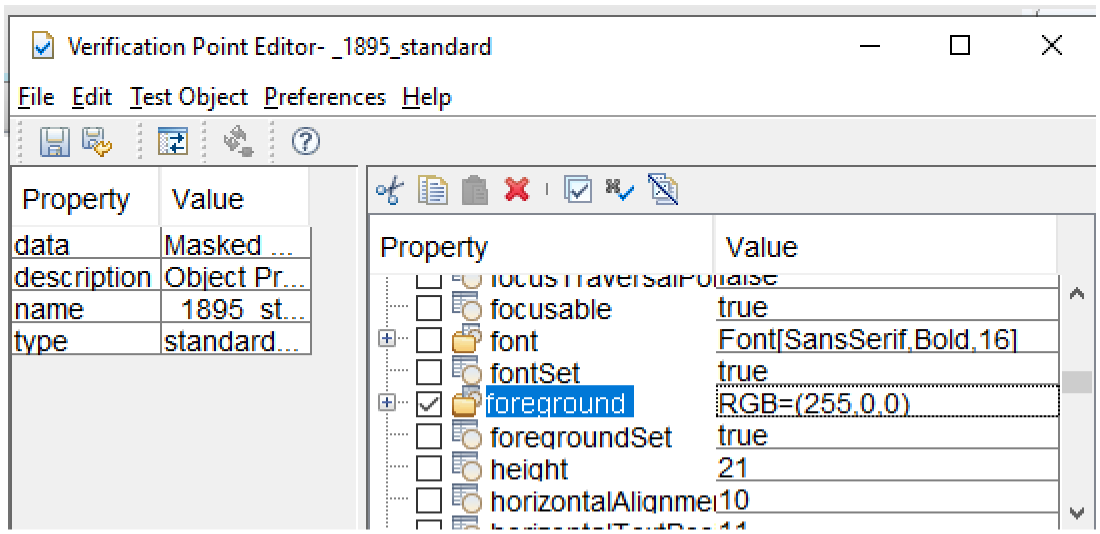
Verification Point Data

Check the properties to include in the verification point.

Property	Value
data	Masked ...
description	Object Pr...
name	Propertie...
type	properties

Property	Value
☐ focusTraversalPolicySet	false
☐ focusable	true
☐ font	Font[SansSerif,Bold,16]
☐ fontSet	true
✓ foreground	RGB=(0,0,0)
☐ foregroundSet	true
☐ height	21
☐ horizontalAlignment	10
☐ horizontalTextPosition	11
☐ iconTextGap	4
☐ ignoreRepaint	false
☐ inheritsPopupMenu	true

However, we can edit the actual baseline value like this:

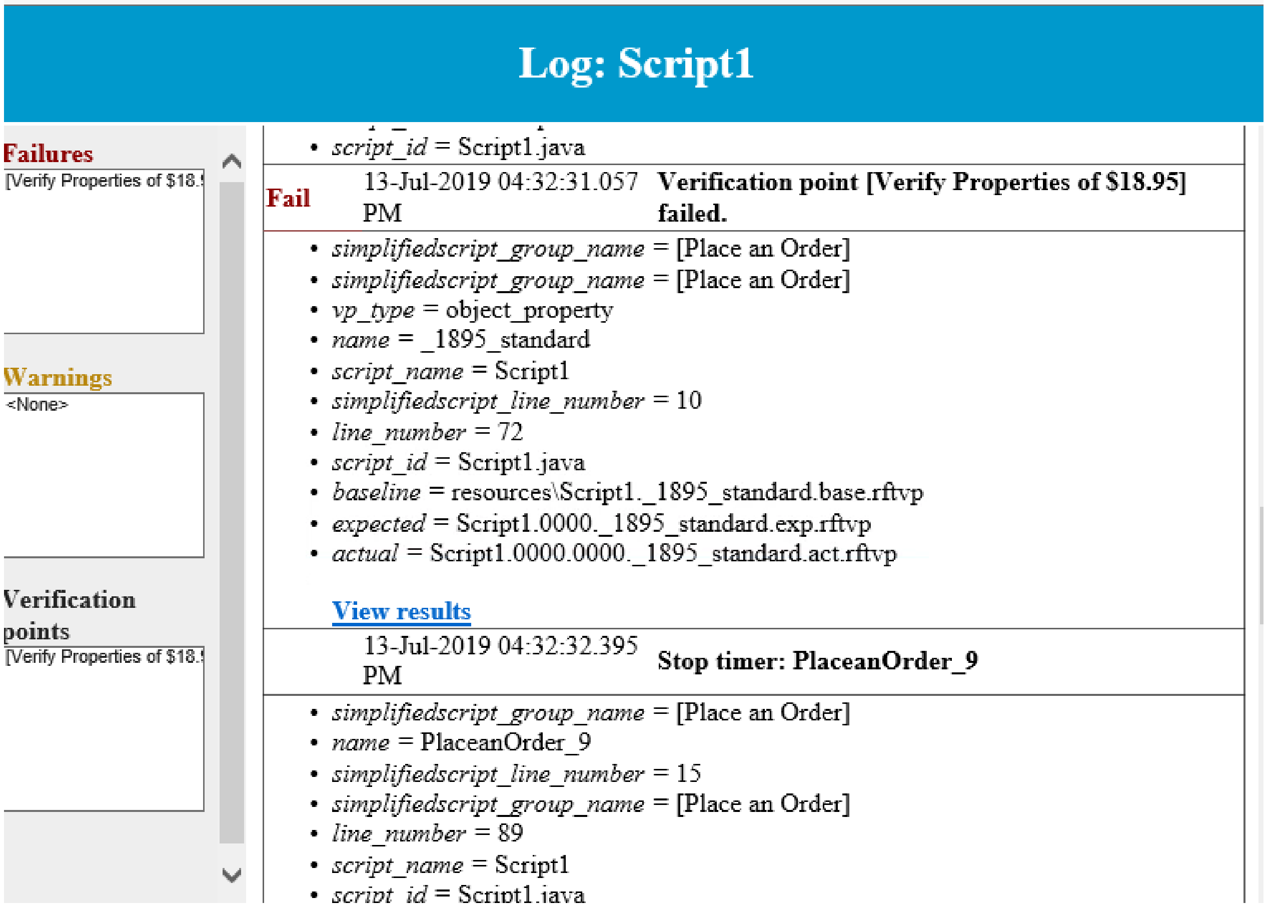

Verification Point Editor- _1895_standard

Property	Value
data	Masked ...
description	Object Pr...
name	1895_st...
type	standard...

Property	Value
☐ focusTraversalPol	false
☐ focusable	true
☐ font	Font[SansSerif,Bold,16]
☐ fontSet	true
✓ foreground	RGB=(255,0,0)
☐ foregroundSet	true
☐ height	21
☐ horizontalAlignme	10
☐ horizontalTextPos	11

Which now says that the text for the total amount should be in red.

Running this edited script produces the following log file.



```
• script_id = Script1.java
Fail  13-Jul-2019 04:32:31.057 PM  Verification point [Verify Properties of $18.95] failed.
• simplifiedscript_group_name = [Place an Order]
• simplifiedscript_group_name = [Place an Order]
• vp_type = object_property
• name = _1895_standard
• script_name = Script1
• simplifiedscript_line_number = 10
• line_number = 72
• script_id = Script1.java
• baseline = resources\Script1._1895_standard.base.rftvp
• expected = Script1.0000._1895_standard.exp.rftvp
• actual = Script1.0000.0000._1895_standard.act.rftvp

View results
      13-Jul-2019 04:32:32.395 PM  Stop timer: PlaceanOrder_9
• simplifiedscript_group_name = [Place an Order]
• name = PlaceanOrder_9
• simplifiedscript_line_number = 15
• simplifiedscript_group_name = [Place an Order]
• line_number = 89
• script_name = Script1
• script_id = Script1.java
```

In this case, because the displayed font wasn’t red, the verification point failed.

5 Test Object Maps

When we record a script, RFT captures and stores information about each of the program objects that the test script interacted with. The result is a hierarchical list of all of the objects the test script interacted with called the *Test Object Map.*

The test object map contains individual items called *test objects* which store certain administrative property and value information about the those object that were interacted with during the script.

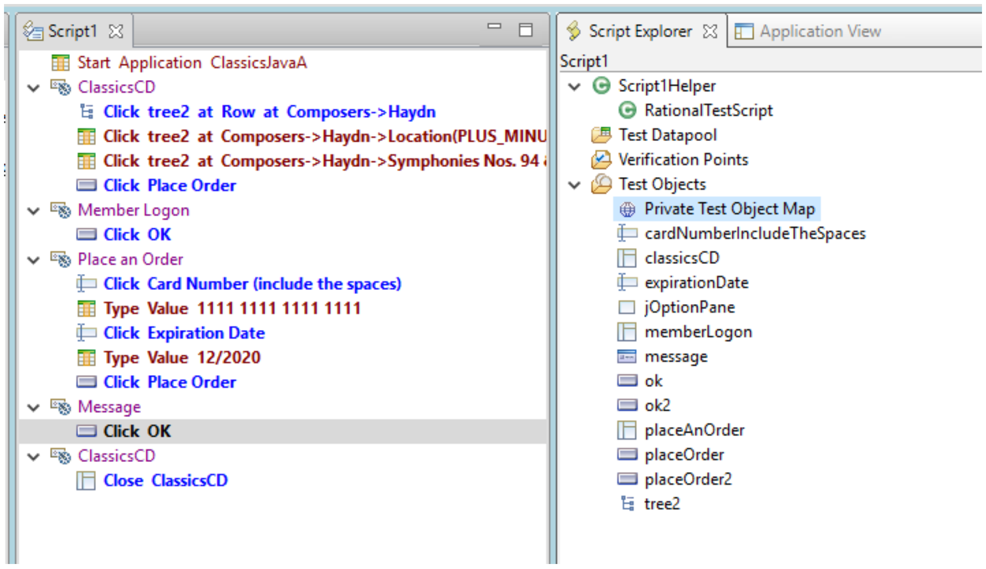
There are two kinds of objects we need to distinguish in an RFT test environment:

1. The actual application run time objects which are executing. These include buttons, text fields, frames and other program objects.

2. Test objects which are records of the properties and characteristics of the actual run time objects as they were encountered when the script was recorded.

Lets consider two scripts. The first is the standard script we have been working with, but we will modify it to produce a second script that selects a few items that the first script does not.

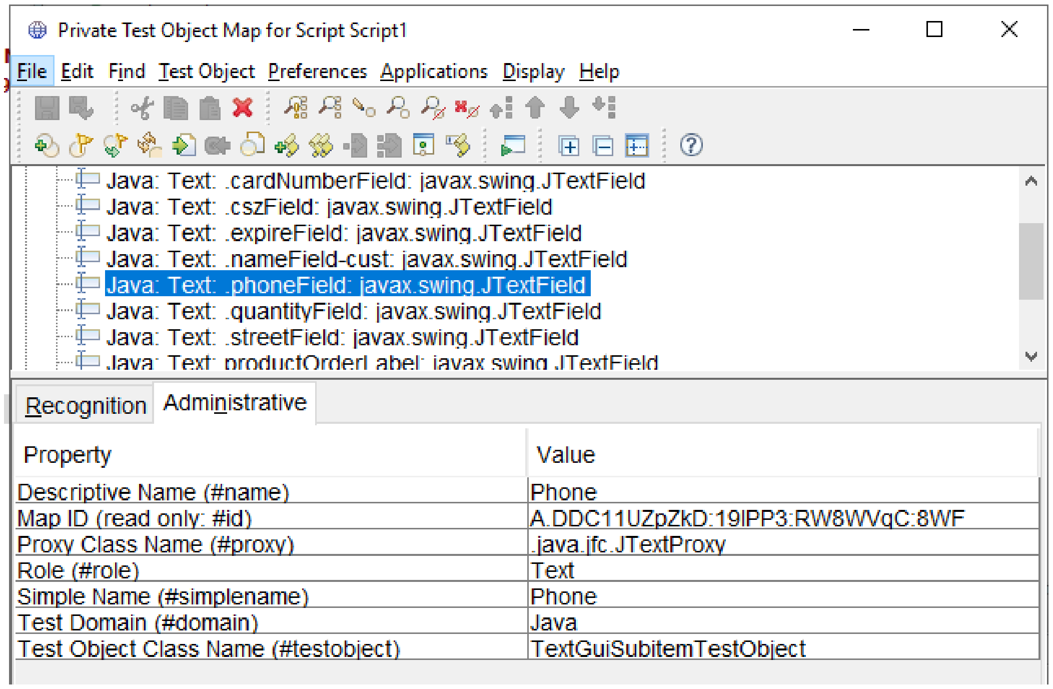
In the screenshot below, it can be seen that objects in the object map correspond to items that were clicked on or typed on.



```
Script1
  Start Application ClassicsJavaA
  ClassicsCD
    Click tree2 at Row at Composers->Haydn
    Click tree2 at Composers->Haydn->Location(PLUS_MINU
    Click tree2 at Composers->Haydn->Symphonies Nos. 94
    Click Place Order
  Member Logon
    Click OK
  Place an Order
    Click Card Number (include the spaces)
    Type Value 1111 1111 1111 1111
    Click Expiration Date
    Type Value 12/2020
    Click Place Order
  Message
    Click OK
  ClassicsCD
    Close ClassicsCD
```

```
Script Explorer    Application View
Script1
  Script1Helper
    RationalTestScript
  Test Datapool
  Verification Points
  Test Objects
    Private Test Object Map
    cardNumberIncludeTheSpaces
    classicsCD
    expirationDate
    jOptionPane
    memberLogon
    message
    ok
    ok2
    placeAnOrder
    placeOrder
    placeOrder2
    tree2
```

If we open up any of these objects, then we get a list of the values an properties associ-ated with the run time object. For example for the phone number field:

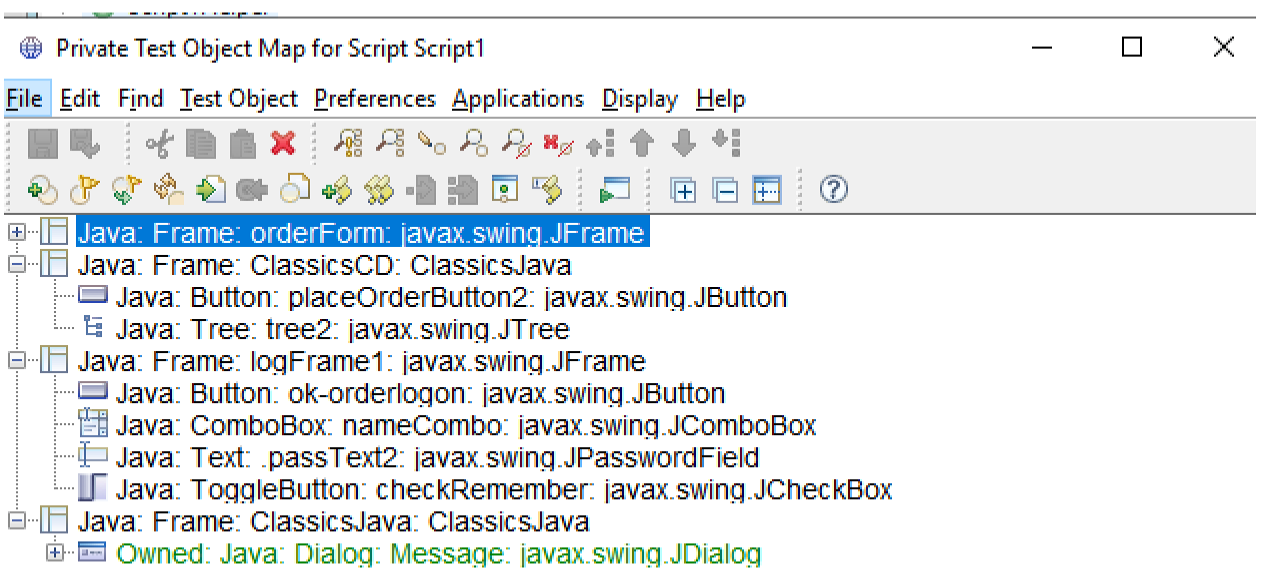

Private Test Object Map for Script Script1
File Edit Find Test Object Preferences Applications Display Help
Recognition Administrative

```
Java: Text: .cardNumberField: javax.swing.JTextField
Java: Text: .cszField: javax.swing.JTextField
Java: Text: .expireField: javax.swing.JTextField
Java: Text: .nameField-cust: javax.swing.JTextField
Java: Text: .phoneField: javax.swing.JTextField
Java: Text: .quantityField: javax.swing.JTextField
Java: Text: .streetField: javax.swing.JTextField
Java: Text: productOrderLabel: javax.swing.JTextField
```

Property	Value
Descriptive Name (#name)	Phone
Map ID (read only: #id)	A.DDC11UZpZkD:19IPP3:RW8WVqC:8WF
Proxy Class Name (#proxy)	.java.jfc.JTextProxy
Role (#role)	Text
Simple Name (#simplename)	Phone
Test Domain (#domain)	Java
Test Object Class Name (#testobject)	TextGuiSubitemTestObject

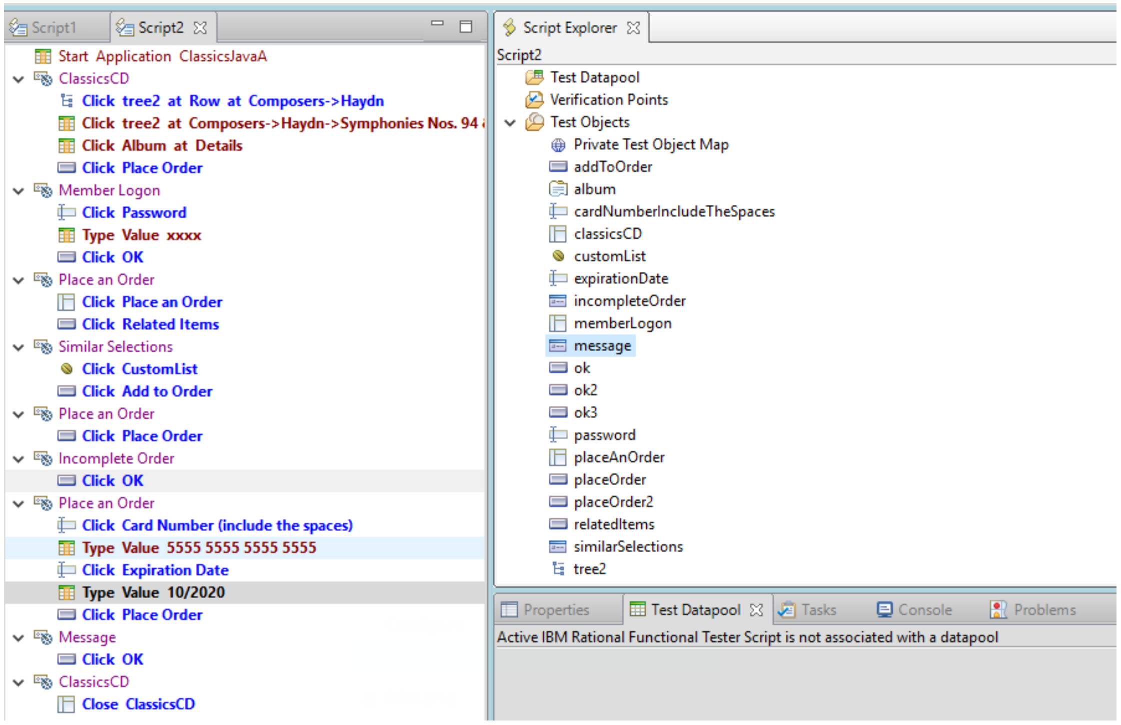
The properties that are of interest to the map are the ones that tell RFT how this objects fits into the hierarchy of objects the script will encounter. If we want to get lower level properties, like the font used, to use in a verification point, we have to use one of the tools, like the verification wizard, to collect that information. The actual test object map is shown on the next page.

Every time we record a script, a test object map is created called the private test object map.


Private Test Object Map for Script Script1
File Edit Find Test Object Preferences Applications Display Help

```
Java: Frame: orderForm: javax.swing.JFrame
Java: Frame: ClassicsCD: ClassicsJava
    Java: Button: placeOrderButton2: javax.swing.JButton
    Java: Tree: tree2: javax.swing.JTree
Java: Frame: logFrame1: javax.swing.JFrame
    Java: Button: ok-orderlogon: javax.swing.JButton
    Java: ComboBox: nameCombo: javax.swing.JComboBox
    Java: Text: .passText2: javax.swing.JPasswordField
    Java: ToggleButton: checkRemember: javax.swing.JCheckBox
Java: Frame: ClassicsJava: ClassicsJava
    Owned: Java: Dialog: Message: javax.swing.JDialog
```

Creating a second script that interacts with a couple of widgets that were not used in Script1 produces the following:



```
Start Application ClassicsJavaA
ClassicsCD
    Click tree2 at Row at Composers->Haydn
    Click tree2 at Composers->Haydn->Symphonies Nos. 94
    Click Album at Details
    Click Place Order
Member Logon
    Click Password
    Type Value xxxx
    Click OK
Place an Order
    Click Place an Order
    Click Related Items
Similar Selections
    Click CustomList
    Click Add to Order
Place an Order
    Click Place Order
Incomplete Order
    Click OK
Place an Order
    Click Card Number (include the spaces)
    Type Value 5555 5555 5555 5555
    Click Expiration Date
    Type Value 10/2020
    Click Place Order
Message
    Click OK
ClassicsCD
    Close ClassicsCD
```

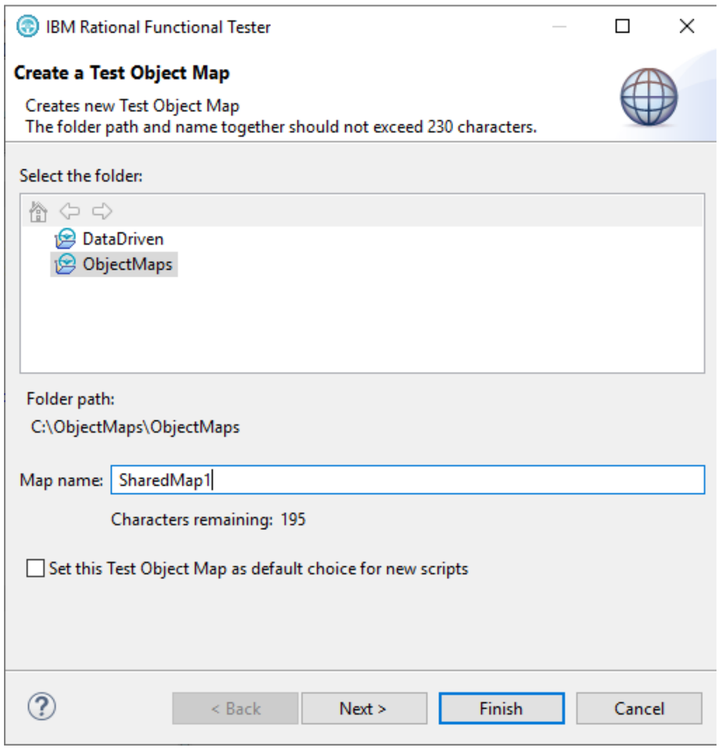
```
Script2
    Test Datapool
    Verification Points
    Test Objects
        Private Test Object Map
        addToOrder
        album
        cardNumberIncludeTheSpaces
        classicsCD
        customList
        expirationDate
        incompleteOrder
        memberLogon
        message
        ok
        ok2
        ok3
        password
        placeAnOrder
        placeOrder
        placeOrder2
        relatedItems
        similarSelections
        tree2
```

Active IBM Rational Functional Tester Script is not associated with a datapool

A quick comparison shows the two test object maps are different because the scripts interacted with different objects.

5.1 Public Object Maps

However, RFT also allows the creation of shared object maps or public maps. To create one, right mouse click on the RFT project in the project explorer and select the "Add Test Object Map" option, which brings up the following dialogue.


IBM Rational Functional Tester

Create a Test Object Map

Creates new Test Object Map
The folder path and name together should not exceed 230 characters.

Select the folder:

DataDriven
ObjectMaps

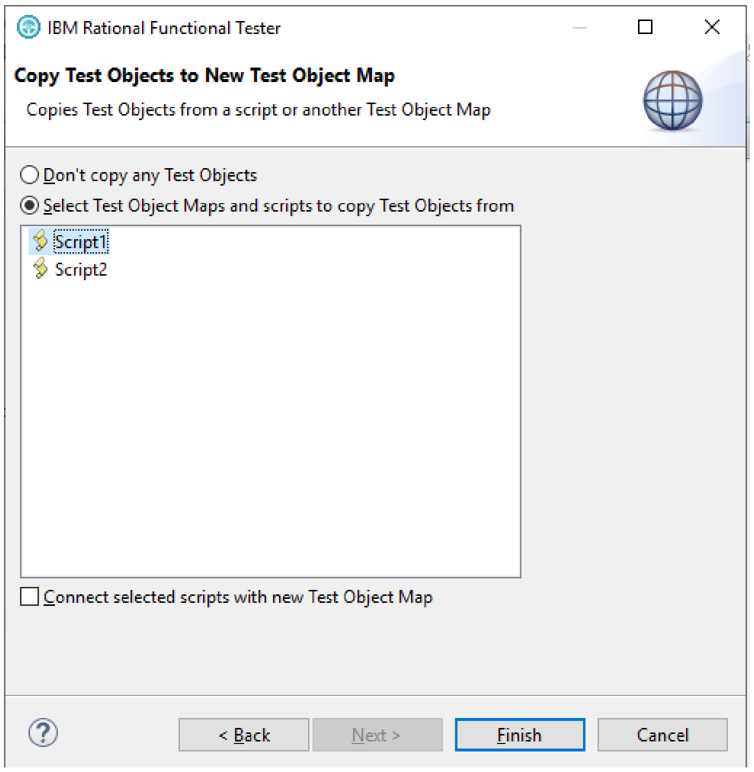
Folder path:
C:\ObjectMaps\ObjectMaps

Map name: SharedMap1

Characters remaining: 195

☐ Set this Test Object Map as default choice for new scripts

Selecting "Next" gives the the option of creating an empty test object map or basing it on the the private map in one of the existing scripts. For this example, we will chose the object map from Script1 as shown on the next page.


IBM Rational Functional Tester

Copy Test Objects to New Test Object Map

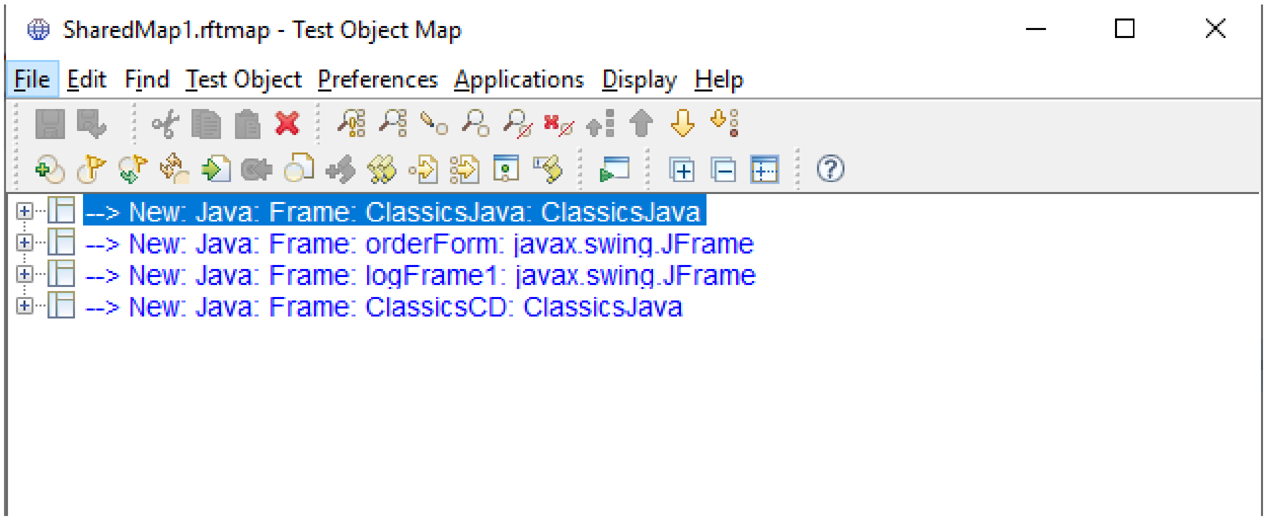
Copies Test Objects from a script or another Test Object Map

( ) Don't copy any Test Objects
(●) Select Test Object Maps and scripts to copy Test Objects from

Script1
Script2

[ ] Connect selected scripts with new Test Object Map

And the new shared map object is created and ready for use

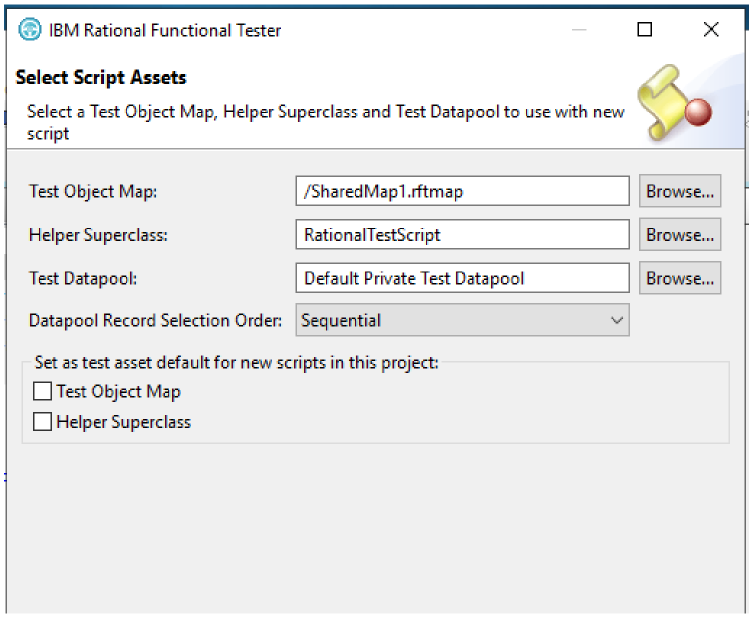

SharedMap1.rftmap - Test Object Map

File Edit Find Test Object Preferences Applications Display Help

--> New: Java: Frame: ClassicsJava: ClassicsJava
--> New: Java: Frame: orderForm: javax.swing.JFrame
--> New: Java: Frame: logFrame1: javax.swing.JFrame
--> New: Java: Frame: ClassicsCD: ClassicsJava

5.2 Using a Shared Map

Now that we have a shared map, it can be used in place of a generated private map. To see this, a new script is created in the recorder, but selecting "Next" gives us the option of using a shared test object map.


IBM Rational Functional Tester

Select Script Assets

Select a Test Object Map, Helper Superclass and Test Datapool to use with new script

Test Object Map: /SharedMap1.rftmap

Helper Superclass: RationalTestScript

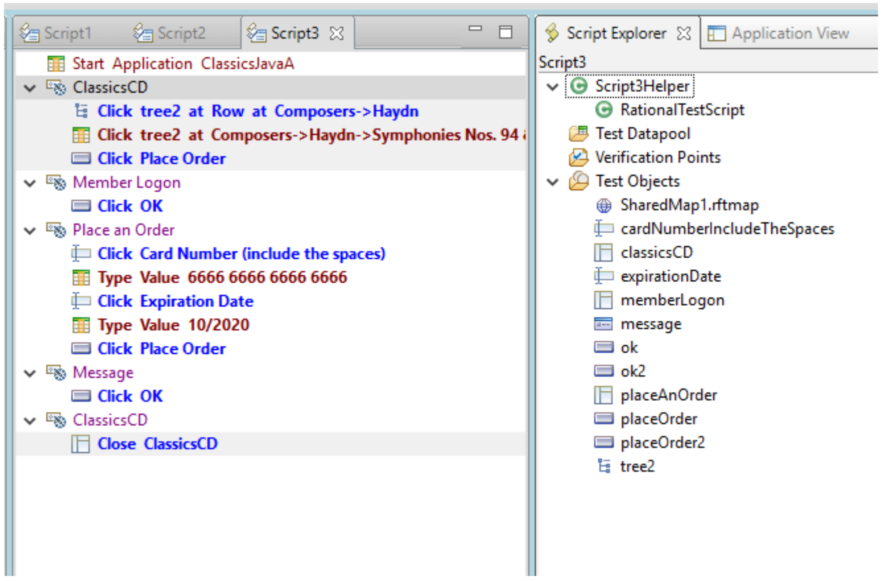
Test Datapool: Default Private Test Datapool

Datapool Record Selection Order: Sequential

Set as test asset default for new scripts in this project:

- [ ] Test Object Map
- [ ] Helper Superclass

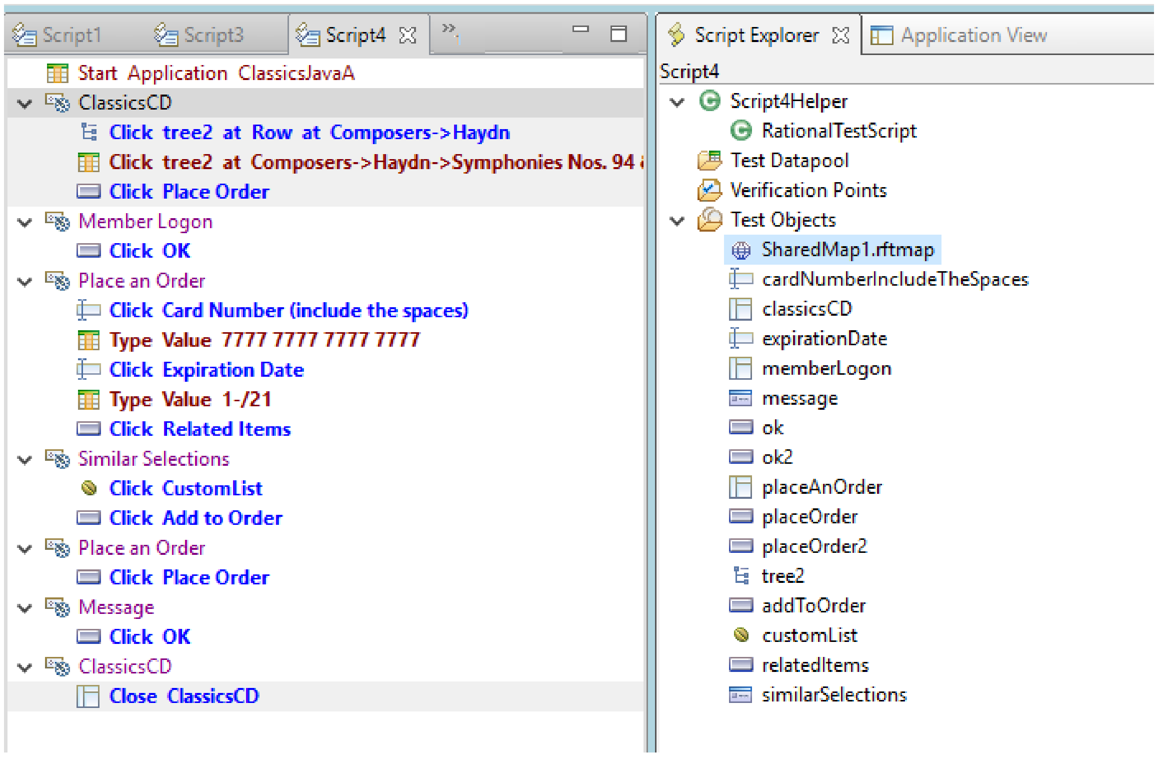
Which, if we record Script3 to be the same as Script1, then we get the following:



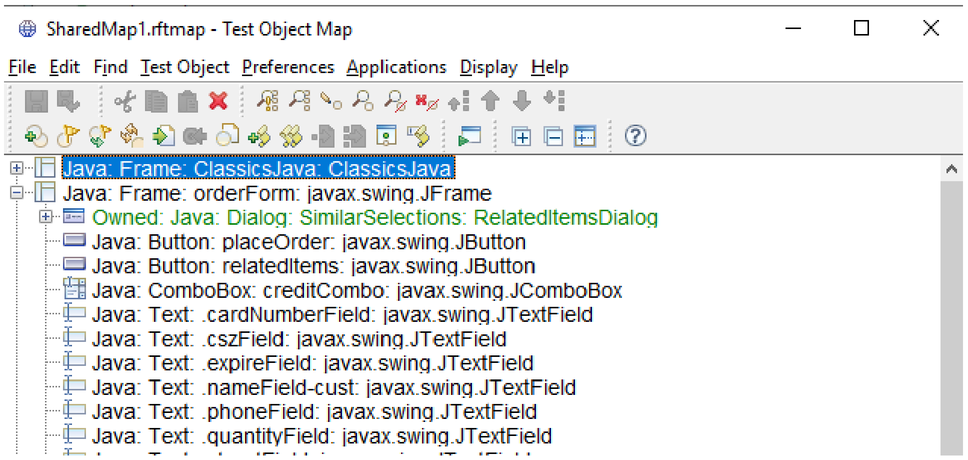
```
Start Application ClassicsJavaA
ClassicsCD
    Click tree2 at Row at Composers->Haydn
    Click tree2 at Composers->Haydn->Symphonies Nos. 94
    Click Place Order
Member Logon
    Click OK
Place an Order
    Click Card Number (include the spaces)
    Type Value 6666 6666 6666 6666
    Click Expiration Date
    Type Value 10/2020
    Click Place Order
Message
    Click OK
ClassicsCD
    Close ClassicsCD
```

```
Script3
  Script3Helper
    RationalTestScript
  Test Datapool
  Verification Points
  Test Objects
    SharedMap1.rftmap
    cardNumberIncludeTheSpaces
    classicsCD
    expirationDate
    memberLogon
    message
    ok
    ok2
    placeAnOrder
    placeOrder
    placeOrder2
    tree2
```

If we record Script4 using the same shared object map and click on something new (in this case the "similar selections" button), the shared map is updated. For example, this is how the object map looks in Script4:



But if we look at the object map in Script3, it has also been updated. In other words, when we use a shared object map in a script and the script changes it, those changes are available to all the other scripts that share that object map.


SharedMap1.rftmap - Test Object Map
File Edit Find Test Object Preferences Applications Display Help

Java: Frame: ClassicsJava: ClassicsJava
Java: Frame: orderForm: javax.swing.JFrame
Owned: Java: Dialog: SimilarSelections: RelatedItemsDialog
Java: Button: placeOrder: javax.swing.JButton
Java: Button: relatedItems: javax.swing.JButton
Java: ComboBox: creditCombo: javax.swing.JComboBox
Java: Text: .cardNumberField: javax.swing.JTextField
Java: Text: .cszField: javax.swing.JTextField
Java: Text: .expireField: javax.swing.JTextField
Java: Text: .nameField-cust: javax.swing.JTextField
Java: Text: .phoneField: javax.swing.JTextField
Java: Text: .quantityField: javax.swing.JTextField

5.2.1 Why Shared Test Object Maps?

The fundamental reason for using a shared text map is to allow more efficient updating of test objects. There are a wide range of operations we can perform on test maps including merging test maps, adding objects dynamically and altering test objects. These operations are out of scope for this course so we won't be covering them.

However, if each script could only use its private test object map, then any changes that would be common to all the scripts' test object maps would have to be done one by one – a very tedious, time consuming and error prone process.

By having related scripts use a common test object map, it allows changes to be made once and in one place, which then propagate across all the scripts that share that test object map.

Data Driven Testing

Module Four

6 What a Datapool Does – Data Driving Tests

Looking at the scripts we generated so far, we can see that the data in the scripts is hard coded, which means that every time we run the script, the same data is used every time.

When you data-drive a test, the script uses variables for key application input fields and programs instead of literal values so that you can use external data to drive the application you are testing.

Data-driven testing uses data from an external file, a *datapool*, as input to a test. A datapool is a collection of related data records which supplies data values to the variables in a test script during test script playback.

Because data is separated from the test script, you can:

1. Modify test data without affecting the test script
2. Add new test cases by modifying the data, not the test script
3. Share the test data with many test scripts

Here are some examples of problems that data driving tests solve:

Problem: *During recording, you create a personnel file for a new employee, using the employee's unique social security number. Each time the test is run, there is an attempt to create the same personnel file and supply the same social security number. The application rejects the duplicate requests.*

Solution: You can data-drive the test script to send different employee data, including social security numbers, to the server each time the test is run.

Problem*: You delete a record during recording. When you run the test, Functional Tester attempts to delete the same record, and "Record Not Found" errors result.*

Solution: You can data-drive the test script to reference a different record in the deletion request each time the script is played back.

Problem: *You delete a record during recording. During playback, each transaction attempts to delete the same record, and "Record Not Found" errors are returned.*

Solution: You can use a datapool to reference a different record in the deletion request each time the transaction repeats.

6.1.1 Datapools

A datapool is a test dataset, a collection of related data records which supplies data values to the variables in a test script during test script playback.

When you create a data-driven test by using Functional Tester, you select the objects in an application-under-test to data-drive. Functional Tester creates a datapool in which you can edit and add data. You can use a single test script repeatedly with varying input and response data.

You can use the datapool feature in several ways:

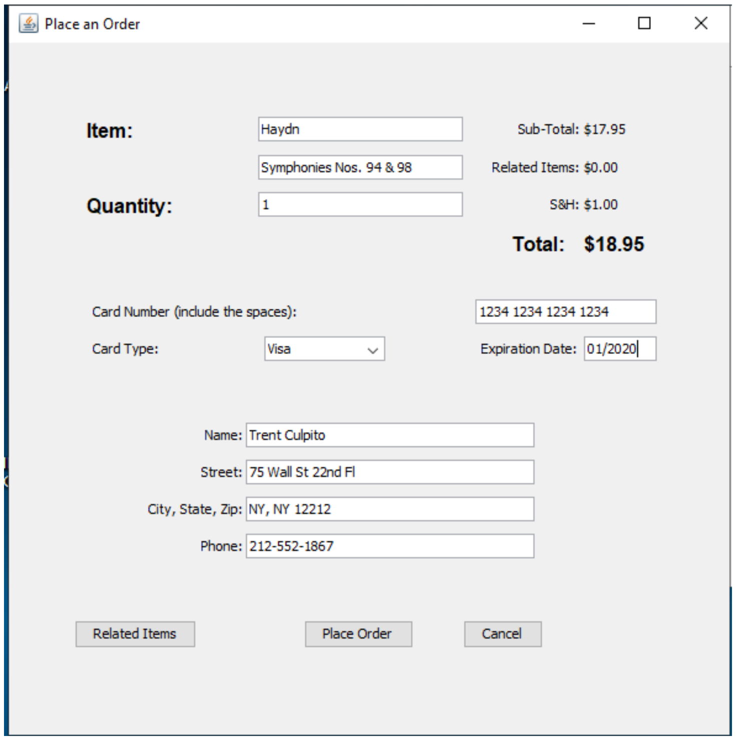
1. To add realistic data to a test script
2. To import data from a Functional Tester datapool, an IBM Rational Quality Manager datapool, or a .csv file created using a spreadsheet application.
3. To create a datapool manually and add data to a datapool.
4. To edit datapool values or change data types
5. To export datapools to use in other Functional Test projects or to a .csv file to edit in a spreadsheet application.
6. To change the datapool record selection order to determine how the test script accesses an associated datapool during playback.

6.1.2 Private and Shared Datapools

Every test script that you create has a private test datapool associated with it. The initial private test datapool is a placeholder and is empty until you data-drive a test script, or add new data to it. You can create a shared datapool by creating a new datapool, or you can associate a datapool with several test scripts to share a datapool.

6.2 Developing a Data Driven Script

We start with the usual way of creating our script until we get to this point in the recording.



Place an Order

Item:	Haydn	Sub-Total: $17.95
	Symphonies Nos. 94 & 98	Related Items: $0.00
Quantity:	1	S&H: $1.00
		Total: $18.95

Card Number (include the spaces): 1234 1234 1234 1234

Card Type: Visa

Expiration Date: 01/2020

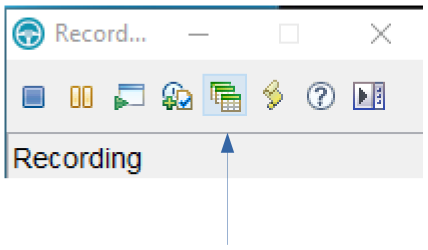
Name: Trent Culpito

Street: 75 Wall St 22nd Fl

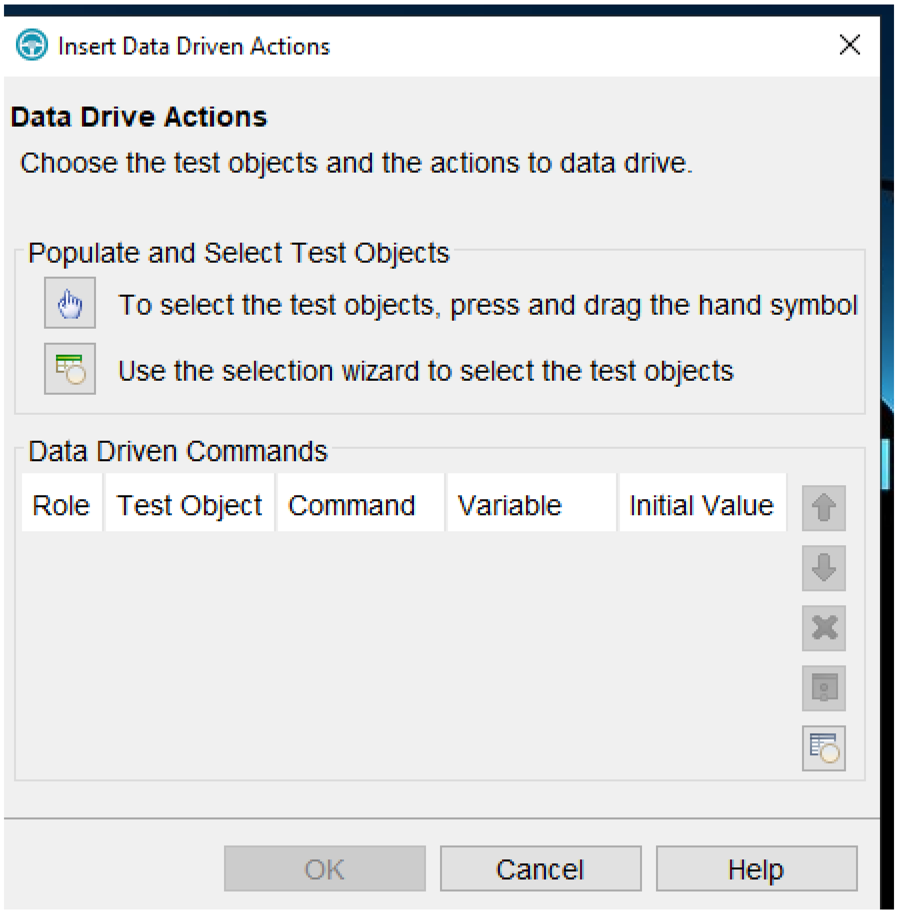
City, State, Zip: NY, NY 12212

Phone: 212-552-1867

On the Recording toolbar, the "Insert Data Driven Commands" option is selected which pauses the recording.



This brings up the "Data Drive Actions" dialog.


Insert Data Driven Actions

Data Drive Actions

Choose the test objects and the actions to data drive.

Populate and Select Test Objects

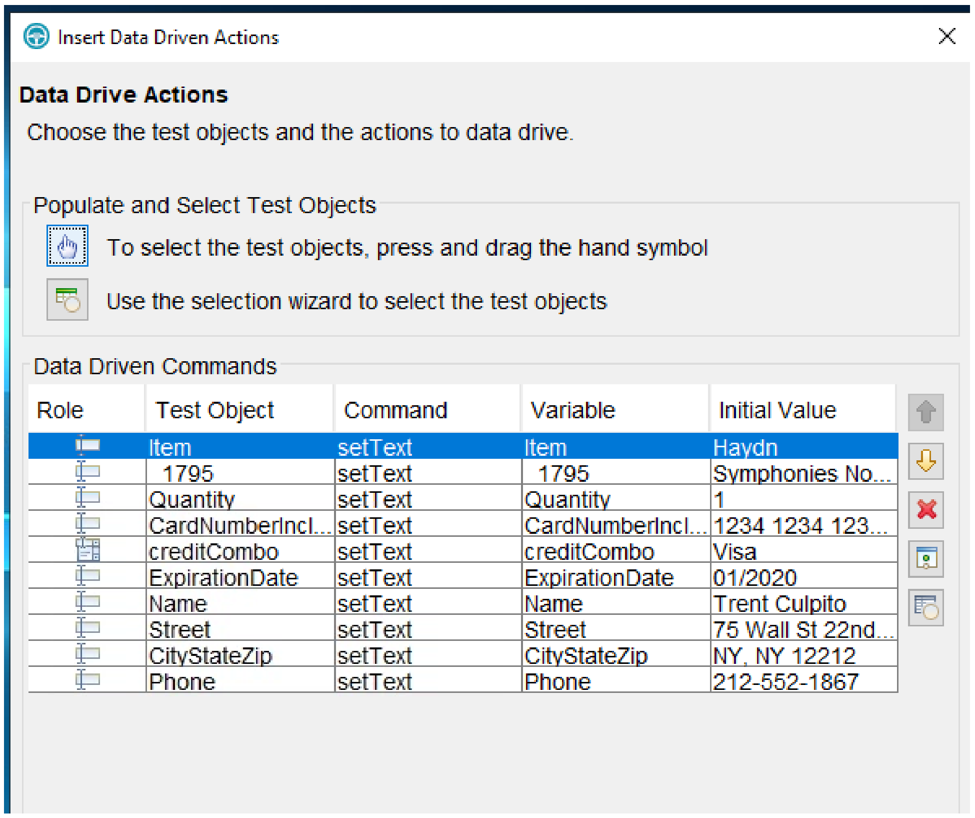
To select the test objects, press and drag the hand symbol

Use the selection wizard to select the test objects

Data Driven Commands

Role	Test Object	Command	Variable	Initial Value

The mouse is used to drag the hand symbol to the title bar of the Place an Order window on the ClassicsCD application. Functional Tester outlines the entire Place an Order window with a red border. Once the mouse button is released, under the DataDriven Commands table, information about the selected objects are displayed.



Insert Data Driven Actions

Data Drive Actions

Choose the test objects and the actions to data drive.

Populate and Select Test Objects

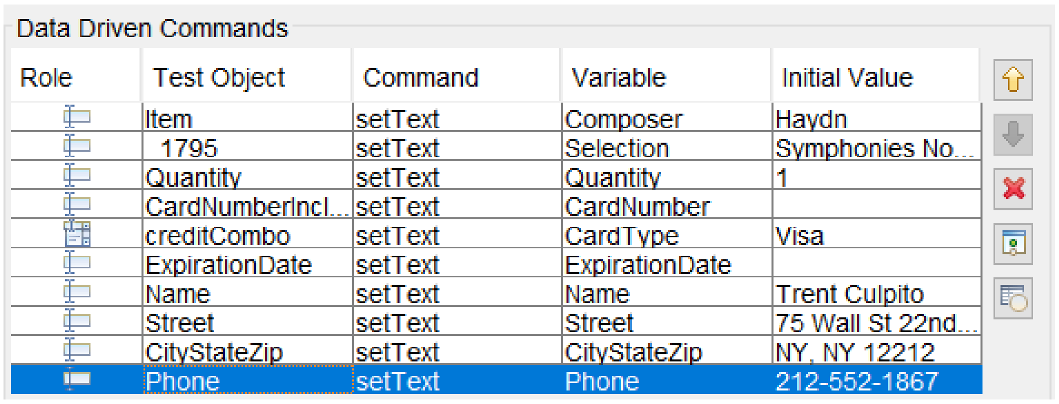
To select the test objects, press and drag the hand symbol

Use the selection wizard to select the test objects

Data Driven Commands

Role	Test Object	Command	Variable	Initial Value
	Item	setText	Item	Haydn
	1795	setText	1795	Symphonies No...
	Quantity	setText	Quantity	1
	CardNumberIncl...	setText	CardNumberIncl...	1234 1234 123...
	creditCombo	setText	creditCombo	Visa
	ExpirationDate	setText	ExpirationDate	01/2020
	Name	setText	Name	Trent Culpito
	Street	setText	Street	75 Wall St 22nd...
	CityStateZip	setText	CityStateZip	NY, NY 12212
	Phone	setText	Phone	212-552-1867

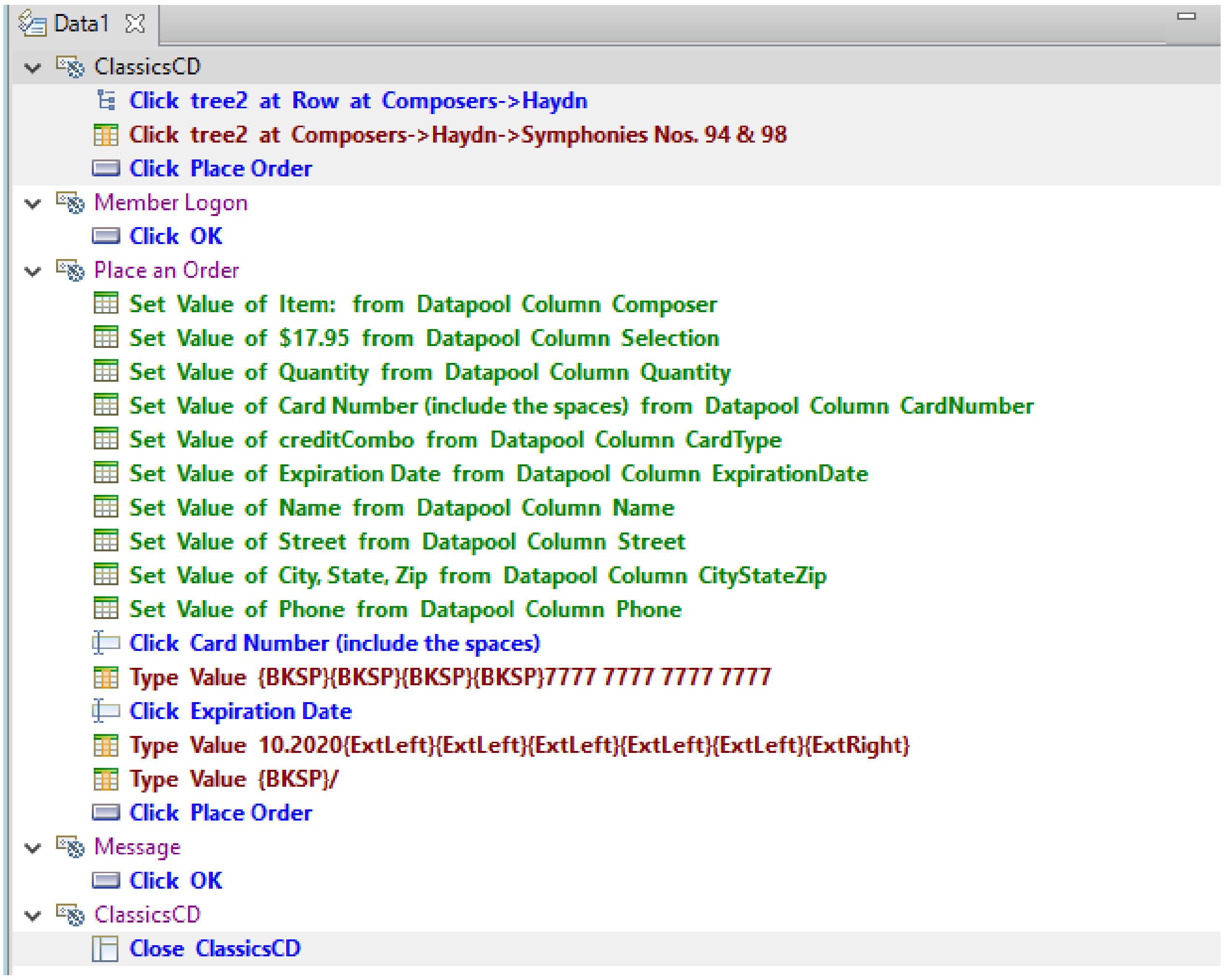
This produces a template of variables that represent the data in the application. Variable names should be edited to produce more meaningful name. As in the following example.



Data Driven Commands

Role	Test Object	Command	Variable	Initial Value
	Item	setText	Composer	Haydn
	1795	setText	Selection	Symphonies No...
	Quantity	setText	Quantity	1
	CardNumberIncl...	setText	CardNumber	
	creditCombo	setText	CardType	Visa
	ExpirationDate	setText	ExpirationDate	
	Name	setText	Name	Trent Culpito
	Street	setText	Street	75 Wall St 22nd...
	CityStateZip	setText	CityStateZip	NY, NY 12212
	Phone	setText	Phone	212-552-1867

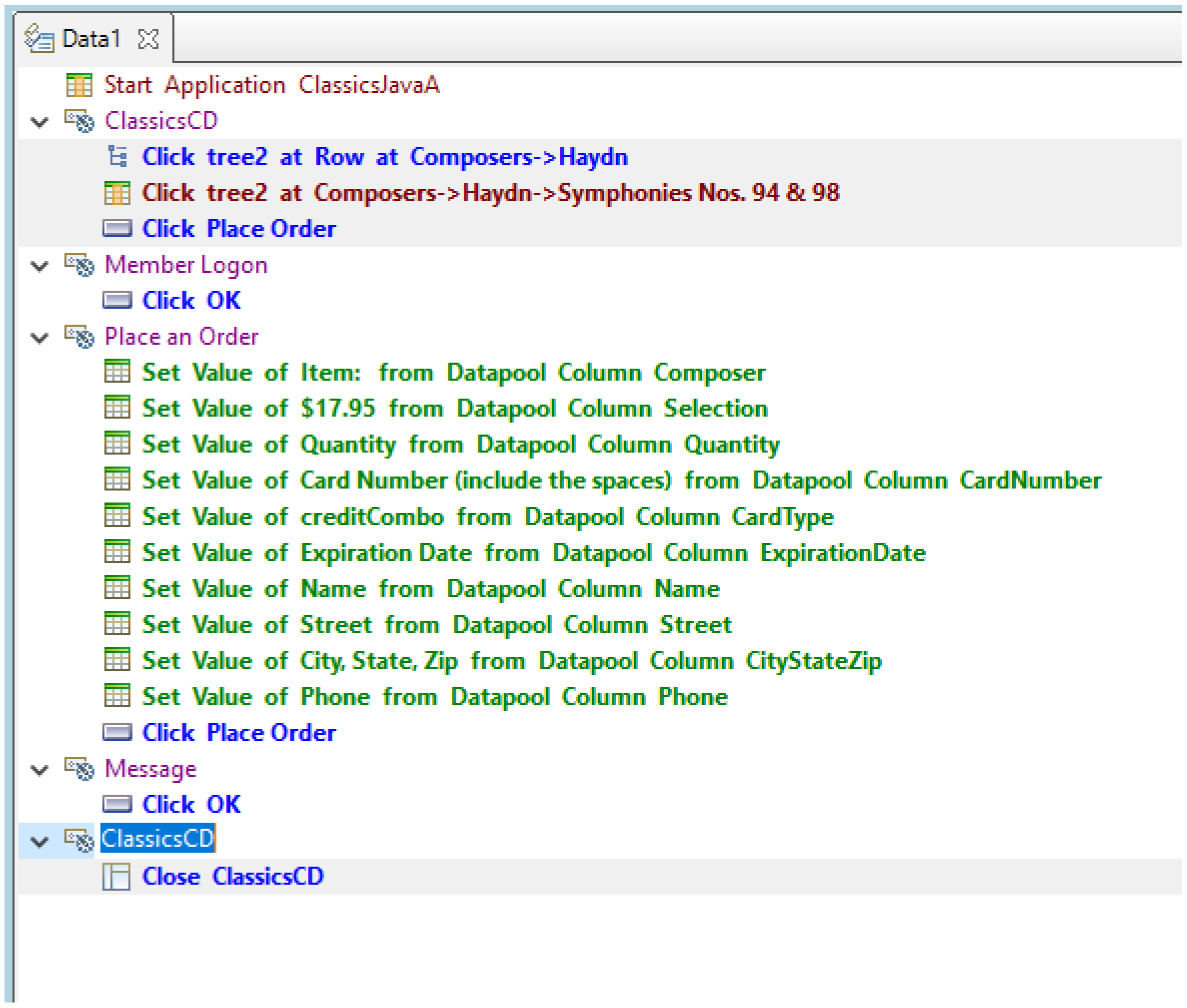
After clicking "OK" we finish off the script as normal to produce the following script listing.



```
Data1
ClassicsCD
    Click tree2 at Row at Composers->Haydn
    Click tree2 at Composers->Haydn->Symphonies Nos. 94 & 98
    Click Place Order
Member Logon
    Click OK
Place an Order
    Set Value of Item: from Datapool Column Composer
    Set Value of $17.95 from Datapool Column Selection
    Set Value of Quantity from Datapool Column Quantity
    Set Value of Card Number (include the spaces) from Datapool Column CardNumber
    Set Value of creditCombo from Datapool Column CardType
    Set Value of Expiration Date from Datapool Column ExpirationDate
    Set Value of Name from Datapool Column Name
    Set Value of Street from Datapool Column Street
    Set Value of City, State, Zip from Datapool Column CityStateZip
    Set Value of Phone from Datapool Column Phone
    Click Card Number (include the spaces)
    Type Value {BKSP}{BKSP}{BKSP}{BKSP}7777 7777 7777 7777
    Click Expiration Date
    Type Value 10.2020{ExtLeft}{ExtLeft}{ExtLeft}{ExtLeft}{ExtLeft}{ExtRight}
    Type Value {BKSP}/
    Click Place Order
Message
    Click OK
ClassicsCD
    Close ClassicsCD
```

However, the typing that was done to complete the script is redundant, so it can be deleted. We don't need the hard coded values because we have created a private data pool that assigns values directly to the underlying program objects instead of reading from the interface widgets.

The data driven script is on the next page.



Data1

Start Application ClassicsJavaA
ClassicsCD
- Click tree2 at Row at Composers->Haydn
- Click tree2 at Composers->Haydn->Symphonies Nos. 94 & 98
- Click Place Order

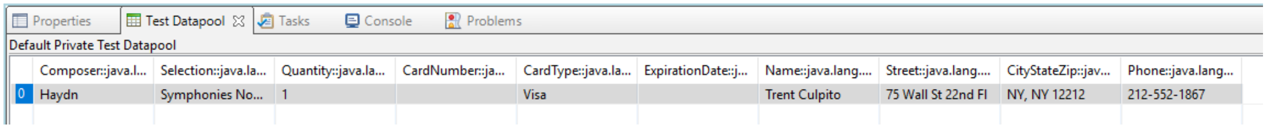
Member Logon
- Click OK

Place an Order
- Set Value of Item: from Datapool Column Composer
- Set Value of $17.95 from Datapool Column Selection
- Set Value of Quantity from Datapool Column Quantity
- Set Value of Card Number (include the spaces) from Datapool Column CardNumber
- Set Value of creditCombo from Datapool Column CardType
- Set Value of Expiration Date from Datapool Column ExpirationDate
- Set Value of Name from Datapool Column Name
- Set Value of Street from Datapool Column Street
- Set Value of City, State, Zip from Datapool Column CityStateZip
- Set Value of Phone from Datapool Column Phone
- Click Place Order

Message
- Click OK

ClassicsCD
- Close ClassicsCD

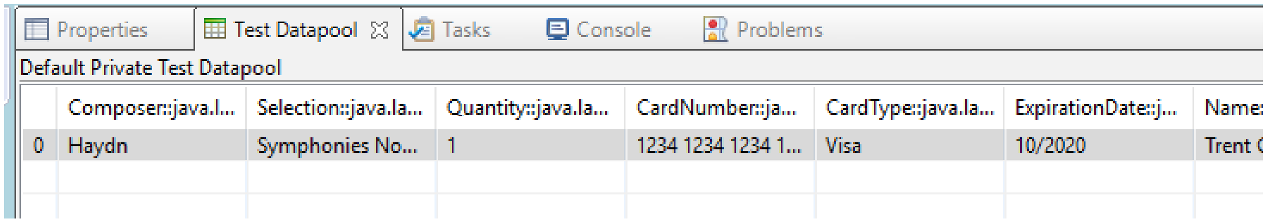
All of the data will come from the private data pool we have just created.



Default Private Test Datapool

	Composer::java.l...	Selection::java.la...	Quantity::java.la...	CardNumber::ja...	CardType::java.la...	ExpirationDate::j...	Name::java.lang....	Street::java.lang....	CityStateZip::jav...	Phone::java.lang...
0	Haydn	Symphonies No...	1		Visa		Trent Culpito	75 Wall St 22nd Fl	NY, NY 12212	212-552-1867

However, since we created this before we entered the credit card info, we need to enter it here in the data pool.



Properties | Test Datapool | Tasks | Console | Problems

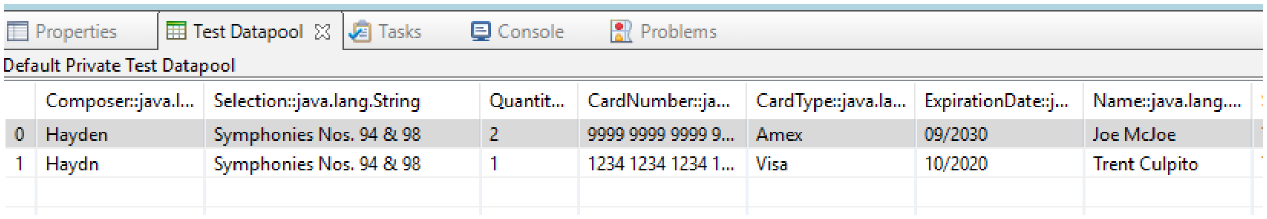
Default Private Test Datapool

	Composer::java.l...	Selection::java.la...	Quantity::java.la...	CardNumber::ja...	CardType::java.la...	ExpirationDate::j...	Name:
0	Haydn	Symphonies No...	1	1234 1234 1234 1...	Visa	10/2020	Trent (

Now we can run the script but the data pool value will be used and the result will be identical to what we had done before.

6.3 Adding Data to the Datapool

The whole point of the data pool is that we can have more than just one set of data to be used. For example, we can manually add a new row to the datapool.



Properties | Test Datapool | Tasks | Console | Problems

Default Private Test Datapool

	Composer::java.l...	Selection::java.lang.String	Quantit...	CardNumber::ja...	CardType::java.la...	ExpirationDate::j...	Name::java.lang....
0	Hayden	Symphonies Nos. 94 & 98	2	9999 9999 9999 9...	Amex	09/2030	Joe McJoe
1	Haydn	Symphonies Nos. 94 & 98	1	1234 1234 1234 1...	Visa	10/2020	Trent Culpito

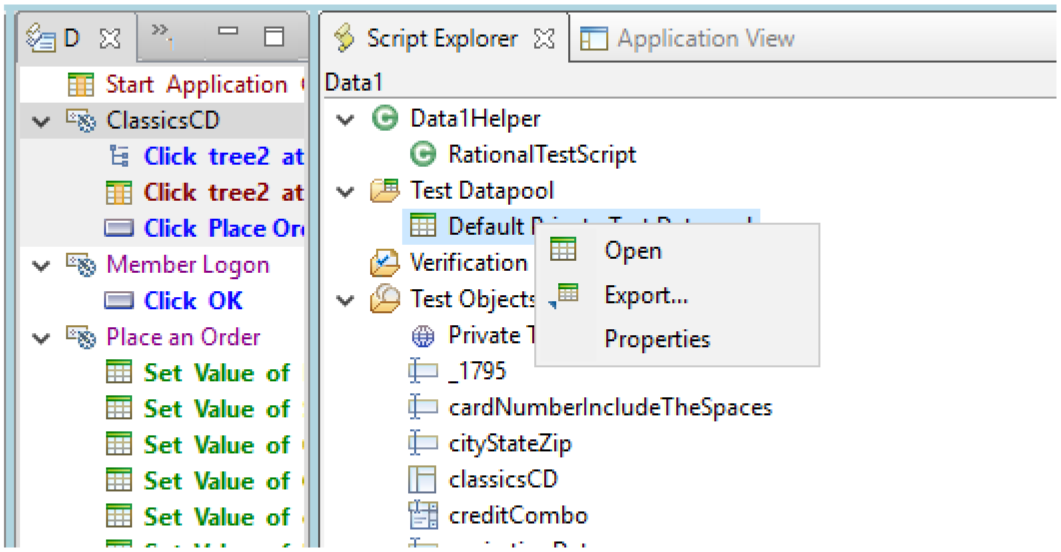
Now we can run the script twice by just using the datapool. To do this, we start the script but we select the "Next" button and choose the number of times we want to run the script. In this case, we are selecting "until we run out of data" which should be only twice because we only have two lines of data.

If you examine the log file, you will see that about halfway through, we can see the first iteration of the script has finished with a pass, and the second iteration has begun. However, watching the playback makes it obvious that the script is executing twice.

By the way, this is the main reason that we always close our application before ending a script. If we don't, then when we are doing multiple script iterations, the application environment will become polluted with multiple copies of the application – something that usually does not have a happy outcome.

6.4 Exporting Datapools

If datapools were only available to the application that created them, they would be of limited use. However, we can export our test data into a CSV file so that we can reuse it.



Is we export this test data as "ClassicsData.csv" we can then use it to create a public datapool.

6.5 Shared Datapools

One of the fundamental best practices in software testing is to decouple the test design (in this case our scripts) from the test data (in this case our datapool).

One of the main reasons this is done is because our test data often represents a high value test artifact which has been carefully designed and created to produce optimal test results with maximal efficiency. The ability to use this test data, or subsets of it, in various test environments is critical to ensure both the thoroughness and the efficiency of the testing process.

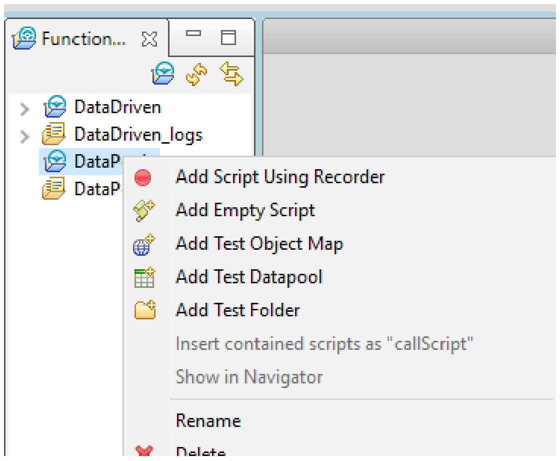
Conversely, a well designed test is of less value when it has a specific test case "hard wired" into it: it becomes a much more powerful test artifact when it can be run with different sets of test data or different test suites.

Shared datapools provide us with exactly those two benefits: we can develop high quality test data that can be reused across multiple scripts, and a single script can be run with different sets of test data that represent different possible scenarios or use cases.

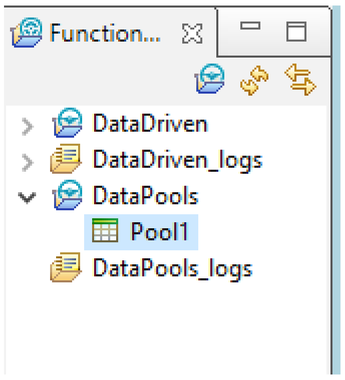
6.5.1 Creating a Public Datapool

To do this, open an RFT Project, here we are calling it "DataPool". First we will create an empty datapool.

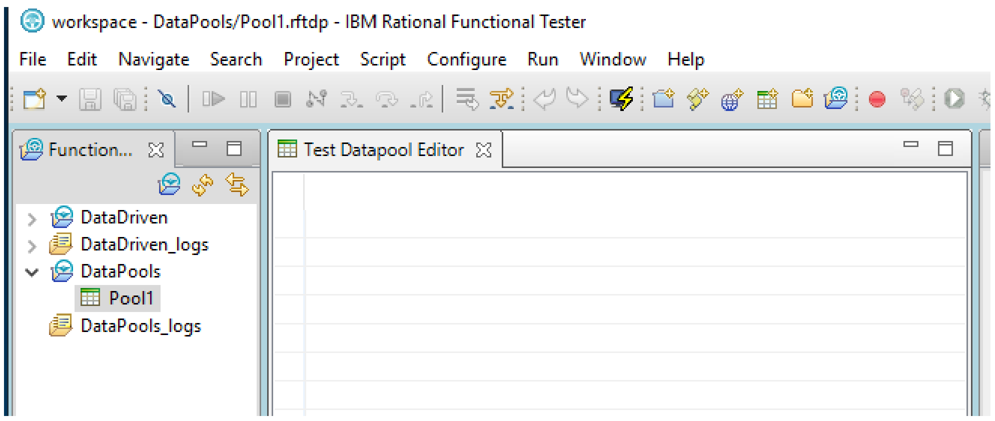
Right mouse click on the project and select the option to add a test datapool.



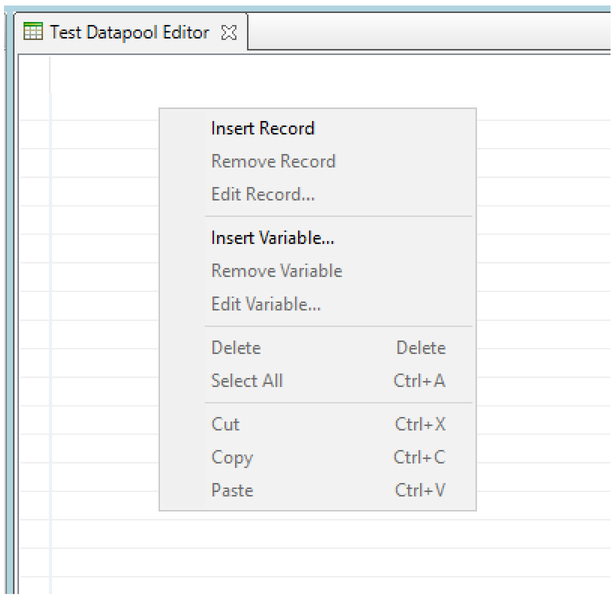
Now we can create an empty datapool by not importing anything. Once created, it appears in the project explorer.



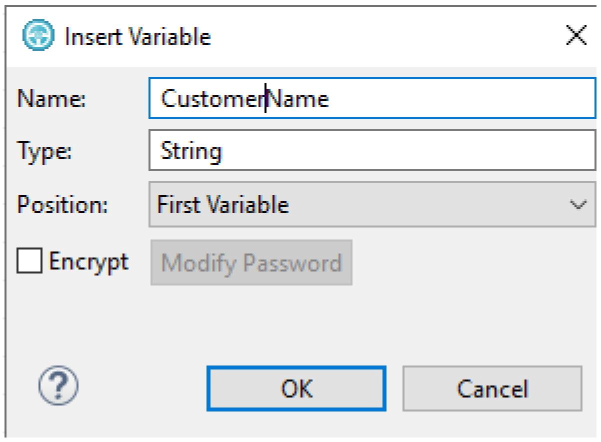
Double clicking on the pool opens the datapool editor.


workspace - DataPools/Pool1.rftdp - IBM Rational Functional Tester
File Edit Navigate Search Project Script Configure Run Window Help
Function...
Test Datapool Editor
DataDriven
DataDriven_logs
DataPools
Pool1
DataPools_logs

We can add variables (columns) by right mouse clicking and adding a variable.

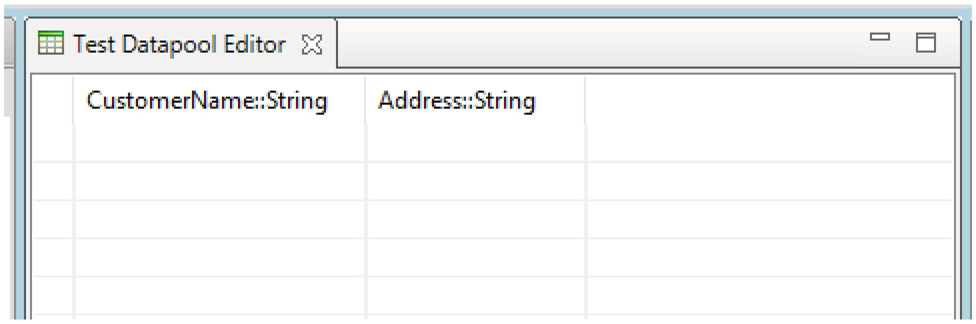


Selecting "Insert Variable" brings up the variable dialogue. Generally, most of the variables we define will be strings.



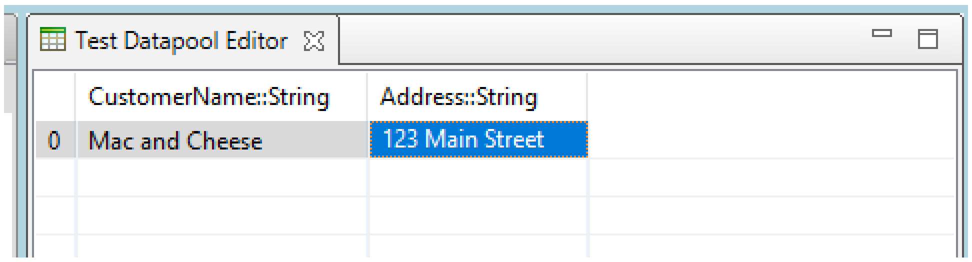
Name: CustomerName
Type: String
Position: First Variable

Repeating this with CustomerAddress we get the following:



	CustomerName::String	Address::String

Data (rows) can be entered using the Insert Record option from the right mouse button menu. This creates a row in the pool that we can now add data to.



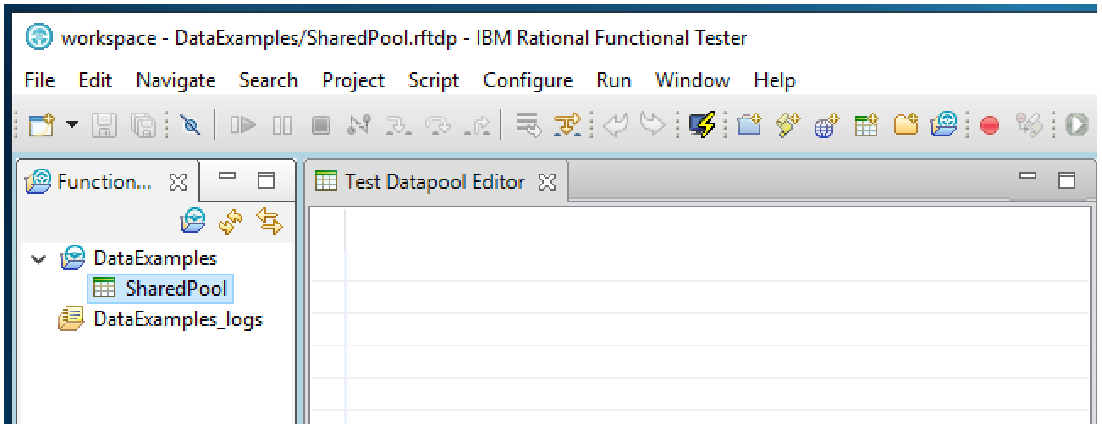
	CustomerName::String	Address::String
0	Mac and Cheese	123 Main Street

6.6 A Simpler Approach

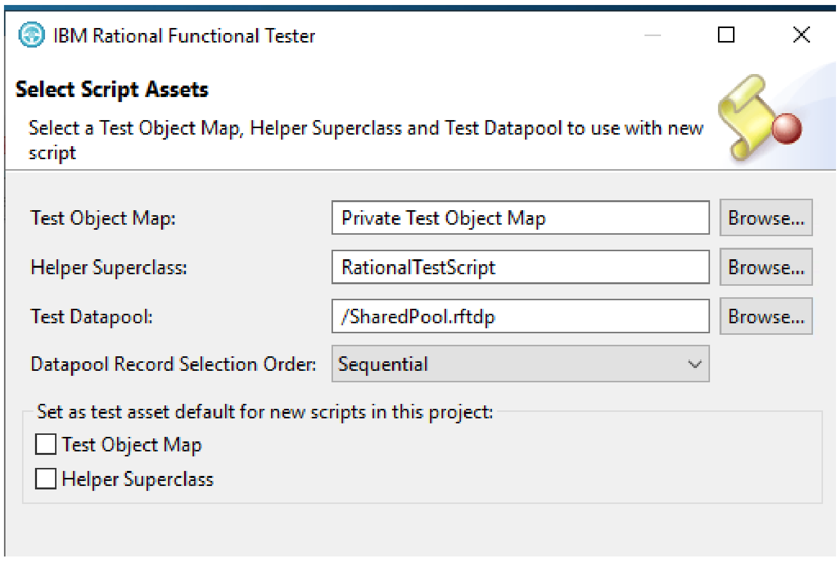
Creating shared datapools from scratch isn't a lot a of fun as well as being prone to error. There is a simpler way to create a shared datapool.

In this example, we will create a shared data pool via a script and also insert a verification point in the process that we will use in the next section.

Starting with a new RFT project, we create an empty datapool as we did previously.


workspace - DataExamples/SharedPool.rftdp - IBM Rational Functional Tester
File Edit Navigate Search Project Script Configure Run Window Help
Function...
Test Datapool Editor
DataExamples
SharedPool
DataExamples_logs

Just like test object maps, we can use a script to populate the data pool for us. Create a new script, but in the "Next" options, select the SharedPool as the datapool to be used.



Select Script Assets

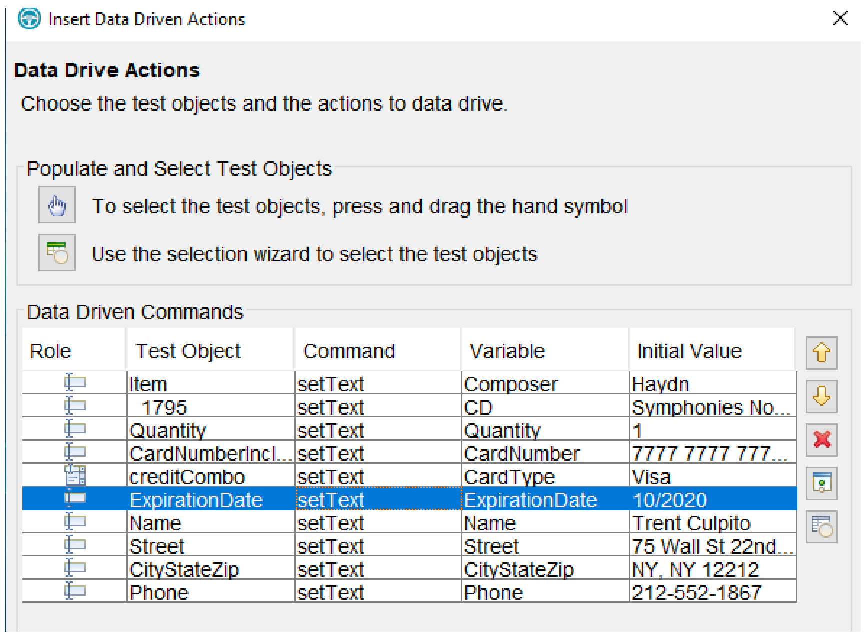
Select a Test Object Map, Helper Superclass and Test Datapool to use with new script

Test Object Map:	Private Test Object Map	Browse...
Helper Superclass:	RationalTestScript	Browse...
Test Datapool:	/SharedPool.rftdp	Browse...
Datapool Record Selection Order:	Sequential	

Set as test asset default for new scripts in this project:
- ☐ Test Object Map
- ☐ Helper Superclass

Recording the scrip exactly like we have the in the past, we make sure that:

We record the datapool structure like we did before.



Insert Data Driven Actions

Data Drive Actions

Choose the test objects and the actions to data drive.

Populate and Select Test Objects

To select the test objects, press and drag the hand symbol

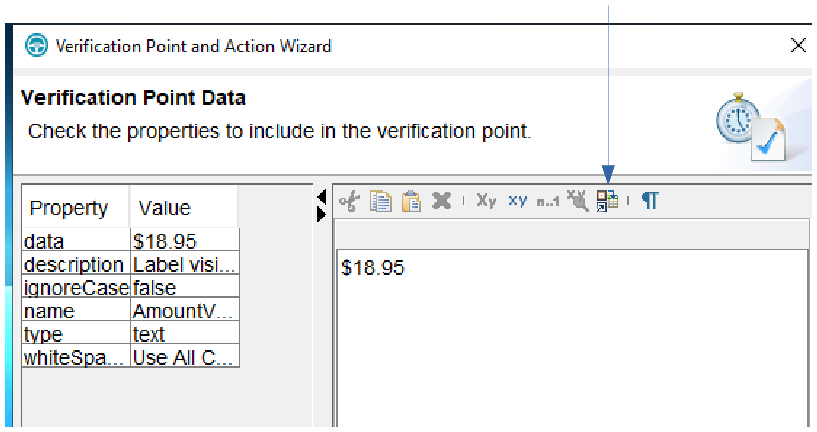
Use the selection wizard to select the test objects

Data Driven Commands

Role	Test Object	Command	Variable	Initial Value
	Item	setText	Composer	Haydn
	1795	setText	CD	Symphonies No...
	Quantity	setText	Quantity	1
	CardNumberIncl...	setText	CardNumber	7777 7777 777...
	creditCombo	setText	CardType	Visa
	ExpirationDate	setText	ExpirationDate	10/2020
	Name	setText	Name	Trent Culpito
	Street	setText	Street	75 Wall St 22nd...
	CityStateZip	setText	CityStateZip	NY, NY 12212
	Phone	setText	Phone	212-552-1867

And we record a verification point to verify the data value for amount. However, we don't want this hardcoded into script since each record in the datapool will have a different amount that is expected.

We start with the adding the verification point as before, but when we get to this point we select the button indicated which replaces this hardcoded value with something from the datapool,



Verification Point and Action Wizard

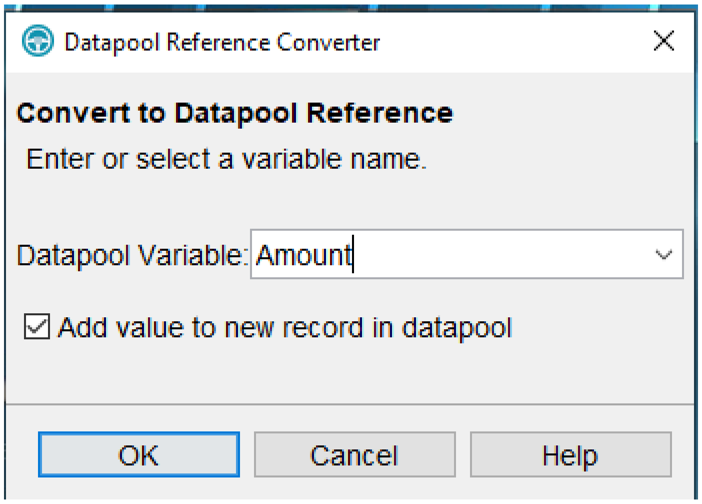
Verification Point Data

Check the properties to include in the verification point.

Property	Value
data	$18.95
description	Label visi...
ignoreCase	false
name	AmountV...
type	text
whiteSpa...	Use All C...

$18.95

This brings up this dialog box:



Datapool Reference Converter

Convert to Datapool Reference

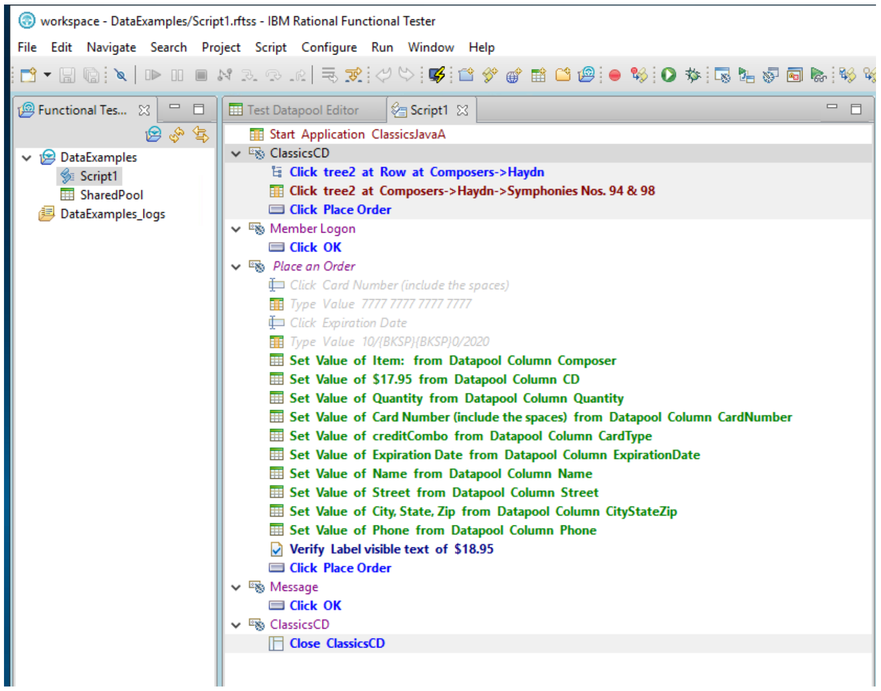
Enter or select a variable name.

Datapool Variable: Amount

[✓] Add value to new record in datapool

Which means that a new variable called "Amount" will be added to the datapool with the value that has been captured off the screen.

When the script is finished being recorded, then we have this:

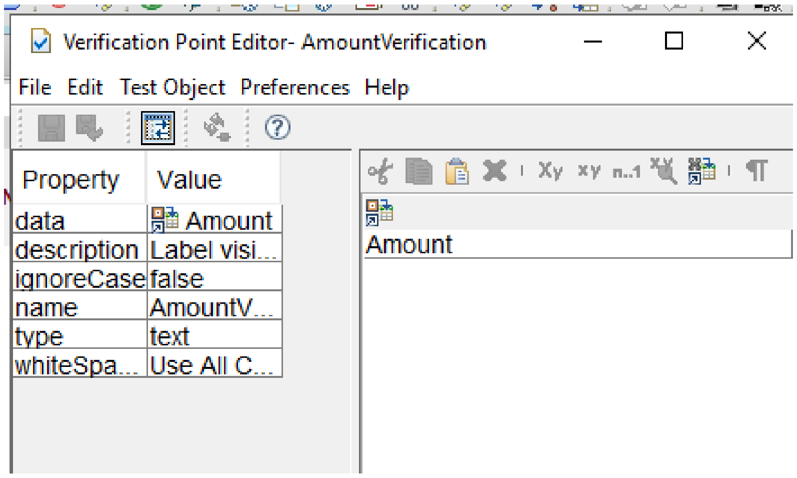


workspace - DataExamples/Script1.rftss - IBM Rational Functional Tester

```
Start Application ClassicsJavaA
ClassicsCD
    Click tree2 at Row at Composers->Haydn
    Click tree2 at Composers->Haydn->Symphonies Nos. 94 & 98
    Click Place Order
Member Logon
    Click OK
Place an Order
    Click Card Number (include the spaces)
    Type Value 7777 7777 7777 7777
    Click Expiration Date
    Type Value 10/{BKSP}{BKSP}0/2020
    Set Value of Item: from Datapool Column Composer
    Set Value of $17.95 from Datapool Column CD
    Set Value of Quantity from Datapool Column Quantity
    Set Value of Card Number (include the spaces) from Datapool Column CardNumber
    Set Value of creditCombo from Datapool Column CardType
    Set Value of Expiration Date from Datapool Column ExpirationDate
    Set Value of Name from Datapool Column Name
    Set Value of Street from Datapool Column Street
    Set Value of City, State, Zip from Datapool Column CityStateZip
    Set Value of Phone from Datapool Column Phone
    Verify Label visible text of $18.95
    Click Place Order
Message
    Click OK
ClassicsCD
    Close ClassicsCD
```

The "verify label text of $18.95" can be ignored since it was generated during the script before we converted the hard coded reference to a datapool reference.

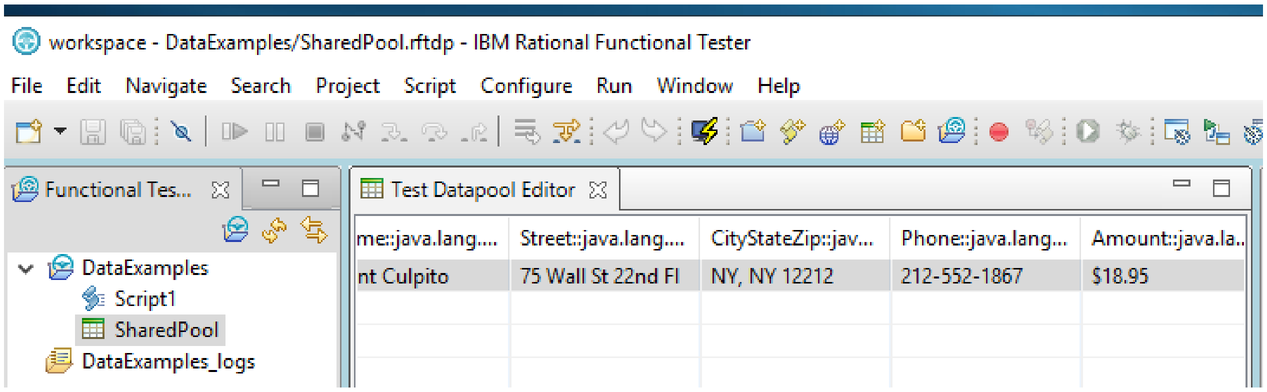
If we actually open up the verification point, we see that it does refer to that new variable in the datapool.


Verification Point Editor- AmountVerification
File Edit Test Object Preferences Help

Property	Value
data	Amount
description	Label visi...
ignoreCase	false
name	AmountV...
type	text
whiteSpa...	Use All C...

Amount

And we can see the variable in the datapool when we open it up from the project explorer.

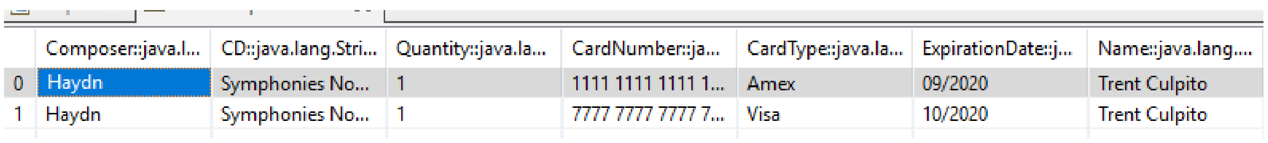

workspace - DataExamples/SharedPool.rftdp - IBM Rational Functional Tester
File Edit Navigate Search Project Script Configure Run Window Help
Functional Tes...
DataExamples
Script1
SharedPool
DataExamples_logs
Test Datapool Editor

me::java.lang....	Street::java.lang....	CityStateZip::jav...	Phone::java.lang...	Amount::java.la..
nt Culpito	75 Wall St 22nd Fl	NY, NY 12212	212-552-1867	$18.95

Running this script now performs exactly as our similar scripts have in the past, however we have been using a shared data pool to do so. We have decoupled the test data from the test script.

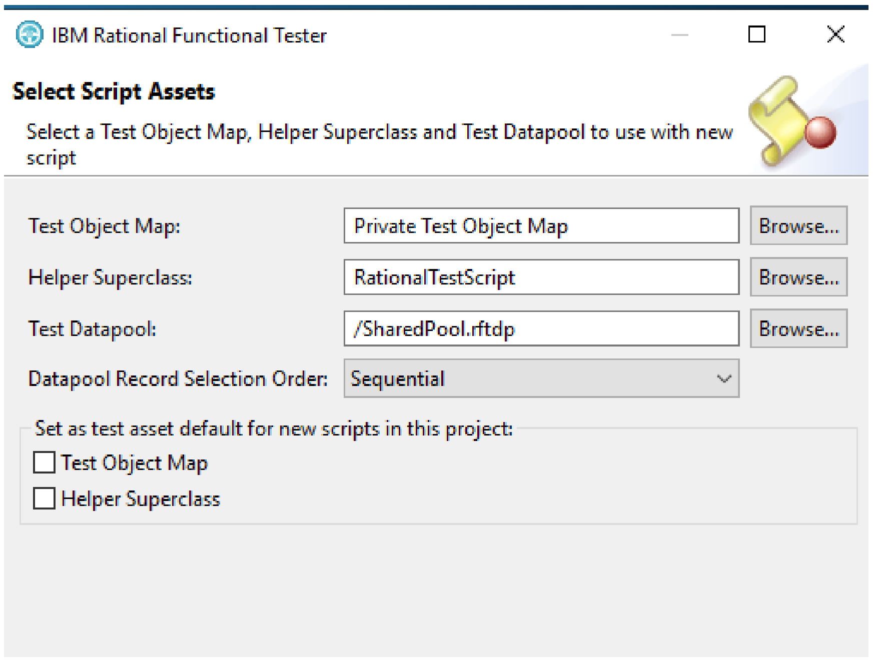
6.7 Adding a Datapool to a Script

To make things interesting, we are going to add another line to the datapool, which will just be a copy of the first line with a couple of minor changes.



	Composer::java.l...	CD::java.lang.Stri...	Quantity::java.la...	CardNumber::ja...	CardType::java.la...	ExpirationDate::j...	Name::java.lang....
0	Haydn	Symphonies No...	1	1111 1111 1111 1...	Amex	09/2020	Trent Culpito
1	Haydn	Symphonies No...	1	7777 7777 7777 7...	Visa	10/2020	Trent Culpito

Now we record a new script which cancels the order instead of placing it. We associate the datapool we have created with the script, but since we will not be doing any data-driven commands during the recording, the datapool will be unaltered.



IBM Rational Functional Tester

Select Script Assets

Select a Test Object Map, Helper Superclass and Test Datapool to use with new script

Test Object Map: Private Test Object Map [Browse...]

Helper Superclass: RationalTestScript [Browse...]

Test Datapool: /SharedPool.rftdp [Browse...]

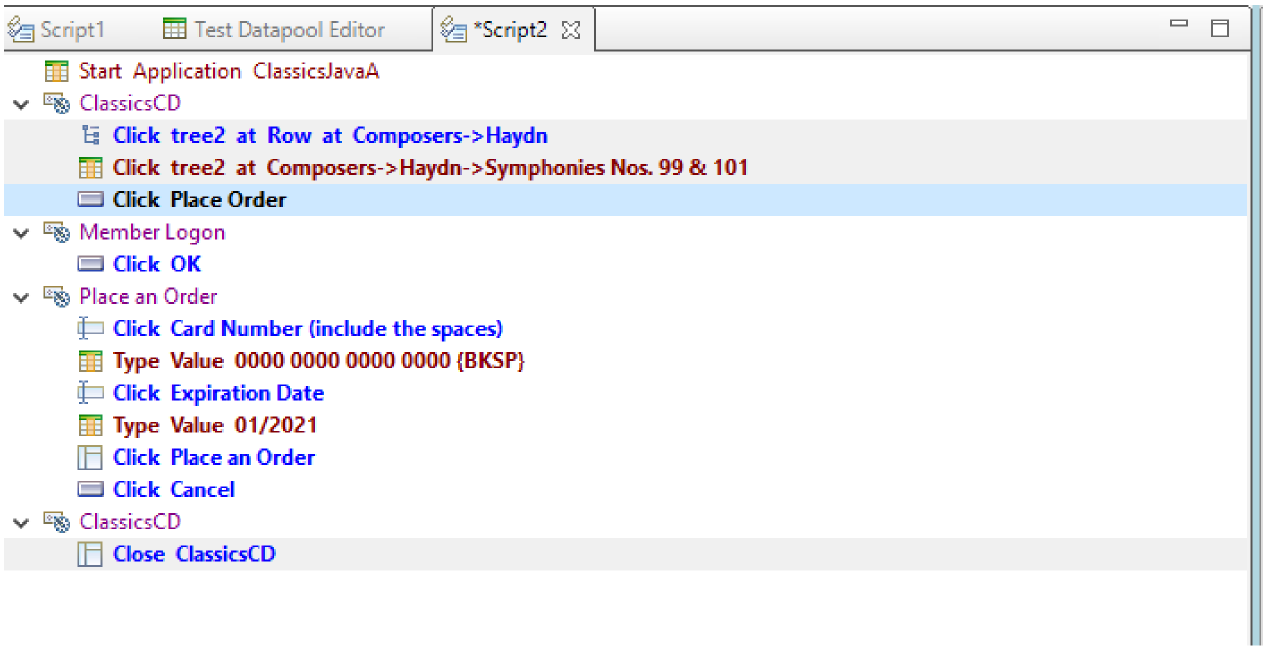
Datapool Record Selection Order: Sequential

Set as test asset default for new scripts in this project:

☐ Test Object Map

☐ Helper Superclass

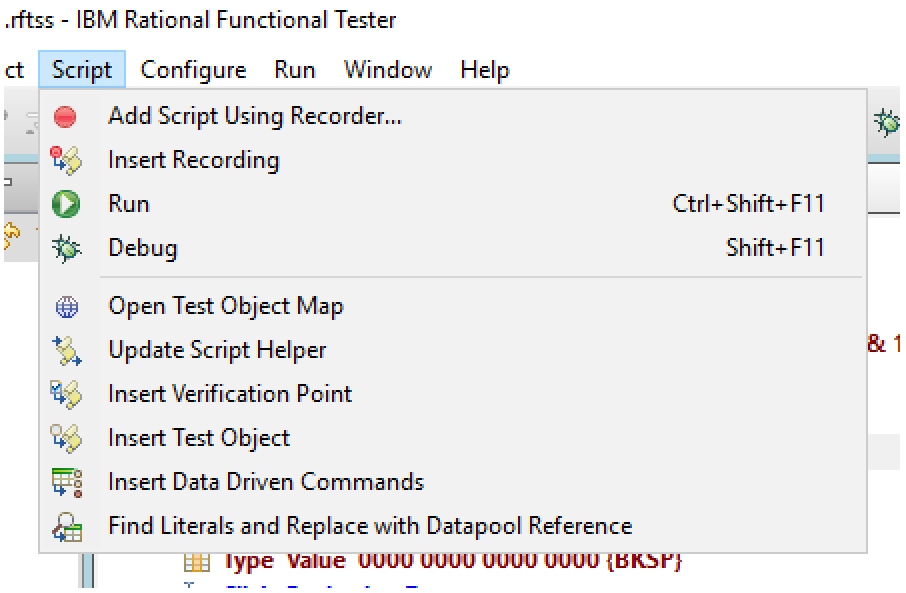
Recording the script produces the result seen on the next page. Notice that all of our values are hard coded and the datapool is not referenced.



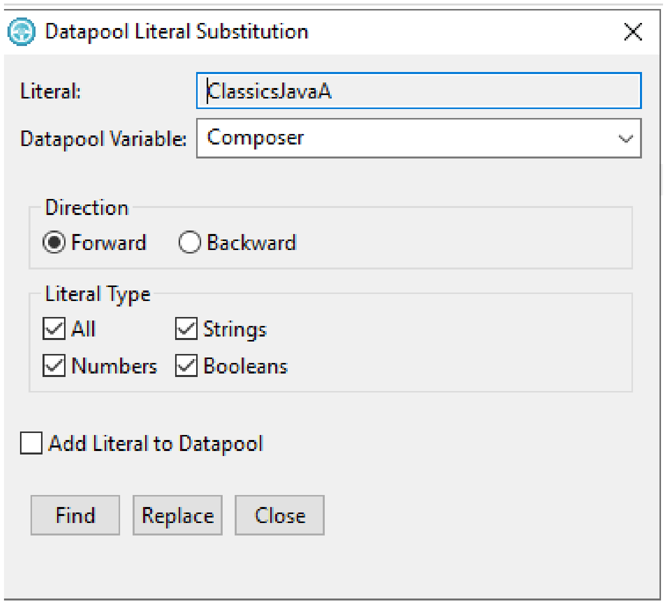
Start Application ClassicsJavaA
ClassicsCD
Click tree2 at Row at Composers->Haydn
Click tree2 at Composers->Haydn->Symphonies Nos. 99 & 101
Click Place Order
Member Logon
Click OK
Place an Order
Click Card Number (include the spaces)
Type Value 0000 0000 0000 0000 {BKSP}
Click Expiration Date
Type Value 01/2021
Click Place an Order
Click Cancel
ClassicsCD
Close ClassicsCD

Now we replace the literals with the corresponding datapool references. However it is important to remember that a script may not have to use ALL of the variables in the data-pool, such as in this case the "Amount" variable.

To do this, we select the replace literals by datapool references option from the script menu:


.rftss - IBM Rational Functional Tester
ct Script Configure Run Window Help
Add Script Using Recorder...
Insert Recording
Run Ctrl+Shift+F11
Debug Shift+F11
Open Test Object Map
Update Script Helper
Insert Verification Point
Insert Test Object
Insert Data Driven Commands
Find Literals and Replace with Datapool Reference

Since RFT has no idea what variables we do have in the datapool, it just runs through all the literals it can find. In this case below, we do not want to replace the name of the application so we just hit "Find" to move on to the next literal.


Datapool Literal Substitution

Literal: ClassicsJavaA

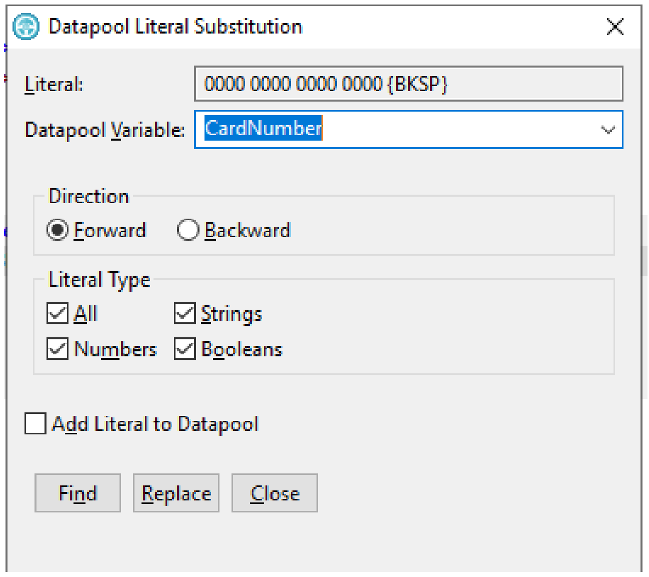
Datapool Variable: Composer

Direction: [✓ Forward] Backward

Literal Type: [✓ All] [✓ Strings] [✓ Numbers] [✓ Booleans]

[ ] Add Literal to Datapool

When we find a literal we want to replace, we can use the "Replace" option.


Datapool Literal Substitution

Literal: 0000 0000 0000 0000 {BKSP}

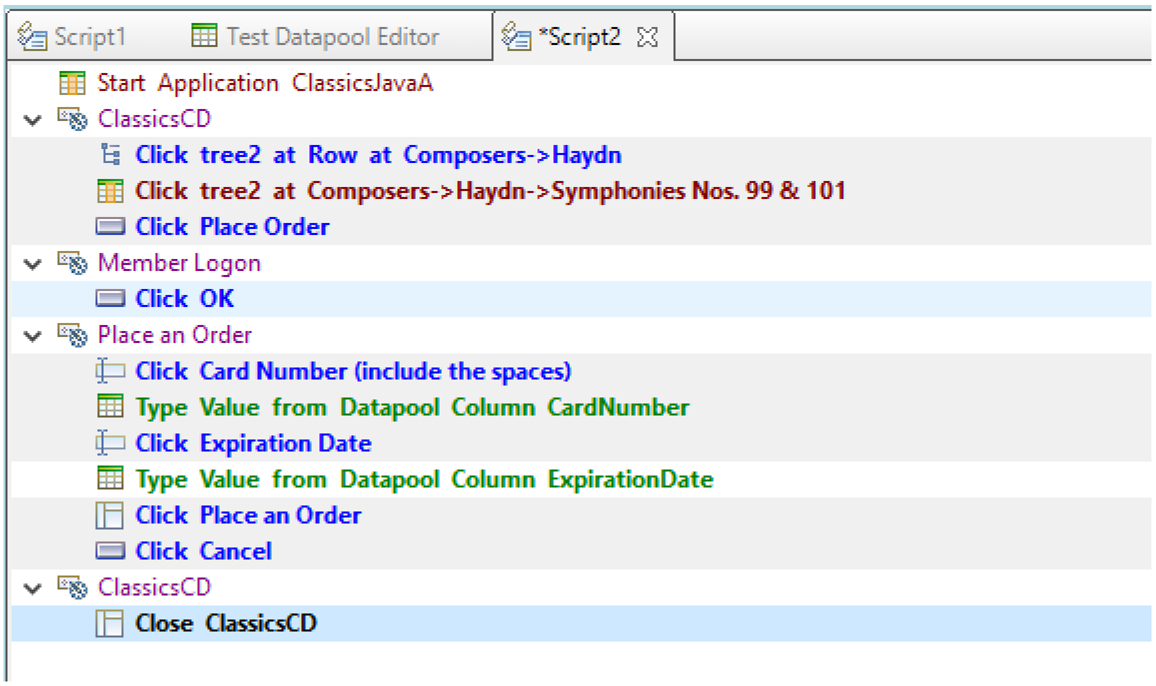
Datapool Variable: CardNumber

Direction: [✓ Forward] Backward

Literal Type: [✓ All] [✓ Strings] [✓ Numbers] [✓ Booleans]

[ ] Add Literal to Datapool

Replacing the rest of the literals we want to replace gives us the appropriate links to the datapool.



```
Start Application ClassicsJavaA
ClassicsCD
    Click tree2 at Row at Composers->Haydn
    Click tree2 at Composers->Haydn->Symphonies Nos. 99 & 101
    Click Place Order
Member Logon
    Click OK
Place an Order
    Click Card Number (include the spaces)
    Type Value from Datapool Column CardNumber
    Click Expiration Date
    Type Value from Datapool Column ExpirationDate
    Click Place an Order
    Click Cancel
ClassicsCD
    Close ClassicsCD
```

Now we can run the script using the shared datapool.

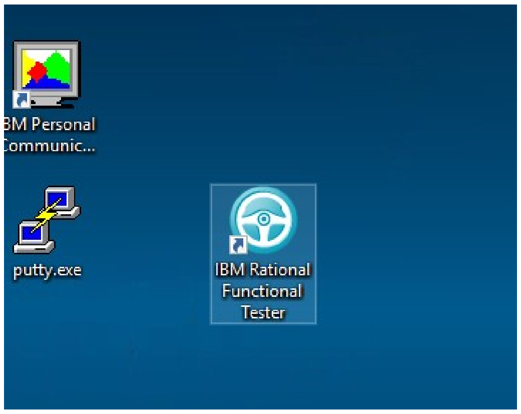
Labs and Exercises

Lab One

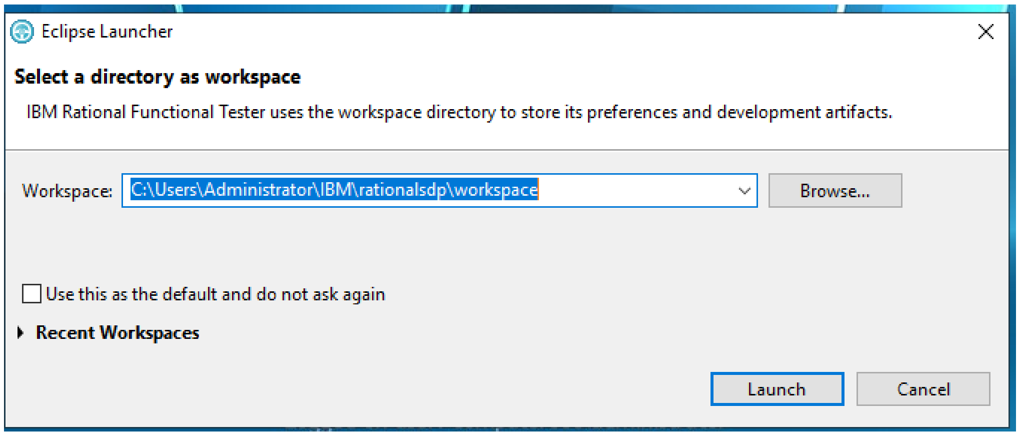
Set up IBM Rational Functional Tester

This lab focuses on the setting up an RFT project. It is highly recommended that you also take some time to explore the RFT environment on your own.

1. Log onto your laptop and open the RFT application from the desktop icon.



2. When the Eclipse screen opens, you can either use the default workspace or create your own.

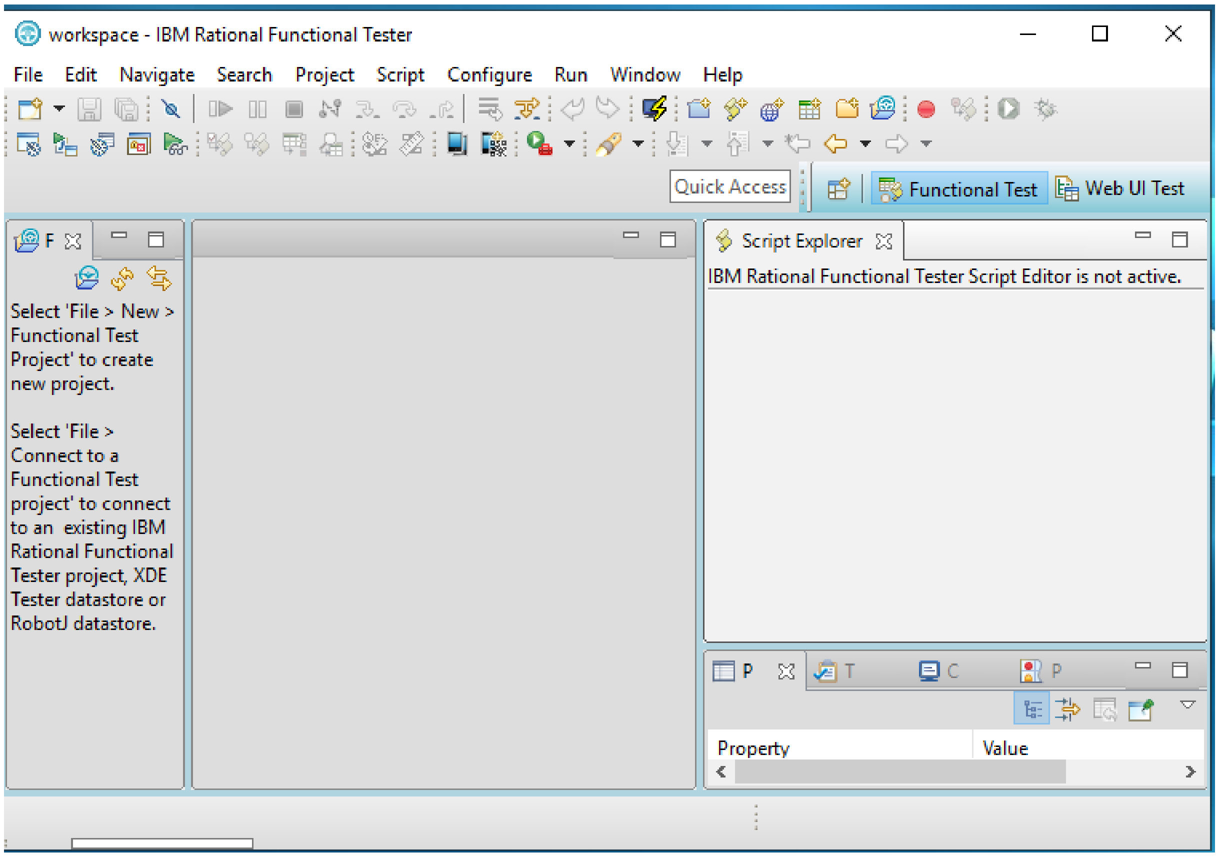

Eclipse Launcher

Select a directory as workspace

IBM Rational Functional Tester uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\Users\Administrator\IBM\rationalsdp\workspace

3. Since RFT has been run previously, when the workspace opens, you should see something like this.


workspace - IBM Rational Functional Tester
File Edit Navigate Search Project Script Configure Run Window Help
Quick Access
Functional Test
Web UI Test
F
Script Explorer
P
T
C
P
Property
Value

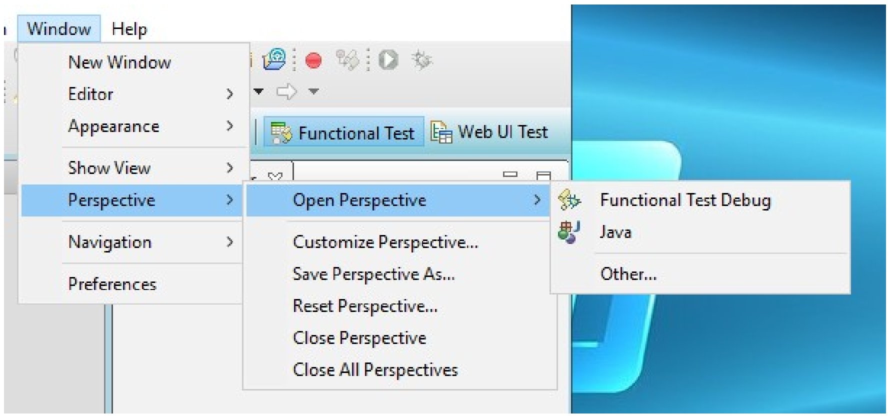
Select 'File > New > Functional Test Project' to create new project.

Select 'File > Connect to a Functional Test project' to connect to an existing IBM Rational Functional Tester project, XDE Tester datastore or RobotJ datastore.

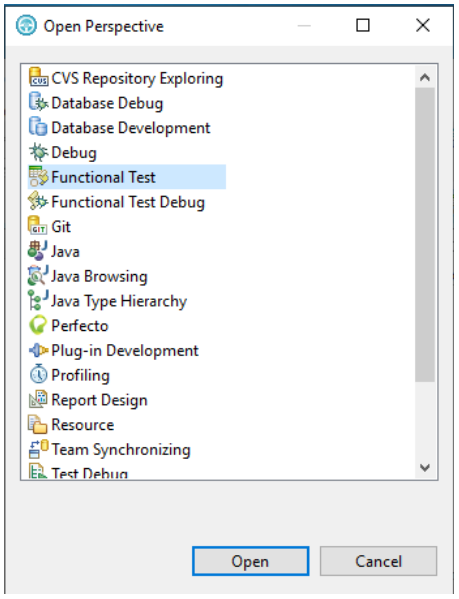
IBM Rational Functional Tester Script Editor is not active.

4. Note that you are in the Functional Test perspective that is shown just above the Script Explorer Window.

5. If you are not in the Functional Test perspective, you can switch to it using the Window menu.



6. The Functional Test perspective is under the "Other Option"



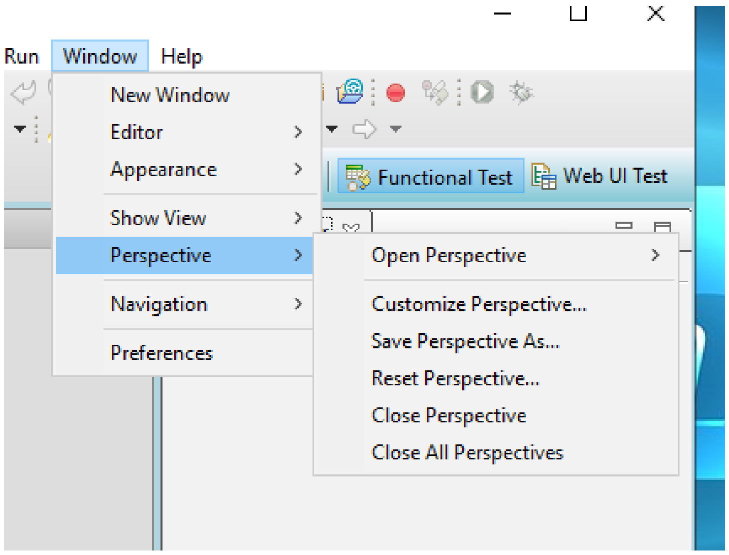
Open Perspective

- CVS Repository Exploring
- Database Debug
- Database Development
- Debug
- Functional Test
- Functional Test Debug
- Git
- Java
- Java Browsing
- Java Type Hierarchy
- Perfecto
- Plug-in Development
- Profiling
- Report Design
- Resource
- Team Synchronizing
- Test Debug

7. Recall that the RFT scripts are always stored as special kinds of Java projects. This means that we need to open an RFT project to record a script.

8. If you want to experiment, you might try switching to other perspectives and then back to the RFT perspective.

9. Hint: If you mess up your workspace by dragging things around or deleting various panels, you can always reset your workspace to the way it looked when you opened it up by choosing the "Reset Perspective" Option in the Windows Menu

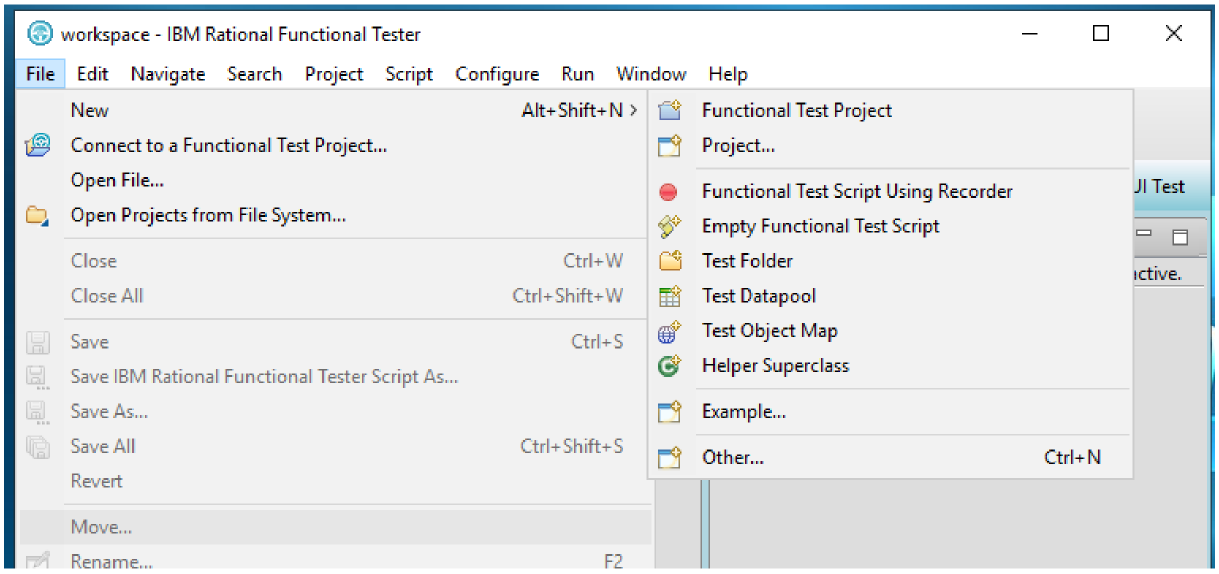

Run
Window
Help
New Window
Editor
Appearance
Show View
Perspective
Navigation
Preferences
Open Perspective
Customize Perspective...
Save Perspective As...
Reset Perspective...
Close Perspective
Close All Perspectives
Functional Test
Web UI Test

Lab Two

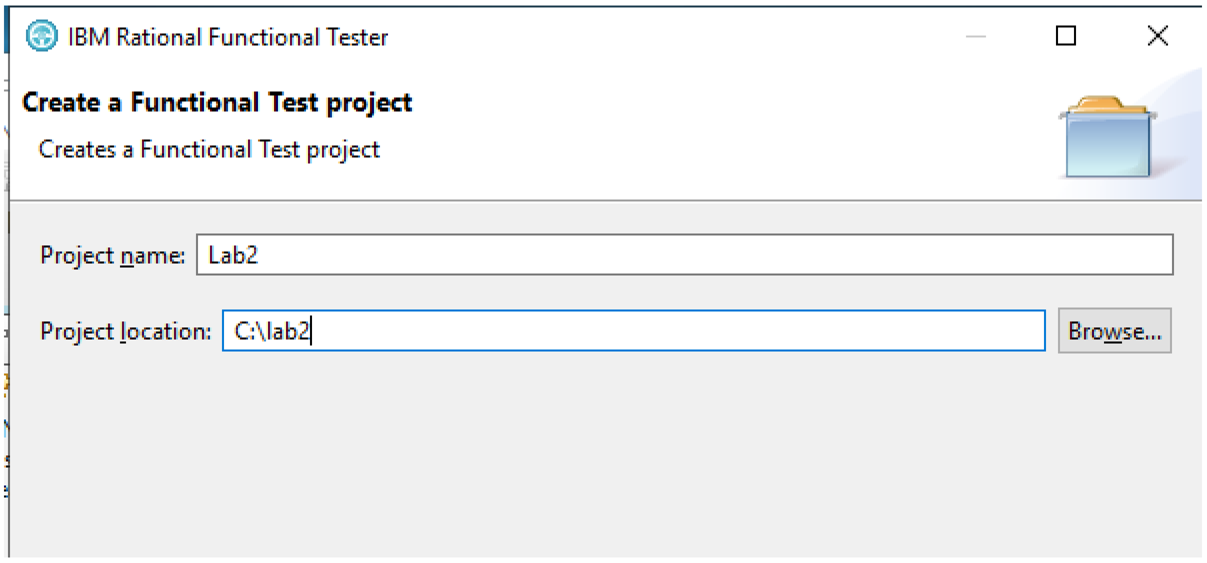
Setting up a Test Project

Part One: Creating a Project

1. Select RFT project from the "New" menu and give it a name.


workspace - IBM Rational Functional Tester
File Edit Navigate Search Project Script Configure Run Window Help
New Alt+Shift+N
Connect to a Functional Test Project...
Open File...
Open Projects from File System...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save IBM Rational Functional Tester Script As...
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Functional Test Project
Project...
Functional Test Script Using Recorder
Empty Functional Test Script
Test Folder
Test Datapool
Test Object Map
Helper Superclass
Example...
Other... Ctrl+N

2. Select a name and location. This creates a project workspace. Specify a location for the project as shown below.



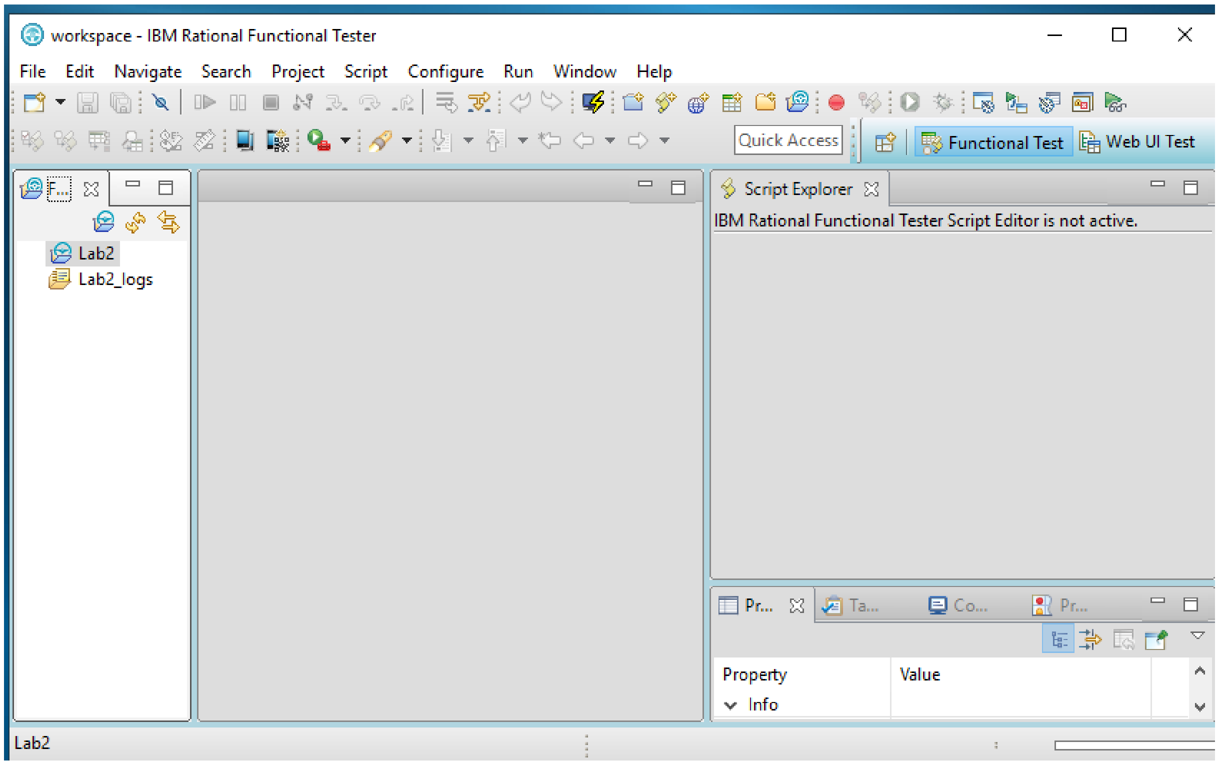
Create a Functional Test project

Creates a Functional Test project

Project name: Lab2

Project location: C:\lab2

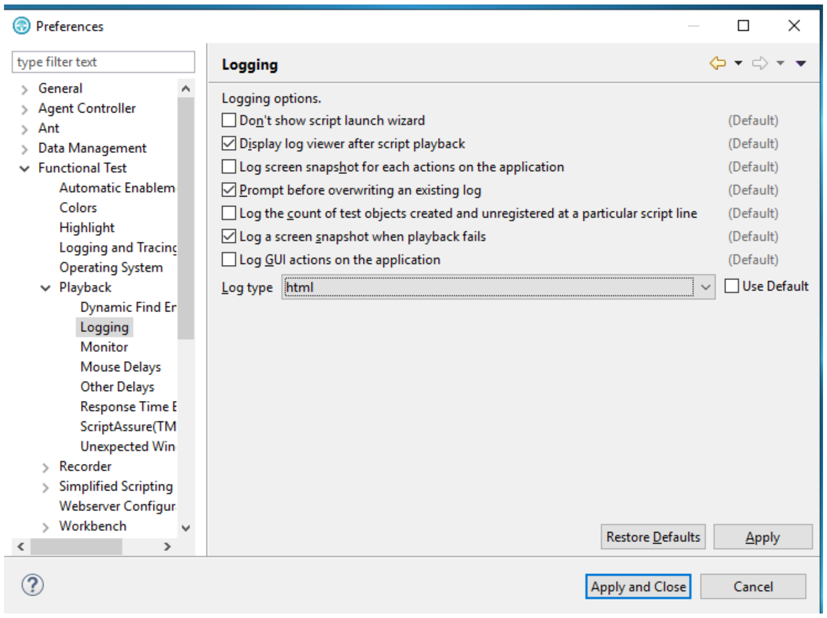
3. The resulting Eclipse workspace should look like this:



IBM Rational Functional Tester Script Editor is not active.

Part Two: Enabling HTML Logging

1. We will be using HTML logging for this class, so we have to reset it from the default XML logging. This can be done from the "Windows -> Preferences" menu as shown below. You will have to uncheck the "use default" box in order to select HTML logging.



Preferences

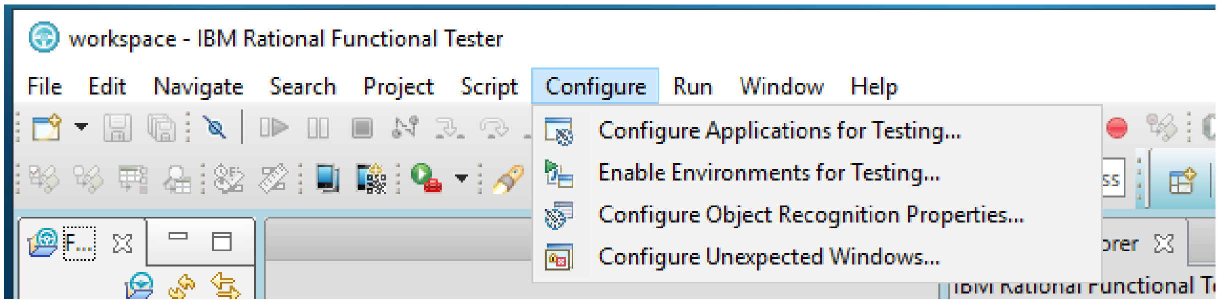
Logging

Logging options.

- [ ] Don't show script launch wizard (Default)
- [x] Display log viewer after script playback (Default)
- [ ] Log screen snapshot for each actions on the application (Default)
- [x] Prompt before overwriting an existing log (Default)
- [ ] Log the count of test objects created and unregistered at a particular script line (Default)
- [x] Log a screen snapshot when playback fails (Default)
- [ ] Log GUI actions on the application (Default)

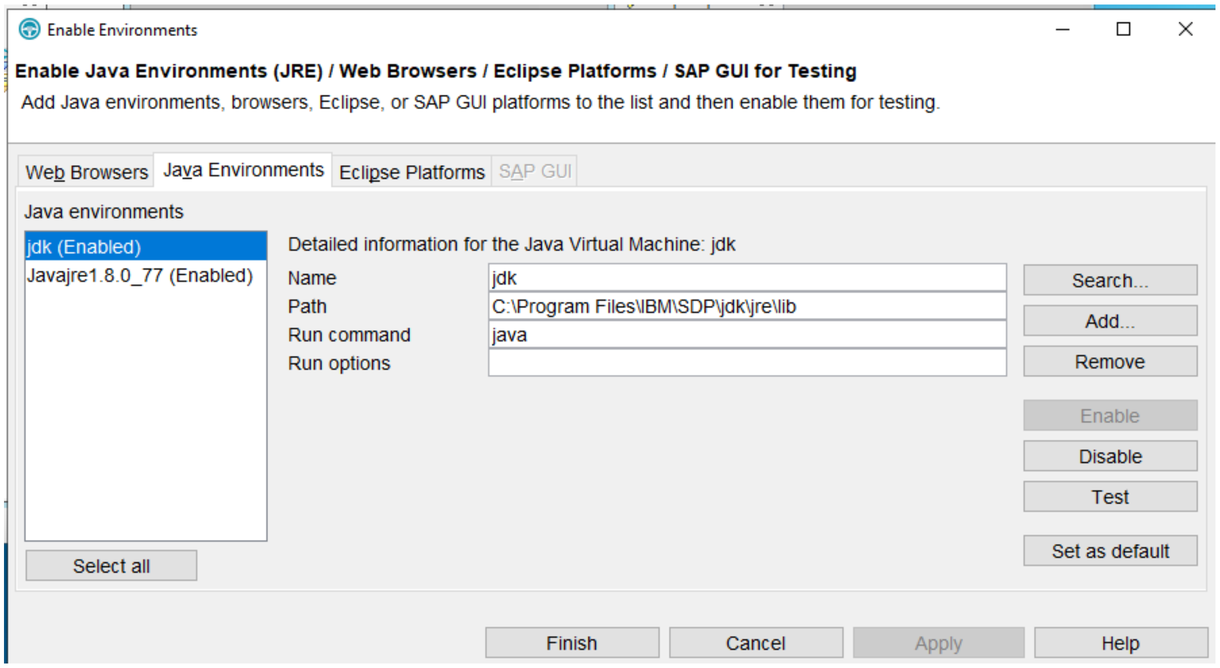
Log type: html — [ ] Use Default

2. We also have to do some configuration for our test environment with the Configuration menu. First, we have to enable environments for testing. Some of these will be enabled by default, but if you are having problems running RFT, you should check to see the correct environments are enabled.



workspace - IBM Rational Functional Tester
File
Edit
Navigate
Search
Project
Script
Configure
Run
Window
Help
Configure Applications for Testing...
Enable Environments for Testing...
Configure Object Recognition Properties...
Configure Unexpected Windows...

3. First, we have ensure the Java environment is enabled. It is enabled in the following screen shot.



Enable Environments

Enable Java Environments (JRE) / Web Browsers / Eclipse Platforms / SAP GUI for Testing

Add Java environments, browsers, Eclipse, or SAP GUI platforms to the list and then enable them for testing.

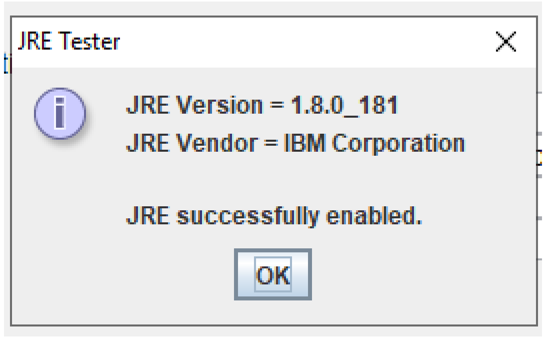
Java environments

- jdk (Enabled)
- Javajre1.8.0_77 (Enabled)

Detailed information for the Java Virtual Machine: jdk

Name	jdk
Path	C:\Program Files\IBM\SDP\jdk\jre\lib
Run command	java
Run options	

4. We can test that Java environment has been configured correctly by pressing the test button. We should see the following to confirm the environment is working.

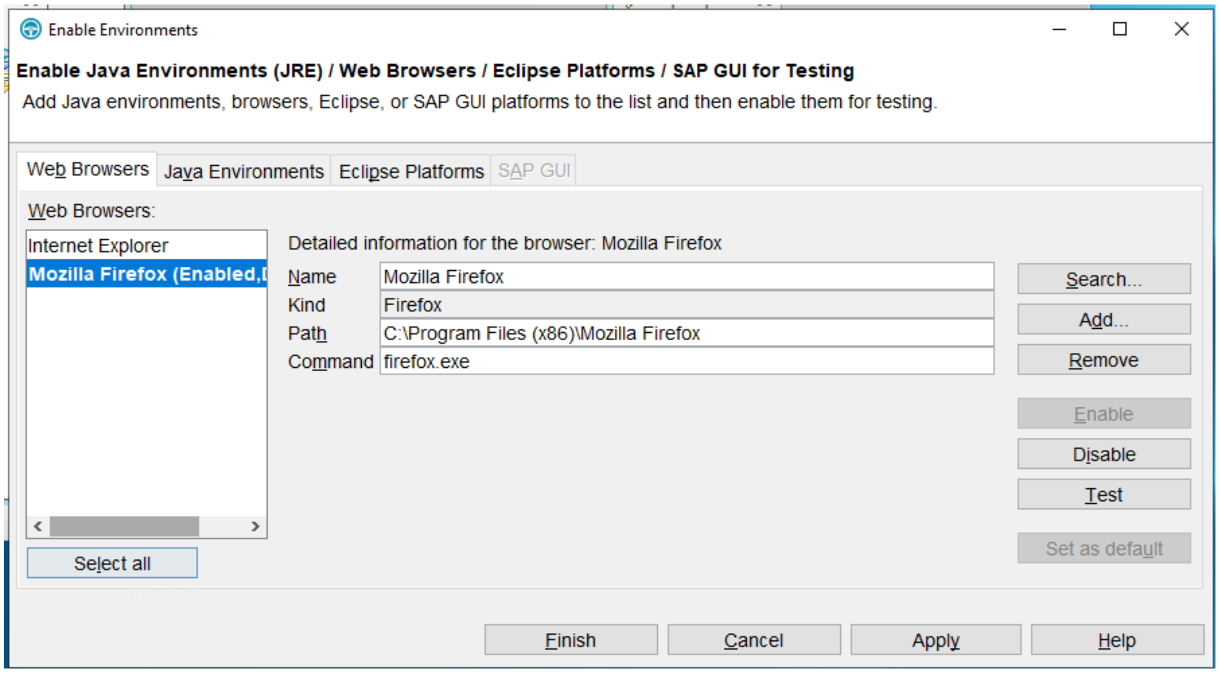


JRE Tester

JRE Version = 1.8.0_181
JRE Vendor = IBM Corporation

JRE successfully enabled.

5. While we are not using browsers in this class, if we are doing HTML application testing, we would enable the appropriate browsers here. The Windows IE browser is the default. In the following screenshot, the FireFox browser has been enabled and is the default browser.


Enable Environments

Enable Java Environments (JRE) / Web Browsers / Eclipse Platforms / SAP GUI for Testing

Add Java environments, browsers, Eclipse, or SAP GUI platforms to the list and then enable them for testing.

Web Browsers | Java Environments | Eclipse Platforms | SAP GUI

Web Browsers:

- Internet Explorer
- Mozilla Firefox (Enabled,I

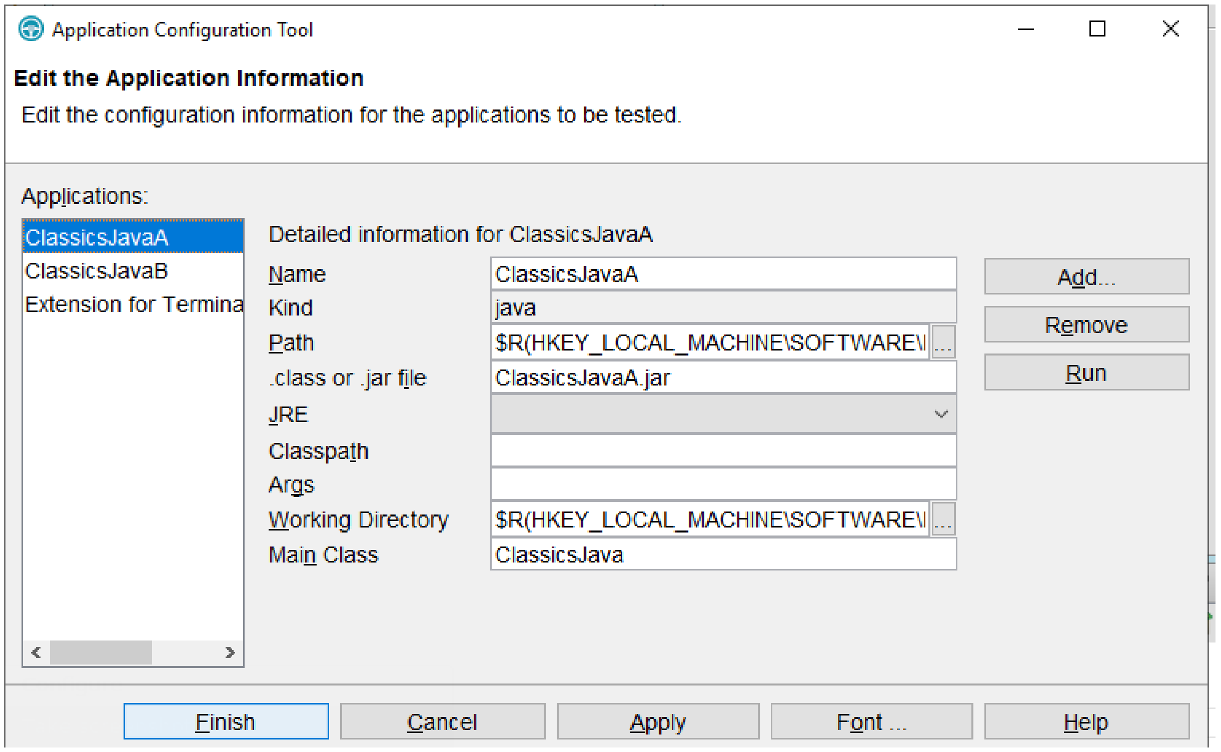
Detailed information for the browser: Mozilla Firefox

Name	Mozilla Firefox
Kind	Firefox
Path	C:\Program Files (x86)\Mozilla Firefox
Command	firefox.exe

6. There can be a problem testing applications that themselves use the Eclipse interface – so to avoid getting the application interface confused with the RTF interface, we can enable specific Eclipse applications as well. Since we do not have any Eclipse applications installed, this tab is empty. We also cannot configure the SAP GUI applications because we do not have the appropriate components installed.

Part Three: Configuring Applications

1. We cannot test applications that RTF can’t run. We can configure the applications in the Configuration menu:



Application Configuration Tool

Edit the Application Information

Edit the configuration information for the applications to be tested.

Applications:

- ClassicsJavaA
- ClassicsJavaB
- Extension for Termina

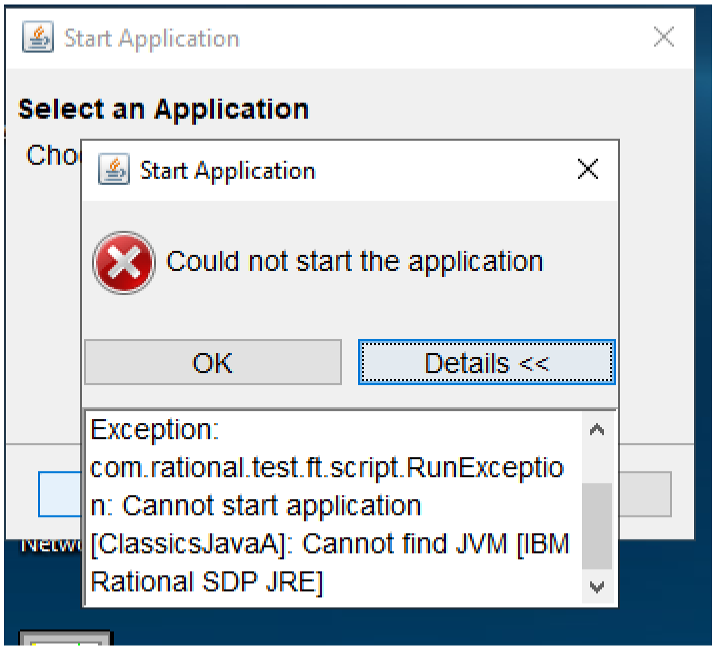
Detailed information for ClassicsJavaA

Name	ClassicsJavaA
Kind	java
Path	$R(HKEY_LOCAL_MACHINE\SOFTWARE\
.class or .jar file	ClassicsJavaA.jar
JRE	
Classpath	
Args	
Working Directory	$R(HKEY_LOCAL_MACHINE\SOFTWARE\
Main Class	ClassicsJava

2. For this course, we will be using the two ClassicsJava applications that are installed with the tool. In the above screen shot, ClassicsJavaA will not run because we have not specified a Java Virtual Machine to use (note: configuring an application to use a specific JVM is not the same as enabling an environment.)

   We may want to test an application with different JVMs. We would enable the appropriate JVMs and there specify here which on should be used to run the application.

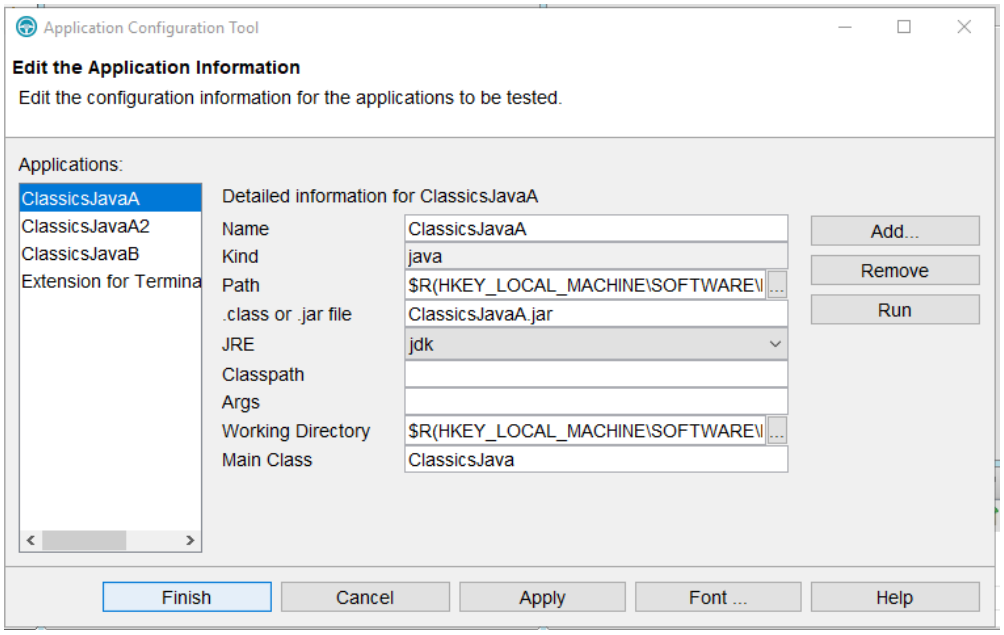
3. If we try to have RFT start JavaClassicsA, we get the following message:



Could not start the application

Exception: com.rational.test.ft.script.RunException: Cannot start application [ClassicsJavaA]: Cannot find JVM [IBM Rational SDP JRE]

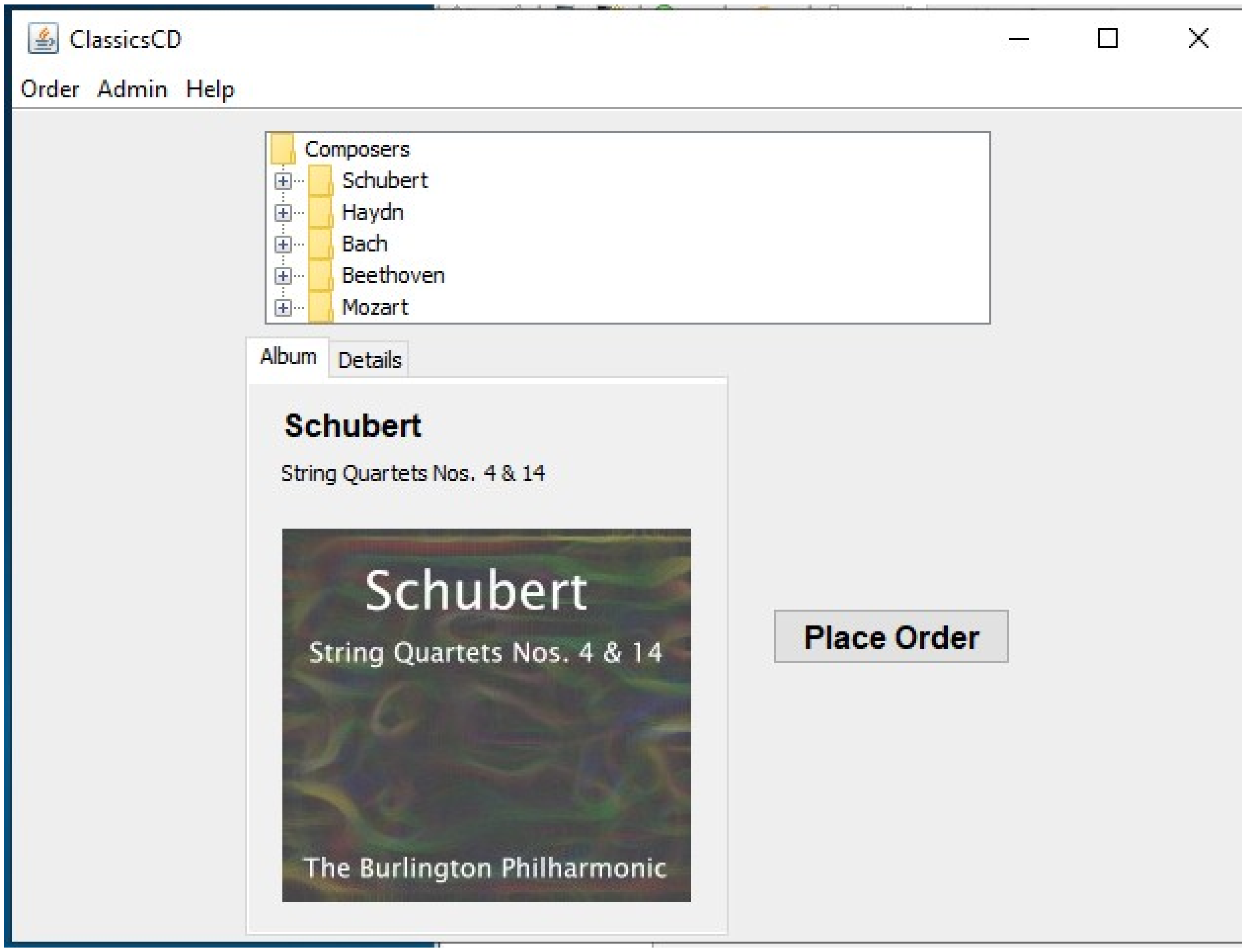
4. By specifying the jre as shown below, the problem is corrected and JavaClassicsA will be able to be started by RFT.


Application Configuration Tool
Edit the Application Information
Edit the configuration information for the applications to be tested.
Applications:
ClassicsJavaA
ClassicsJavaA2
ClassicsJavaB
Extension for Termina
Add...
Remove
Run
Finish
Cancel
Apply
Font ...
Help

Detailed information for ClassicsJavaA

Name	ClassicsJavaA
Kind	java
Path	$R(HKEY_LOCAL_MACHINE\SOFTWARE\
.class or .jar file	ClassicsJavaA.jar
JRE	jdk
Classpath	
Args	
Working Directory	$R(HKEY_LOCAL_MACHINE\SOFTWARE\
Main Class	ClassicsJava

5. Confirm that that ClassicsJavaA application is configured correctly by pressing the "Run" Button. You should see the following after an initial splash screen.



Lab Three

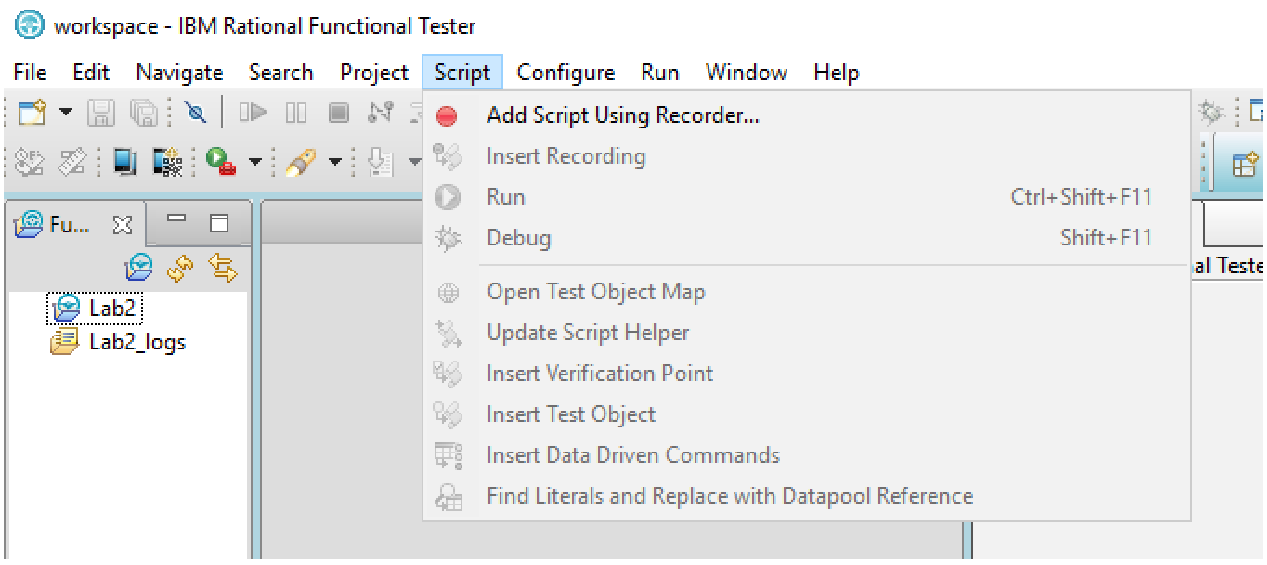
Recording and Playing Back a Script

In this lab, you will record a simple script and then play it back. At various points in the lab, you will also examine the state of the test artifacts.

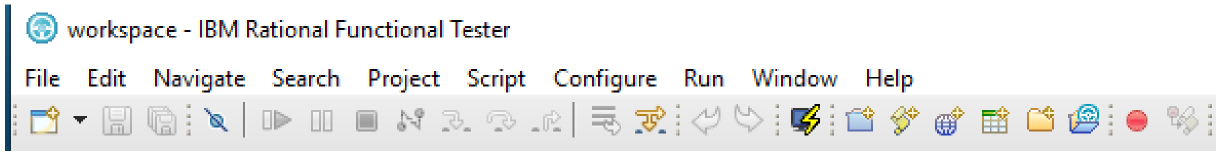
This lab uses the project "lab2" created in the previous lab.

Part One: Start the Script

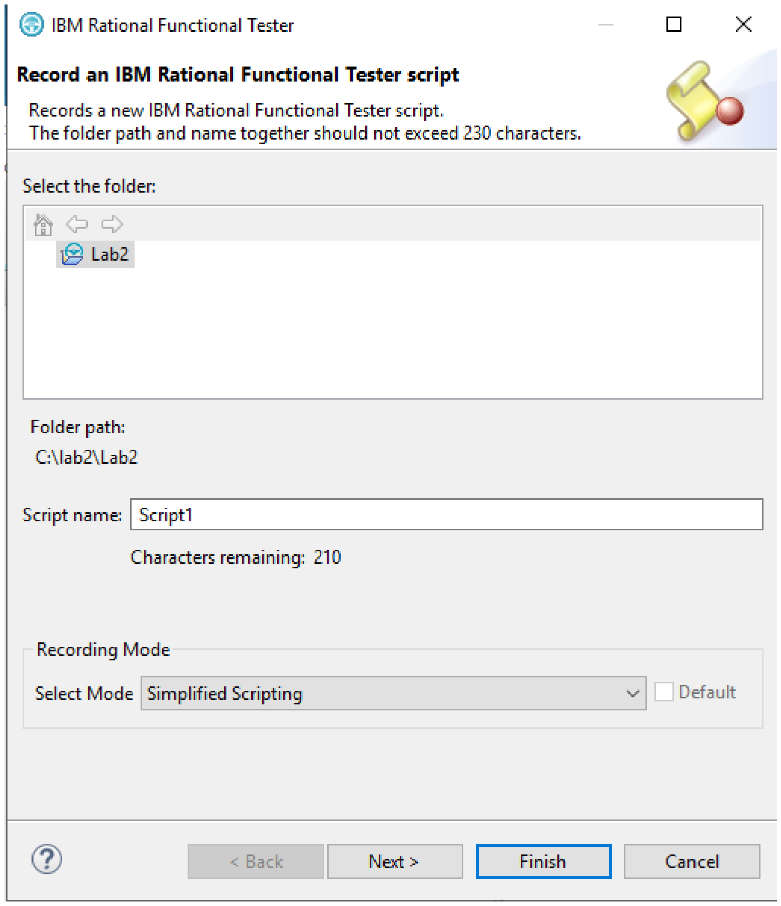
1. The first step is to start the scrip recorder by selecting the "Record" option either from the scrip menu


workspace - IBM Rational Functional Tester
File Edit Navigate Search Project Script Configure Run Window Help
Fu...
Lab2
Lab2_logs
Add Script Using Recorder...
Insert Recording
Run Ctrl+Shift+F11
Debug Shift+F11
Open Test Object Map
Update Script Helper
Insert Verification Point
Insert Test Object
Insert Data Driven Commands
Find Literals and Replace with Datapool Reference

or it can be started directly from the red circle on the task bar.


workspace - IBM Rational Functional Tester
File Edit Navigate Search Project Script Configure Run Window Help

2. Starting the recorder brings up the configuration dialogue box.



IBM Rational Functional Tester

Record an IBM Rational Functional Tester script

Records a new IBM Rational Functional Tester script.
The folder path and name together should not exceed 230 characters.

Select the folder:

Lab2

Folder path:
C:\lab2\Lab2

Script name: Script1

Characters remaining: 210

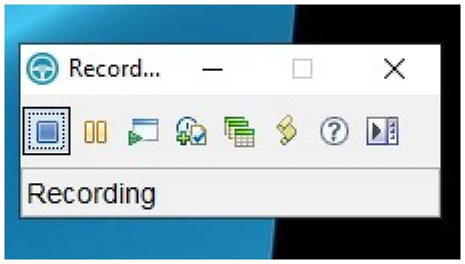
Recording Mode

Select Mode: Simplified Scripting

Default

< Back | Next > | Finish | Cancel

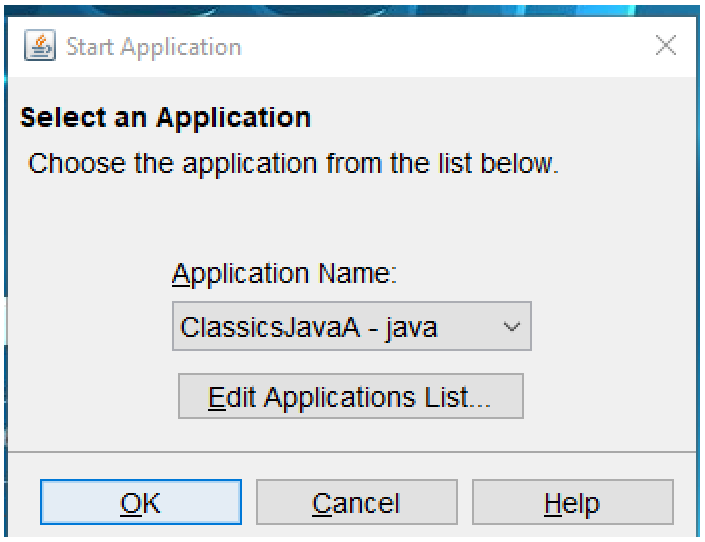
3. For this exercise, select "Finish". The "Next" button allows us to configure test assets, but we will just be using the defaults for this exercise so we don't need to go to the "Next" screen.
4. Once "Finish" is selected, look for the recorder box on the desktop



Record...

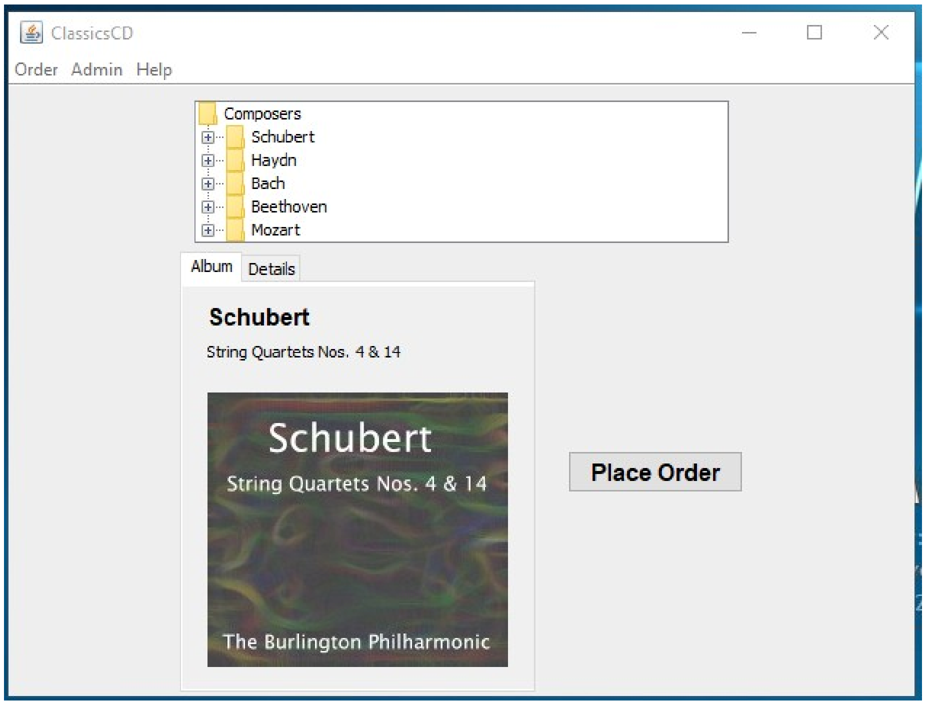
Recording

5. Select the third icon from the left (the little dialogue box with the green arrow) and click. This brings up the Application selection menu.

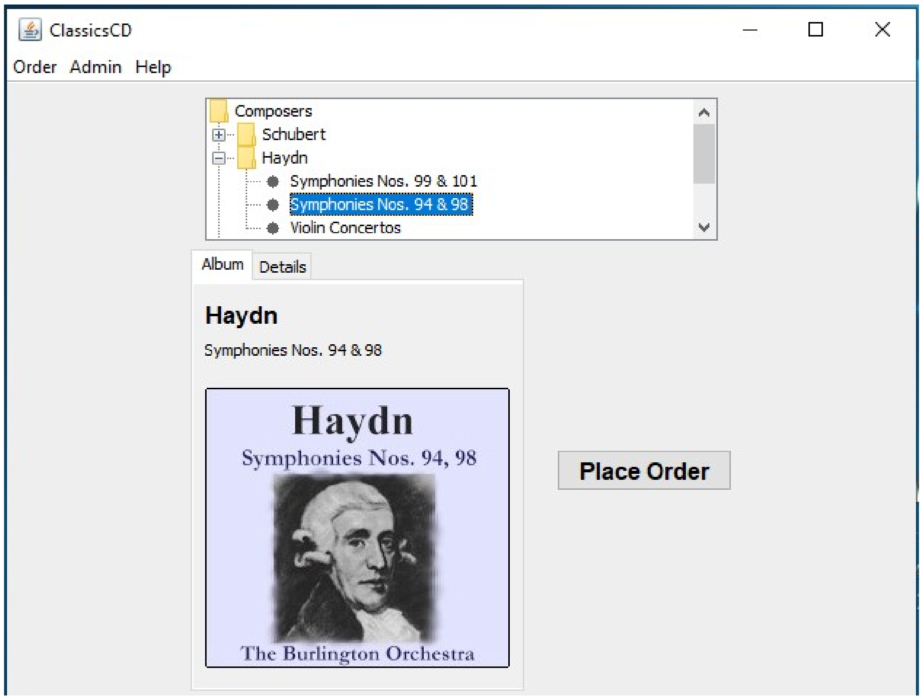


Part Two: Recording the Script

1. Since we are going to be using ClassicsJavaA for this script, select OK on the dialogue box above. After the splash screen, you should see the following screen.



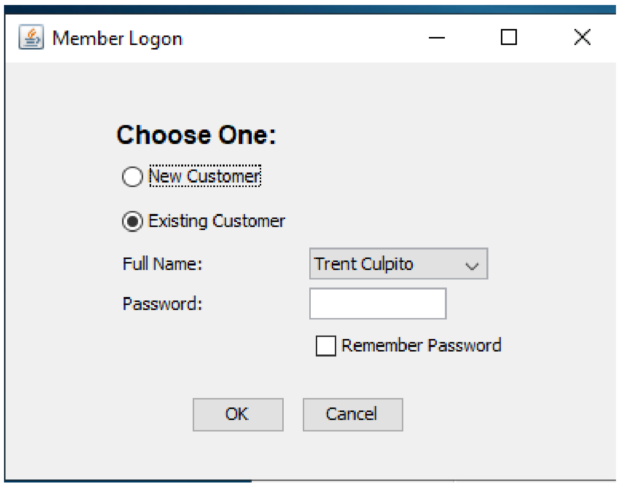
2. Click the + next to Haydn to expand the folder in the Composers tree.
3. In the list, click Symphonies Nos. 94 & 98.



Haydn

Symphonies Nos. 94 & 98

4. Click the Place Order button.
5. In the Member Logon window, keep the default settings of Existing Customer and Trent Culpito. Do not click either of the password fields at this time.



Choose One:

New Customer [✓ Existing Customer]

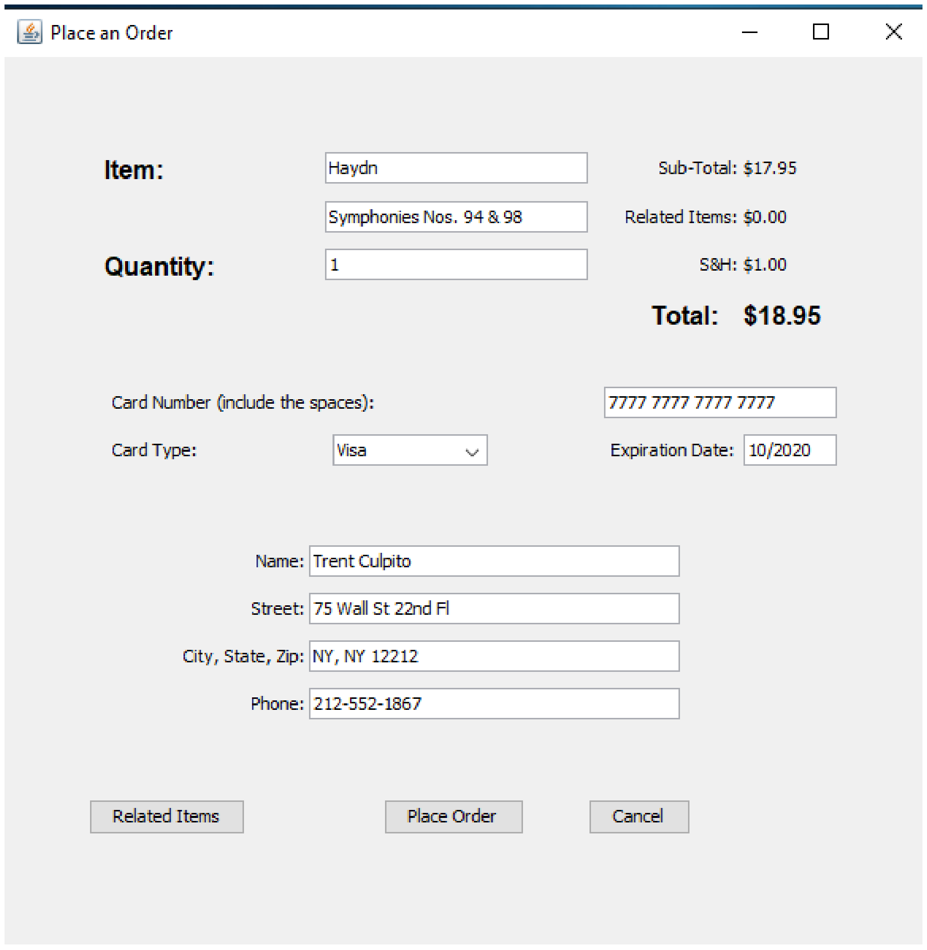
Full Name: Trent Culpito

Password:

☐ Remember Password

6. Click OK.

7. In the card number field, enter a credit card number. You must use the valid format of four sets of four digits here, for instance, `7777 7777 7777 7777`.
8. In the expiration date field, enter a valid format expiration date, `10/2012`.

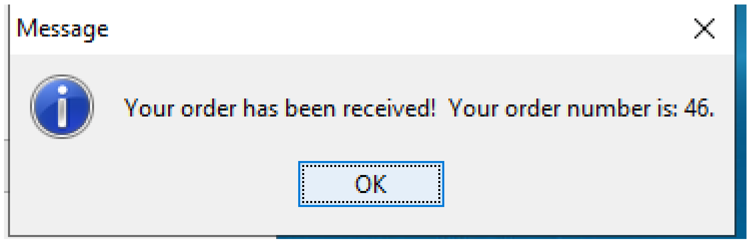

Place an Order

Item:	Haydn	Sub-Total: $17.95
	Symphonies Nos. 94 & 98	Related Items: $0.00
Quantity:	1	S&H: $1.00
		Total: $18.95

Card Number (include the spaces): 7777 7777 7777 7777
Card Type: Visa
Expiration Date: 10/2020

Name: Trent Culpito
Street: 75 Wall St 22nd Fl
City, State, Zip: NY, NY 12212
Phone: 212-552-1867

9. Click Place Order.
10. Click OK in the order confirmation message window.

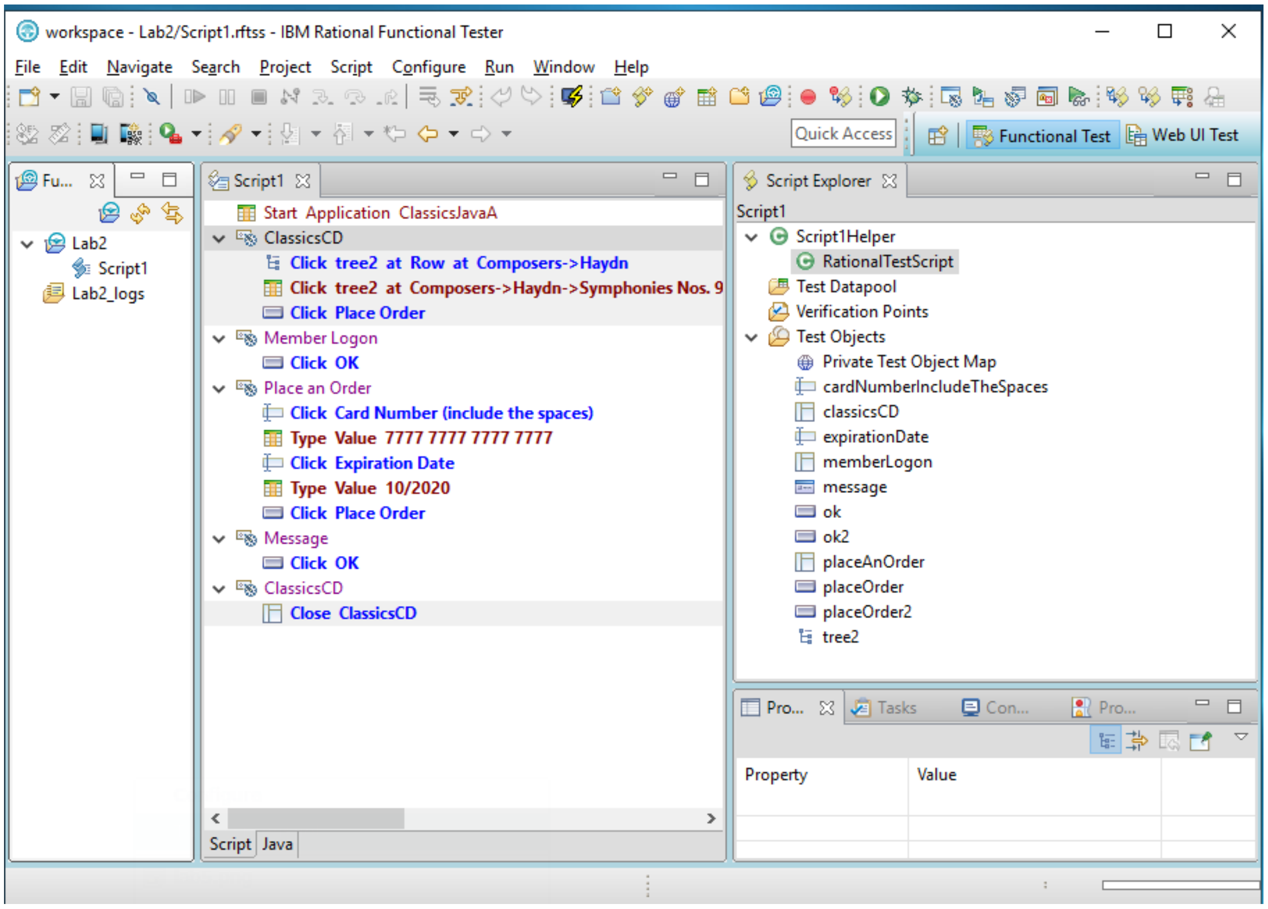

Message

Your order has been received! Your order number is: 46.

11. Close the Application and select "Stop Recording" from the Record Control dialogue box.

Part Three: Examining the Script and Test Assets

The finished script is displayed once the recording ends.

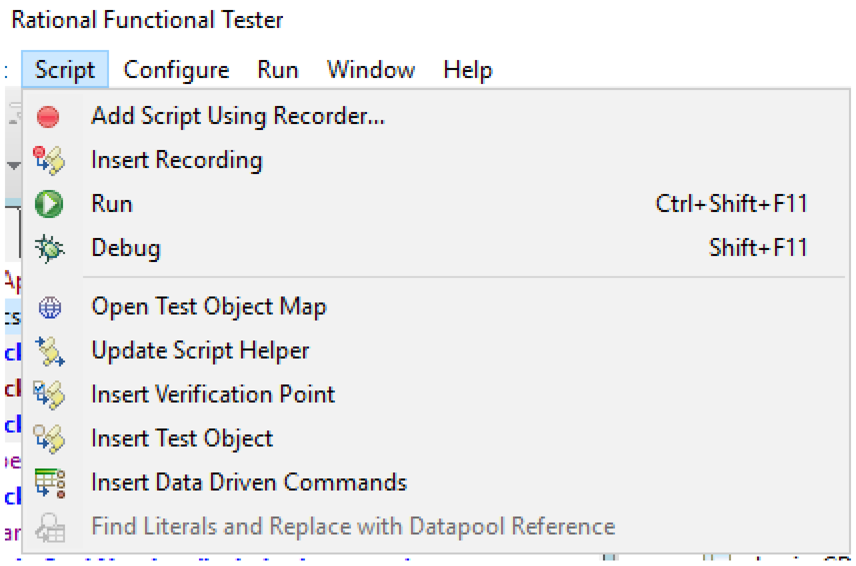


Start Application ClassicsJavaA
ClassicsCD
Click tree2 at Row at Composers->Haydn
Click tree2 at Composers->Haydn->Symphonies Nos. 9
Click Place Order
Member Logon
Click OK
Place an Order
Click Card Number (include the spaces)
Type Value 7777 7777 7777 7777
Click Expiration Date
Type Value 10/2020
Click Place Order
Message
Click OK
ClassicsCD
Close ClassicsCD

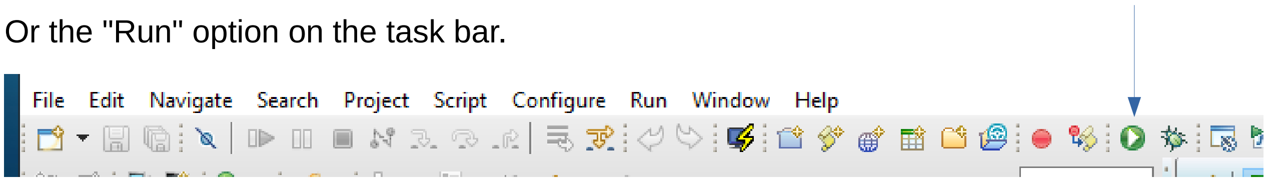
Script1
Script1Helper
RationalTestScript
Test Datapool
Verification Points
Test Objects
Private Test Object Map
cardNumberIncludeTheSpaces
classicsCD
expirationDate
memberLogon
message
ok
ok2
placeAnOrder
placeOrder
placeOrder2
tree2

Part Four: Playing the Script

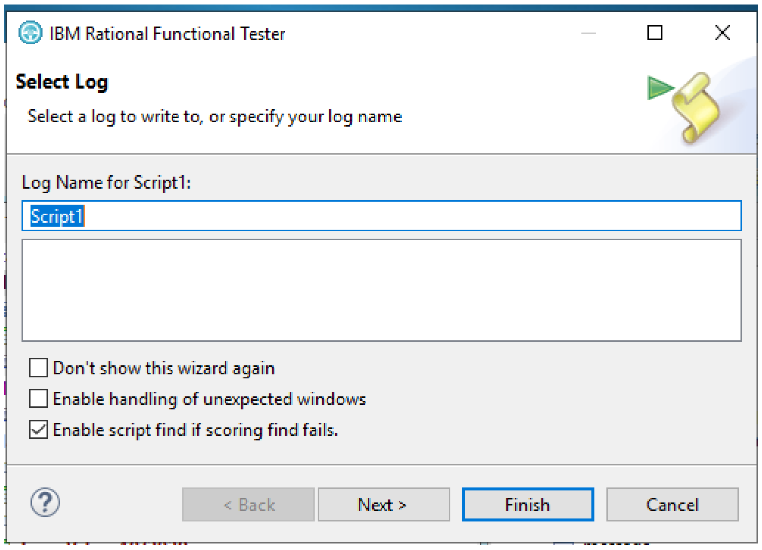
Now that the script has been recorded. It can be played back with either the "Run" option on the script menu:


Rational Functional Tester
Script
Configure
Run
Window
Help
Add Script Using Recorder...
Insert Recording
Run Ctrl+Shift+F11
Debug Shift+F11
Open Test Object Map
Update Script Helper
Insert Verification Point
Insert Test Object
Insert Data Driven Commands
Find Literals and Replace with Datapool Reference

Or the "Run" option on the task bar.



Choose either one of these and run the script. However, before the script executes, you will be prompted for a name for a log file.



IBM Rational Functional Tester

Select Log

Select a log to write to, or specify your log name

Log Name for Script1:

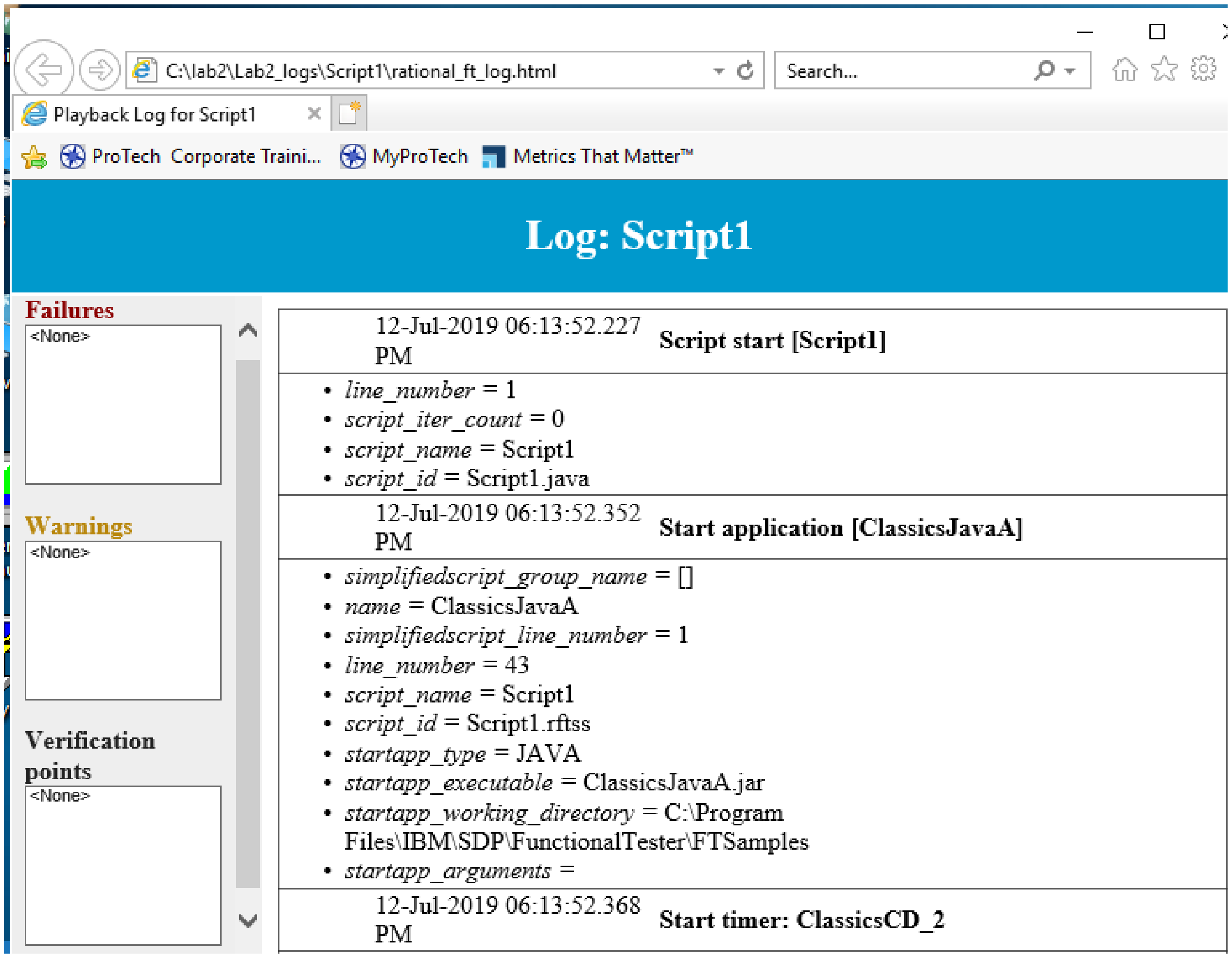
Script1

- [ ] Don't show this wizard again
- [ ] Enable handling of unexpected windows
- [x] Enable script find if scoring find fails.

For the purposes of this lab, use the default and click "Finish".

Notice that once the script runs, we now have a log file.

Once the script has executed, the HTML log file will open in a browser window.


C:\lab2\Lab2_logs\Script1\rational_ft_log.html
Search...
Playback Log for Script1
ProTech Corporate Traini...
MyProTech
Metrics That Matter™

Log: Script1

Failures
<None>

Warnings
<None>

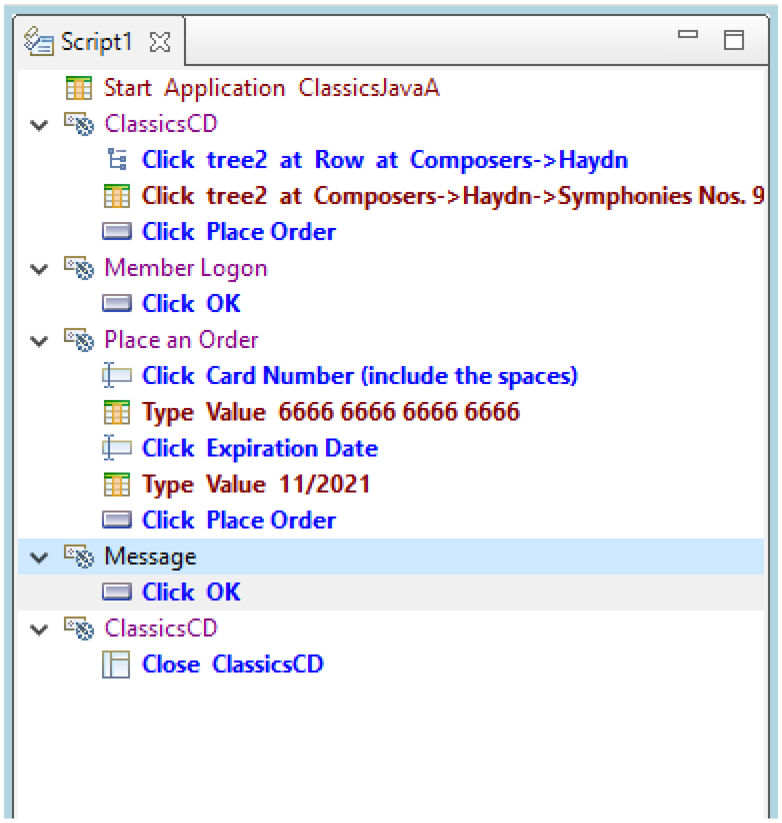
Verification points
<None>

```
12-Jul-2019 06:13:52.227 PM    Script start [Script1]
  • line_number = 1
  • script_iter_count = 0
  • script_name = Script1
  • script_id = Script1.java
12-Jul-2019 06:13:52.352 PM    Start application [ClassicsJavaA]
  • simplifiedscript_group_name = []
  • name = ClassicsJavaA
  • simplifiedscript_line_number = 1
  • line_number = 43
  • script_name = Script1
  • script_id = Script1.rftss
  • startapp_type = JAVA
  • startapp_executable = ClassicsJavaA.jar
  • startapp_working_directory = C:\Program
    Files\IBM\SDP\FunctionalTester\FTSamples
  • startapp_arguments =
12-Jul-2019 06:13:52.368 PM    Start timer: ClassicsCD_2
```

Part Five: Editing the Script

Once we have a script, we can edit it in various ways. However, it is possible to "break" the script if we don't edit it correctly.

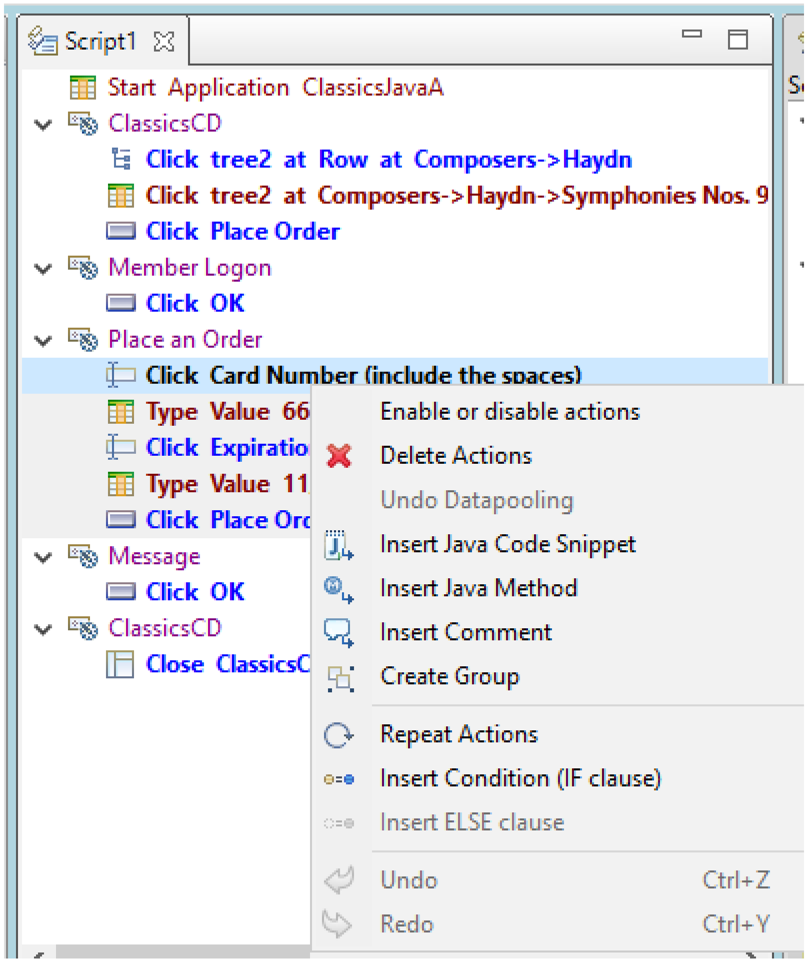
1. In the script window, edit the card number by replacing 7777 7777 7777 7777 with 6666 6666 6666 6666
2. Change the expiry date to 11/2021.
3. Rerun the script and confirm that it executes with the new values.


Script1

Start Application ClassicsJavaA
ClassicsCD
Click tree2 at Row at Composers->Haydn
Click tree2 at Composers->Haydn->Symphonies Nos. 9
Click Place Order
Member Logon
Click OK
Place an Order
Click Card Number (include the spaces)
Type Value 6666 6666 6666 6666
Click Expiration Date
Type Value 11/2021
Click Place Order
Message
Click OK
ClassicsCD
Close ClassicsCD

Part Six: Optional Activity

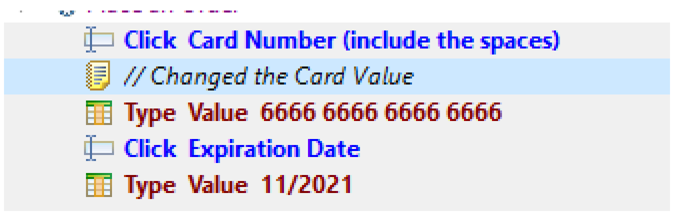
1. Add a comment to the script by selecting a line in the script and using the right mouse button to bring up the menu with the "Insert Comment" option.



Script1

Start Application ClassicsJavaA
ClassicsCD
Click tree2 at Row at Composers->Haydn
Click tree2 at Composers->Haydn->Symphonies Nos. 9
Click Place Order
Member Logon
Click OK
Place an Order
Click Card Number (include the spaces)
Type Value 66
Click Expiratio
Type Value 11
Click Place Orc
Message
Click OK
ClassicsCD
Close ClassicsC

2. Insert a comment.

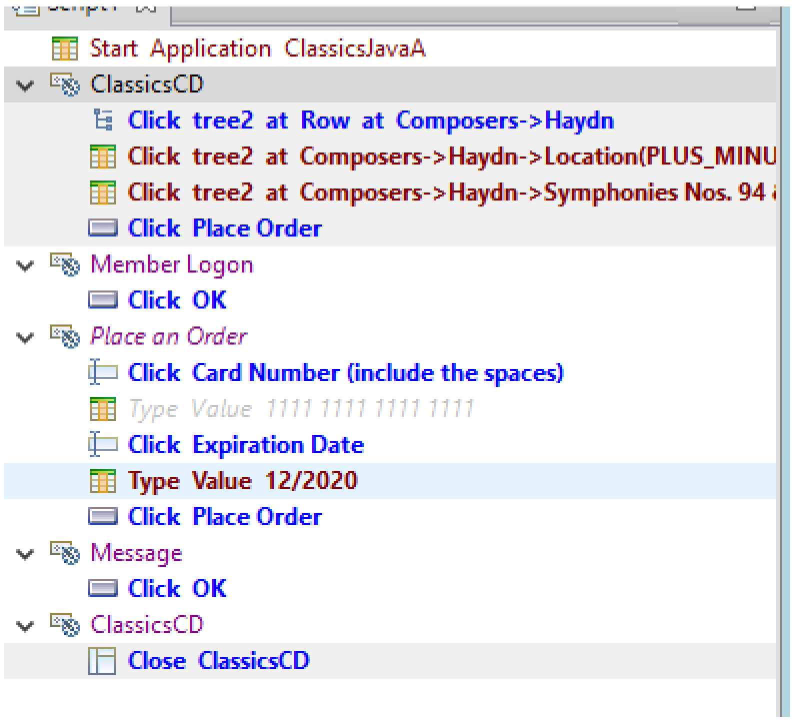


Click Card Number (include the spaces)
// Changed the Card Value
Type Value 6666 6666 6666 6666
Click Expiration Date
Type Value 11/2021

3. Delete the comment by selecting the line, and using the right mouse button to delete the script action.

Advanced Challenge 1: How would you edit the script to stop the application from closing at the end of the script? You may have to close the script window and then reopen the script to see the change. Check the Java code to see how the changes in the script affect the Java code. Once you have disabled this action, re-enable it.

Note: After you disable a script step, save the file. Close the editor window for the script and then re-open the script to ensure you see the disabled step. A disabled step looks like this where the typing of the credit card number is disabled.



```
Start Application ClassicsJavaA
ClassicsCD
    Click tree2 at Row at Composers->Haydn
    Click tree2 at Composers->Haydn->Location(PLUS_MINU
    Click tree2 at Composers->Haydn->Symphonies Nos. 94
    Click Place Order
Member Logon
    Click OK
Place an Order
    Click Card Number (include the spaces)
    Type Value 1111 1111 1111 1111
    Click Expiration Date
    Type Value 12/2020
    Click Place Order
Message
    Click OK
ClassicsCD
    Close ClassicsCD
```

Advanced Challenge 2: Delete a line from the script that causes the playback to fail. Close the application at that point and end the recording. Look at the log to see how the failure of the script to execute was recorded.

Lab Four

Verification Points

Now that you have had a chance to get the basics of writing scripts, the lab instructions will not be populated with screenshots except where needed for clarity. This is partly to encourage you to move around the interface in a more exploratory manner.

If you cannot work your way through any of the labs, as your instructor to demonstrate how the lab should be done.

Advanced: Although each part of the lab involves adding a verification point, it is more challenging to develop a single script that includes *all* of the verification points in the lab. If you want to try something a bit more challenging, read through all the the parts of the lab and then do them all at once in a single script.

Part One: A Simple Verification Point

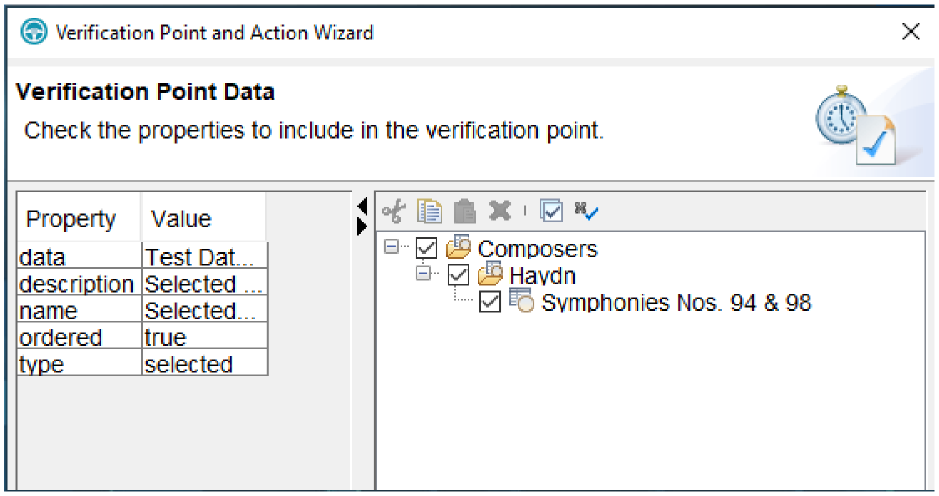
This lab will be essentially the same in structure as the demonstration of validating the total amount.

Follow the same procedure as demonstrated but this time create a verification point to the address and phone number displayed for the default user Trent Culpito.

This means there will be two verification points, one for each piece of data.

Part Two: A Selection Verification Point

One of the demonstrations done was to verify that the correct tree selection was made, the verification point shown here:



Verification Point Data

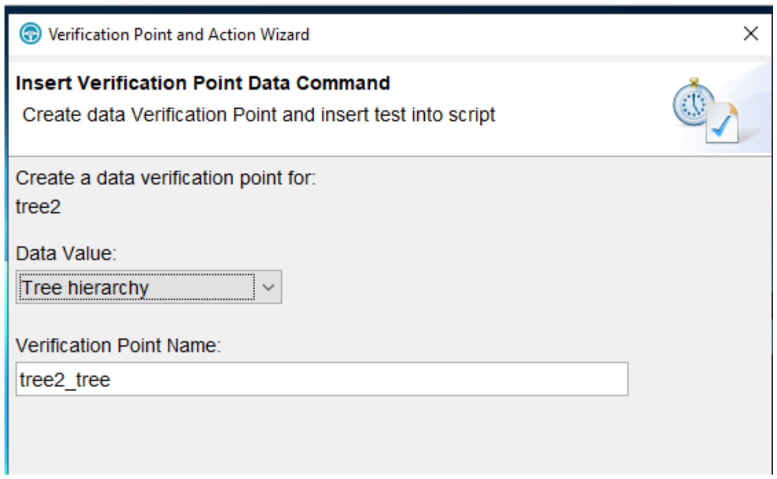
Check the properties to include in the verification point.

Property	Value
data	Test Dat...
description	Selected ...
name	Selected...
ordered	true
type	selected

☑ Composers
☑ Haydn
☑ Symphonies Nos. 94 & 98

This specific verification point only checks the selected portion of the tree.

For this lab, redo the demonstration of how to set the above verification point, but when you get to the place where you set the verification point, make the following selection instead (tree hierarchy instead of "selected tree hierarchy").



Verification Point and Action Wizard

Insert Verification Point Data Command

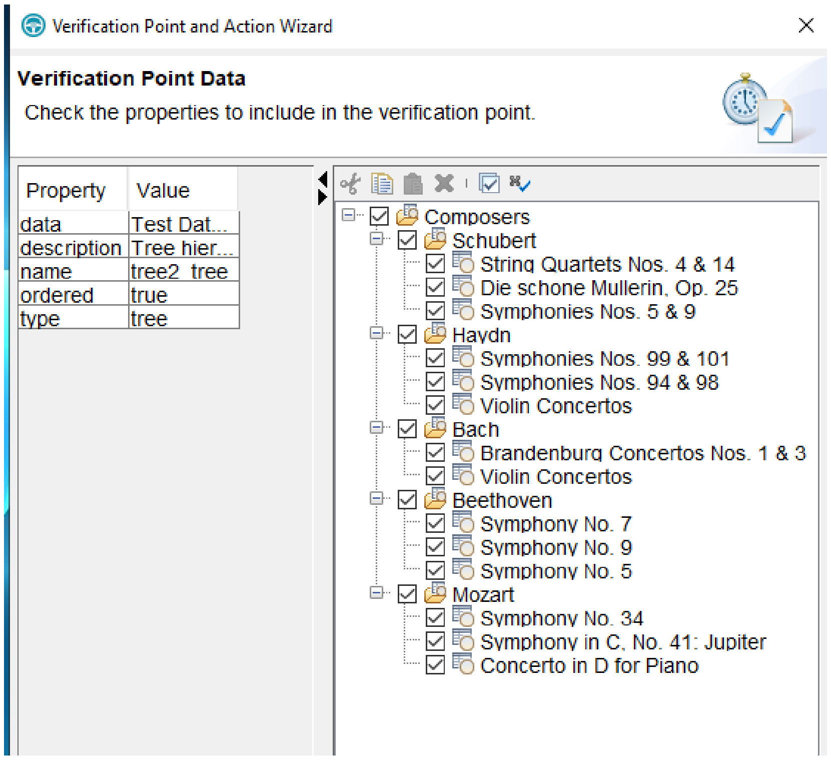
Create data Verification Point and insert test into script

Create a data verification point for:
tree2

Data Value:
Tree hierarchy

Verification Point Name:
tree2_tree

Which produces the verification point:



Verification Point and Action Wizard

Verification Point Data

Check the properties to include in the verification point.

Property	Value
data	Test Dat...
description	Tree hier...
name	tree2_tree
ordered	true
type	tree

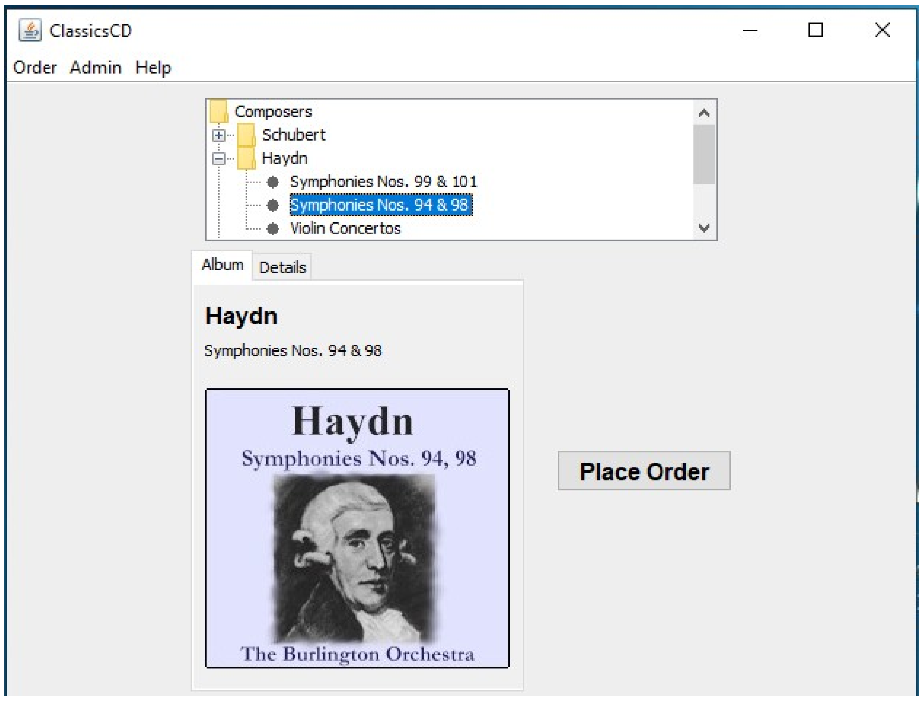
- ✓ Composers
  - ✓ Schubert
    - ✓ String Quartets Nos. 4 & 14
    - ✓ Die schone Mullerin, Op. 25
    - ✓ Symphonies Nos. 5 & 9
  - ✓ Haydn
    - ✓ Symphonies Nos. 99 & 101
    - ✓ Symphonies Nos. 94 & 98
    - ✓ Violin Concertos
  - ✓ Bach
    - ✓ Brandenburg Concertos Nos. 1 & 3
    - ✓ Violin Concertos
  - ✓ Beethoven
    - ✓ Symphony No. 7
    - ✓ Symphony No. 9
    - ✓ Symphony No. 5
  - ✓ Mozart
    - ✓ Symphony No. 34
    - ✓ Symphony in C, No. 41: Jupiter
    - ✓ Concerto in D for Piano

The difference between this and what we did in class, is that this style of verification point allows us to verify that specific parts of the selection tree are actually there, not just the ones that were selected. In the example above, all of the options are checked which means that this verification point passes only when the whole tree as shown is presented by the application.

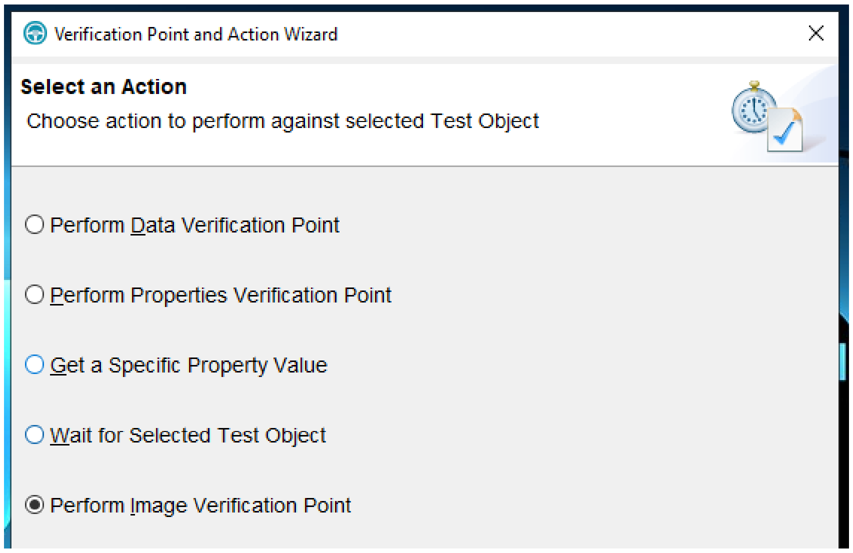
Part Three: Image Verification Point

An image verification point is used to confirm that the user is seeing the correct graphic at a specific point in the interaction. We assume that when we are recording the script, that the application is showing the correct graphic. When we are replaying the script, the verification point will fail if the graphic is not what we originally recorded.

For this exercise, you will create a verification point to ensure that the correct CD cover is displayed here:



The process is exactly the same as we have done previously, only this time, select the verification option as shown below:



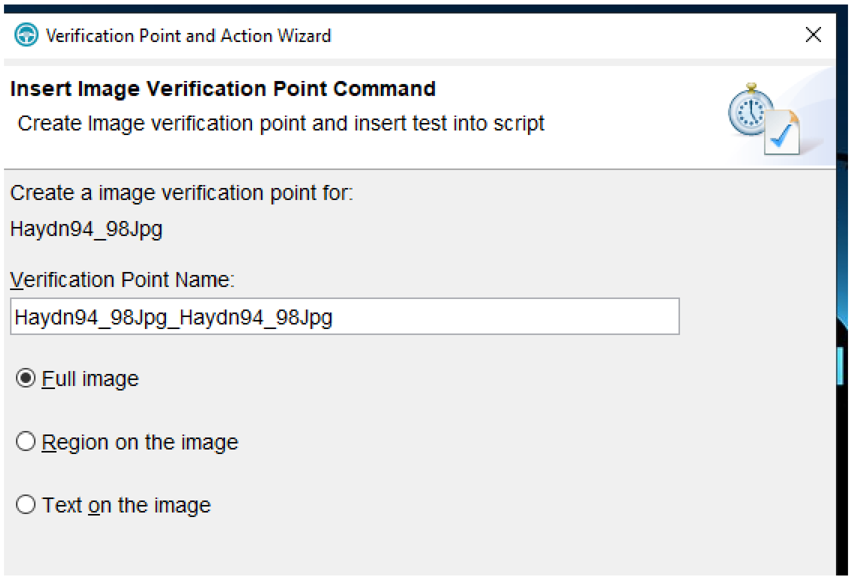
Verification Point and Action Wizard

Select an Action

Choose action to perform against selected Test Object

- ◯ Perform Data Verification Point
- ◯ Perform Properties Verification Point
- ◯ Get a Specific Property Value
- ◯ Wait for Selected Test Object
- ● Perform Image Verification Point

Notice that there are a number of options as to what we can verify.



Verification Point and Action Wizard

Insert Image Verification Point Command

Create Image verification point and insert test into script

Create a image verification point for:

Haydn94_98Jpg

Verification Point Name:

Haydn94_98Jpg_Haydn94_98Jpg

- ● Full image
- ◯ Region on the image
- ◯ Text on the image

For now, we are only going to verify the whole image so leave the default. The next screen should show you the image that will be used for verification purposes.

Complete the creation of the verification point and complete the script.

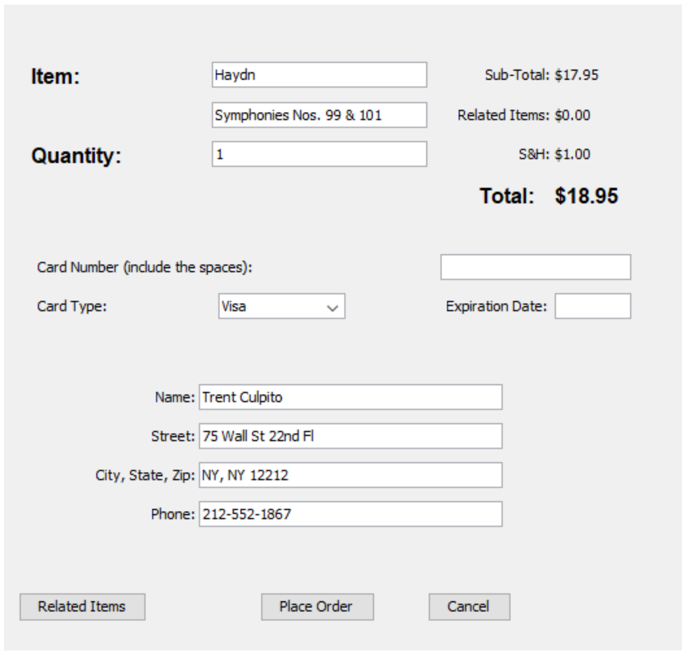
Once you have the finished script, confirm that the verification point is correct by examining the verification point in the "Script Explorer’

Optional: You may want to experiment with capturing a region of the image. Using the mouse for this can be tricky so be sure to read the instructions in the wizard as to which buttons to hold down and for how long.

Part Four: Property Verification Point

One of the walkthroughs that was presented was setting a verification point for a property of a test object. This section will do exactly the same thing, but using a different verification point option.

For this, we will want to verify that the text colour of the Total Amount is black in this screen:



Item: Haydn
Symphonies Nos. 99 & 101
Quantity: 1

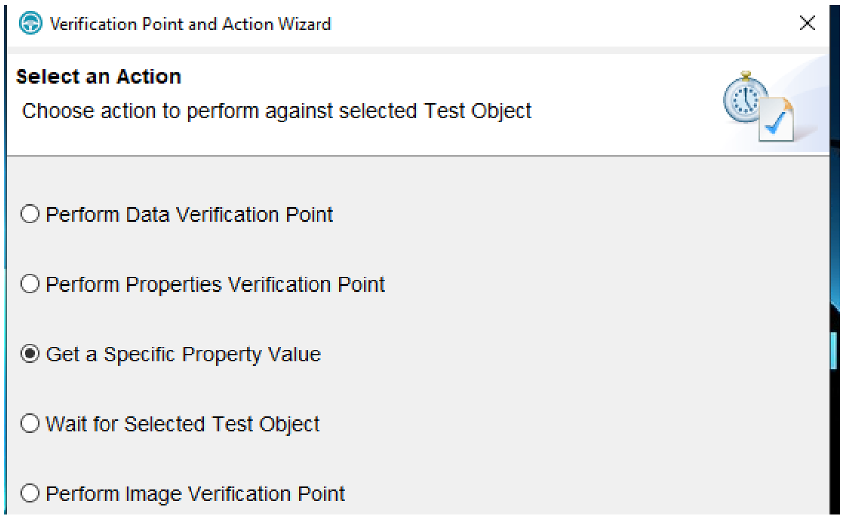
Sub-Total: $17.95
Related Items: $0.00
S&H: $1.00
Total: $18.95

Card Number (include the spaces):
Card Type: Visa
Expiration Date:

Name: Trent Culpito
Street: 75 Wall St 22nd Fl
City, State, Zip: NY, NY 12212
Phone: 212-552-1867

Follow the same steps until you get to the point where you have to chose the type of verification point to check. Previously, the option we used allowed us to describe a set of properties of an object, but often we are only interested in one specific property that we may want to work with.

In this case, the option we want to chose is "get a specific property value" as shown on the next page.



Verification Point and Action Wizard

Select an Action

Choose action to perform against selected Test Object

( ) Perform Data Verification Point

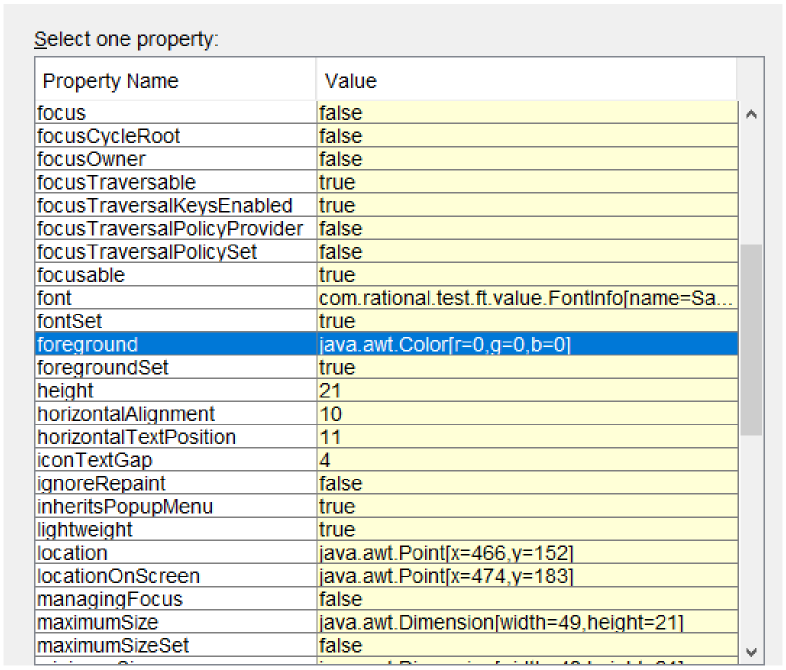
( ) Perform Properties Verification Point

[✓ Get a Specific Property Value]

( ) Wait for Selected Test Object

( ) Perform Image Verification Point

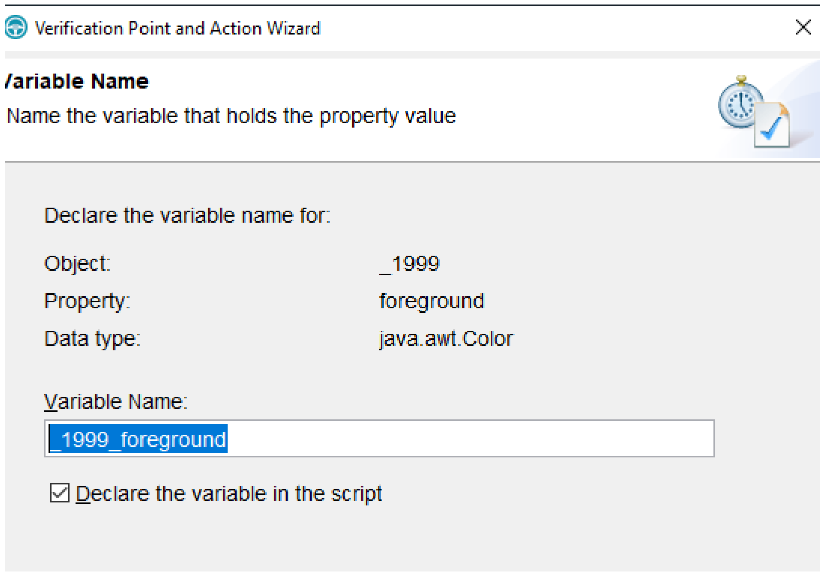
Which brings up a dialogue box where we select the foreground colour option.



Select one property:

Property Name	Value
focus	false
focusCycleRoot	false
focusOwner	false
focusTraversable	true
focusTraversalKeysEnabled	true
focusTraversalPolicyProvider	false
focusTraversalPolicySet	false
focusable	true
font	com.rational.test.ft.value.FontInfo[name=Sa...
fontSet	true
foreground	java.awt.Color[r=0,g=0,b=0]
foregroundSet	true
height	21
horizontalAlignment	10
horizontalTextPosition	11
iconTextGap	4
ignoreRepaint	false
inheritsPopupMenu	true
lightweight	true
location	java.awt.Point[x=466,y=152]
locationOnScreen	java.awt.Point[x=474,y=183]
managingFocus	false
maximumSize	java.awt.Dimension[width=49,height=21]
maximumSizeSet	false

However, notice that we are asked for a variable name in the next step.


Verification Point and Action Wizard

/ariable Name

Name the variable that holds the property value

Declare the variable name for:

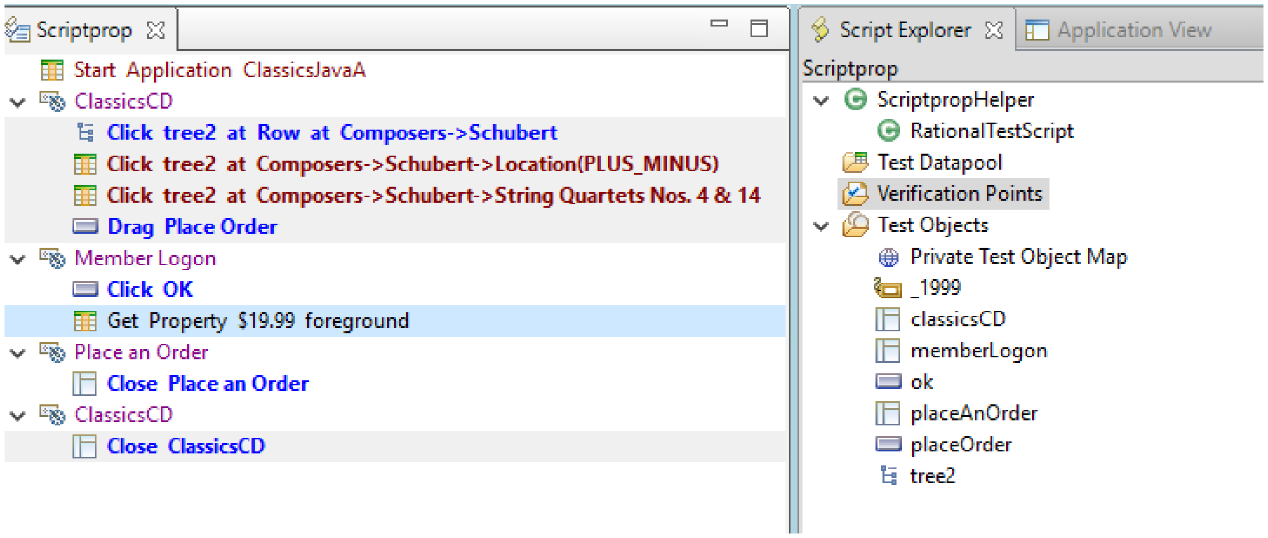
Object:	_1999
Property:	foreground
Data type:	java.awt.Color

Variable Name:

_1999_foreground

☑ Declare the variable in the script

Once the script is completed, notice that there is no verification point created, but instead, there is a line that says "get the property"


Scriptprop

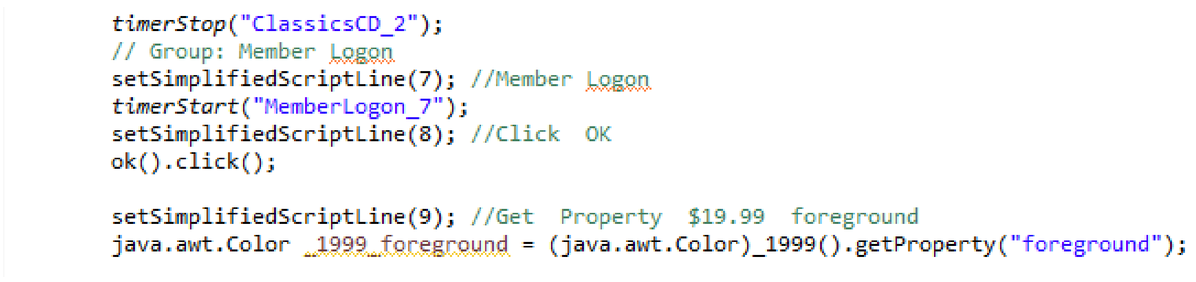
- Start Application ClassicsJavaA
- ClassicsCD
  - Click tree2 at Row at Composers->Schubert
  - Click tree2 at Composers->Schubert->Location(PLUS_MINUS)
  - Click tree2 at Composers->Schubert->String Quartets Nos. 4 & 14
  - Drag Place Order
- Member Logon
  - Click OK
  - Get Property $19.99 foreground
- Place an Order
  - Close Place an Order
- ClassicsCD
  - Close ClassicsCD

Script Explorer | Application View

- Scriptprop
  - ScriptpropHelper
    - RationalTestScript
  - Test Datapool
  - Verification Points
  - Test Objects
    - Private Test Object Map
    - _1999
    - classicsCD
    - memberLogon
    - ok
    - placeAnOrder
    - placeOrder
    - tree2

So what is the point of this operation? It is a bit beyond what we want to accomplish in this course, but what it does is to store the value of this property in a variable in the underlying Java RFT script so that it can be used later on in the script.

The actual place in the script where this takes place in the Java script can be seen here:



```
timerStop("ClassicsCD_2");
// Group: Member Logon
setSimplifiedScriptLine(7); //Member Logon
timerStart("MemberLogon_7");
setSimplifiedScriptLine(8); //Click  OK
ok().click();

setSimplifiedScriptLine(9); //Get  Property  $19.99  foreground
java.awt.Color _1999_foreground = (java.awt.Color)_1999().getProperty("foreground");
```

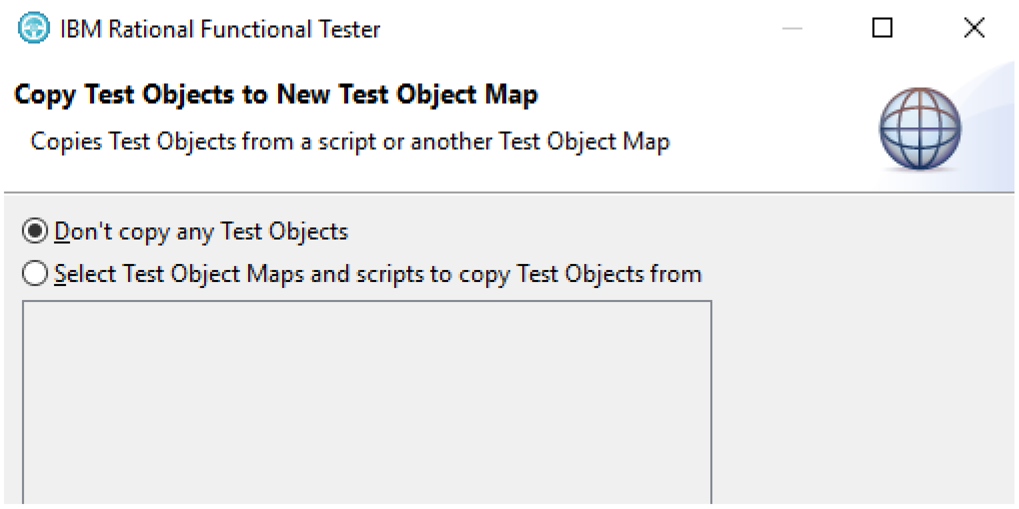
This feature can be thought of as a tool to assist programmers will be working directly with the scripts in Java to ensure they are getting the right references to the right program objects.

Lab Five

Shared Object Maps

Part One: Create a new Shared Test Object Map

1. Create a new RDT project.
2. Create a new shared test object map and set it as the default for new scripts.
3. Create the map with no objects:



IBM Rational Functional Tester

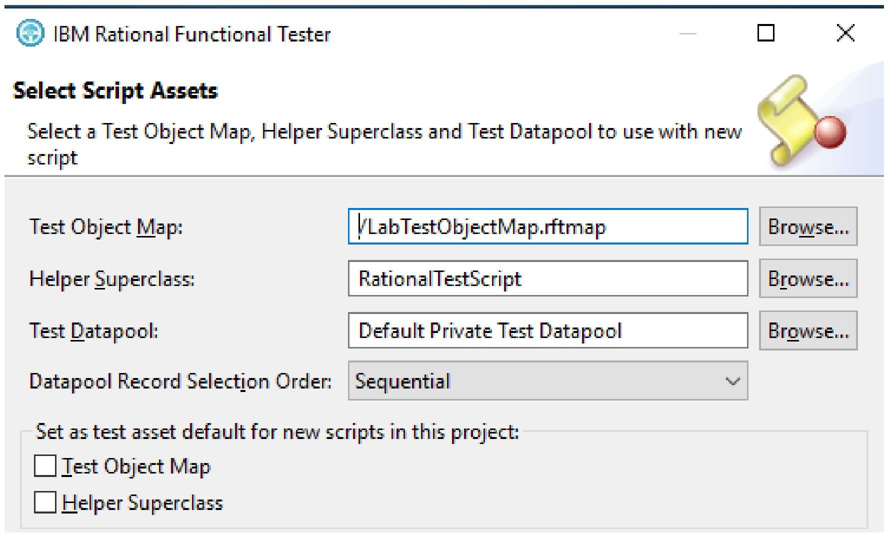
Copy Test Objects to New Test Object Map

Copies Test Objects from a script or another Test Object Map

[✓ Don't copy any Test Objects]

Select Test Object Maps and scripts to copy Test Objects from

4. Open up the map to ensure there is nothing in it.
5. Create a new script along the lines of the standard one we have been using during the class. Make sure that the shared map is being used.



Select Script Assets

Select a Test Object Map, Helper Superclass and Test Datapool to use with new script

Test Object Map: /LabTestObjectMap.rftmap

Helper Superclass: RationalTestScript

Test Datapool: Default Private Test Datapool

Datapool Record Selection Order: Sequential

Set as test asset default for new scripts in this project:

- [ ] Test Object Map
- [ ] Helper Superclass

6. Record the script.
7. Open the shared test object map and verify that test objects have been added.

Lab Six

Data Pools

This lab is essentially working through the various tasks described in the manual.

Part One: Creating a Datapool.

1. Create a new RFT project.
2. In the project, create an empty data pool by right mouse clicking on the project folder and selecting the create datapool option.
3. Give the datapool a name, and ensure it is empty by making sure that the "import" option does NOT specify a file.
4. Once the datapool is created, open it in the datapool editor to verify it is empty.

Part Two: Populating a Datapool

1. In the same project, start recording a script.
2. Make sure you use the datapool you have just created by selecting it in the "Next" tab when you create the script.
3. Open the ClassicsA and create the same basic script we have been using since the start of the course.
4. When you get to the screen where you enter the credit card number and expiration date, enter some values, then use the data driven option on the recorder to create the data structure for the variables. Ensure that you take the time to edit the variable names to make them more readable.
5. Once the data driven addition is completed, then set a verification point for the total price.
6. When you see the display window for the verification point, chose the option, as illustrated in the manual, to convert the hard coded value to a data pool variable. Choose an appropriate name for the variable.
7. Compete the script.
8. Open up the data pool to ensure that the data was correctly recorded.

9. Add several additional records to the data pool using the same data as you recorded (just to keep the amount of typing to a minimum) but change the credit card number so that you will have a way to see which data pool entry is being executed.
10. Go into the script and disable the script commands where you typed the hard coded values. Remember you may have to close and reopen the script window to see the commands in a disabled state.
11. Execute the script with the iteration option in the "Next" tab set to the number of records you have.
12. Verify that the script executes correctly.

Part Three: Using the DataPool

Notice that the script used in part 2 was used a convenience for us to populate the datapool. Normally, this would not be an operational script. The usual procedure is to create the test datapool, create the test scripts, then attach the datapool to the scripts.

1. Create a new script but ensure that the datapool you have created is selected in the "Next" tab.

2. Create the script to do the same thing that the last script did, or something similar. Do NOT use the data-driven option – make sure all of the values are hard-coded into the script.

3. Add a verification point for the amount, use the hard-coded value as shown.

4. Complete the script.

5. In the script window, use the Script option "replace literals with variables" and use the find and replace features to replace all of the hard coded values with data pool variables.

6. In the list of verification points, open the existing verification point and use the option to replace the hard coded value with the datapool variable.

7. Execute the script to ensure it works.