Use IBM Rational Functional Tester to automate custom controls

Level: Intermediate

Daniel Gouveia (dgouveia@us.ibm.com), Field Technical Sales Specialist, IBM

26 Jun 2007

Get an overview of the basic steps for using IBM Rational Functional Tester to automate the actions of custom controls.

Nothing makes a test automation project more interesting than custom controls, especially third-party custom controls. If you have ever done client/server testing, you know about this. If you have ever tested an applet, you have most likely run into these. More often than not, they represent a critical percentage of application functionality. Almost always, they push your test automation tool to the ground, pull its arm up behind its back, and make it scream "Enough, I give up!" Well, make it spit out X,Y coordinates anyway.

For this introduction, the tool subjected to such duress by custom controls is IBM® Rational® Functional Tester. This is the first installment of a two-part series about automating third-party custom controls. This first one is at a high level to provide a brief look at what is involved when you use Rational Functional Tester to automate these controls. The forthcoming associated tutorial will provide a real-world example that focuses on a popular custom control. It will give you step-by-step instructions on how to employ the three steps outlined in this article.

It is not uncommon for a testing group to be ignorant of the fact that the development team has decided to work with a set custom controls, either ones that they have created in-house or purchased from a third-party supplier. In fact, testers and quality assurance (QA) analysts usually become aware of this only as they start to record testing scripts. This is the point where you look at your dog and quote Dorothy from "The Wizard of Oz:" "Toto, I've a feeling we're not in Kansas anymore." In other words, you may not be sure what to do next in this predicament. Do you strap on the spark-generating ruby red slippers and click your heels three times, saying, "There's no test automation like object-oriented test automation"? Not likely. Besides, do you really want to be caught wearing ruby red slippers and chanting to yourself? Yet, if you're serious about continuing your test automation, then you probably need to start trudging down the yellow brick road until you come to the Emerald City of Automation Engineering. Wearing the slippers is optional.

The road to automation engineering

If you stop and think about it for a moment, you're trying to work with controls that you may know nothing about except, perhaps, what they are supposed to do. You know their "black box" behavior, but you are going to have to learn the inner-workings of these controls. Like Dorothy in the Land of Oz, you'll have to do a fair amount of searching to figure out how to make your test tool, Rational Functional Tester, interact with these controls. This will inevitably lend itself to trial and error. However, there are three basic steps that you can take to fend off those flying monkeys in Oz that try to waylay your efforts and stymie you during your automation project:

1. Figure out what you need to do to the controls to make them perform the actions that you want them to perform.
2. Get the information (documentation and such) for the controls, so that you can translate those actions into the language that the controls use.
3. Instruct Rational Functional Tester on how to make the controls perform the desired actions.

Step 1. Determine what you need the controls to do

First off, get down to basics. What actions do you want Rational Functional Tester to perform on the custom controls? Do you want it to extract data? Click on the control? Enter text?

Whatever it is, write it down. This is important stuff, albeit obvious stuff that you are probably thinking you don't need to document. Having it in writing will enable you to draw the parallels with the control's documentation later on. This is the critical first step to successfully getting Rational Functional Tester to interact with custom controls.

Step 2. Find out more about the custom controls

In Step 1, you formed an idea of what you need Rational Functional Tester to do to your custom controls (the actions). Step 2 focuses on finding the proper information for the controls. This could be as simple as asking a developer for the properties and methods to use for each control, or it could be a bit more involved. For instance, you could find yourself downloading a trial version of the custom controls merely to get the API (Application Programming Interface) documentation. You could also take advantage of the **Test Object Inspector** provided by Rational Functional Tester, which will show you all of the properties and methods that you can work with for any specific control.

By the way, did I mention that you're headed down the yellow brick road to the Emerald City of Automation *Engineering*? You will find yourself reading through a lot of fun information that describes how the controls do certain things. For instance, perhaps you need find out the number of rows in a given instance of a Sitraka table (something you should have documented in Step 1), and the Test Object Inspector informs you that you can call the `getNumRows()` method. This pretty much correlates with the action that you want, so you can flag this method to use.

Regardless, this is where the trial and error starts to come into play. After you know what actions to perform on the controls, you need to sift through your information and figure out how to tell them to perform these actions, using the language they understand. This process is somewhat like using a language dictionary in another country, in that you know what you want to say, so you just need to find the correct translation.

When you have your translations (for instance, which method to call or which property to reference), write them next to the control actions that you documented in Step 1. Now, whenever you need to perform a certain action, you know how to tell the control how to do it (again, which method to call or property to reference).

Step 3. Apply the information to Rational Functional Tester

Now you need open the Rational Functional Tester script editor to see what's behind the curtain, so to speak, as Dorothy did with the Wizard of Oz. This is where you'll notice that you're in engineering territory, because you are moving from test automation into automation *engineering*. I differentiate the two by the simple practice of hand-coding. In my opinion, when you write code, you are no longer only a QA analysts who does test automation. You have become an automation engineer, because you are devising a solution beyond the typical means. If you were to tie the assets that you acquired from the first two steps to this step, you would essentially be acting as a proxy. You need to tell Rational Functional Tester how to talk to the controls by using their language.

Here, you have two primary options.

- You can use the reflection methods (for example, `invoke()`) that Rational Functional Tester provides.
- You can use the property methods (`getProperty()`, `getProperties()`, `setProperty()` and so forth).

The decision about which to choose comes from the searching that you did in Step 2. For instance, if you discovered that you need to call a method, you would choose one of the `invoke()` (reflection) methods. After you have accomplished this, you have succeeded in automating Rational Functional Tester interaction with the custom controls that are strewn throughout your application.

Where this road leads from here

This article is the first installment of a two-part series about automating third-party custom controls. This article provided a brief look at what is involved when you use Rational Functional Tester to automate these controls. Automating third-party custom controls is not a trivial process. However, when you do it correctly, the scripts can become as close to bullet-proof as possible. The associated tutorial will provide a real-world example that focuses on a popular custom control. It will give you step-by-step instructions on how to employ the three steps outlined in this article.

Resources

Learn

- Get an Introduction to IBM Rational Functional Tester 7.0. The IBM Rational Functional Tester tool automates testing Java, .NET, and Web-based applications. Starting with Version 7.0, it includes support extensions for both Siebel and SAP, plus integration with IBM Rational ClearQuest, support for the Eclipse Test and Performance Tools Platform (TPTP) logs, and support for testing HTML applications with Mozilla Firefox. This article explains these new features and capabilities.

- Visit the Rational Functional Tester area on developerWorks for more articles, product information, and other resources.

- Visit the Rational software area on developerWorks for technical resources and best practices for the Rational Software Delivery Platform products.
- Subscribe to the developerWorks Rational zone newsletter. Keep up with developerWorks Rational content. Every other week, you'll receive updates on the latest technical resources and best practices for the Rational Software Delivery Platform.
- Subscribe to the Rational Edge e-zine for articles on the concepts behind effective software development.
- Browse the technology bookstore for books on these and other technical topics.

Get products and technologies

- Try a free trial version of Rational Functional Tester 7.0.
- Download trial versions of IBM Rational software.

Discuss

- Get involved in the developerWorks Functional and GUI Testing discussion forum. For users of Rational Functional Tester and for the discussion of general testing topics.
- Check out developerWorks blogs and get involved in the developerWorks community.

About the author

Dan Gouveia is a sales engineer for IBM, primarily focusing is on Rational’s functional and performance testing tools. He works with customers on implementing these tools in the most effective manner, often dealing with topics such as keyword- driven testing, developing automation frameworks, etc.