Ways to use IBM Rational Functional Tester more effectively

Extend the Quality Software Engineering framework to improve productivity and make maintenance easier

Skill Level: Intermediate

Masahiro Ohkawa (mohkawa@jp.ibm.com)
Yamato Software Development Laboratory (YSL)
IBM Japan

Gou Nakashima (gnaka@jp.ibm.com)
Yamato Software Development Laboratory (YSL)
IBM Japan

16 Jul 2009

IBM Rational Functional Tester provides features by which users can automate testing of their GUI applications effectively. The IBM Quality Software Engineering (QSE) team introduced a hierarchical framework, known as the QSE framework, to make maintenance easier. This framework uses the Rational Functional Tester object recognition feature. This article shows you ways to extend your QSE framework and how to use Rational Functional Tester features to improve productivity, too. You will also get tips for testing applications that support multiple languages effectively.

Introduction

IBM® Rational® Functional Tester provides features to automate testing. To use the recording feature, record a test activity and save it as a test script. After that, you can run the test script whenever you want to execute the same test.

When you want to input data while you are testing, you can use the test data pool feature during recording. If you use the test data pool feature, you can change the input data before you run the test script, without re-recording the test activity. You

can use the verification point feature by which Rational Functional Tester compares the actual values with the expected values, and then generates a report with detailed information that shows whether or not you got the expected results. However, a possible concern when you use the recording feature is that you may need to re-record the test activity when your application to be tested is changed.

To minimize your effort, the IBM Quality Software Engineering (QSE) team has introduced a hierarchical framework, known as the QSE framework. The QSE framework isolates GUI objects with test cases. Rational Functional Tester provides the Test Object Map feature by which GUI objects can be obtained. Actions to GUI can be executed by using the methods of the GUI objects. The QSE framework consists of the following three tiers.

- The **appobjects** folder contains GUI objects obtained by the test object map feature.
- The **tasks** folder contains programs for executing GUI actions by using objects in the appobjects folder. GUI actions can be categorized and implemented as tasks. For example, the login process can be implemented as a task.
- The **testcases** folder contains programs you can use to execute test cases by using programs in the tasks folder, and to log the results.

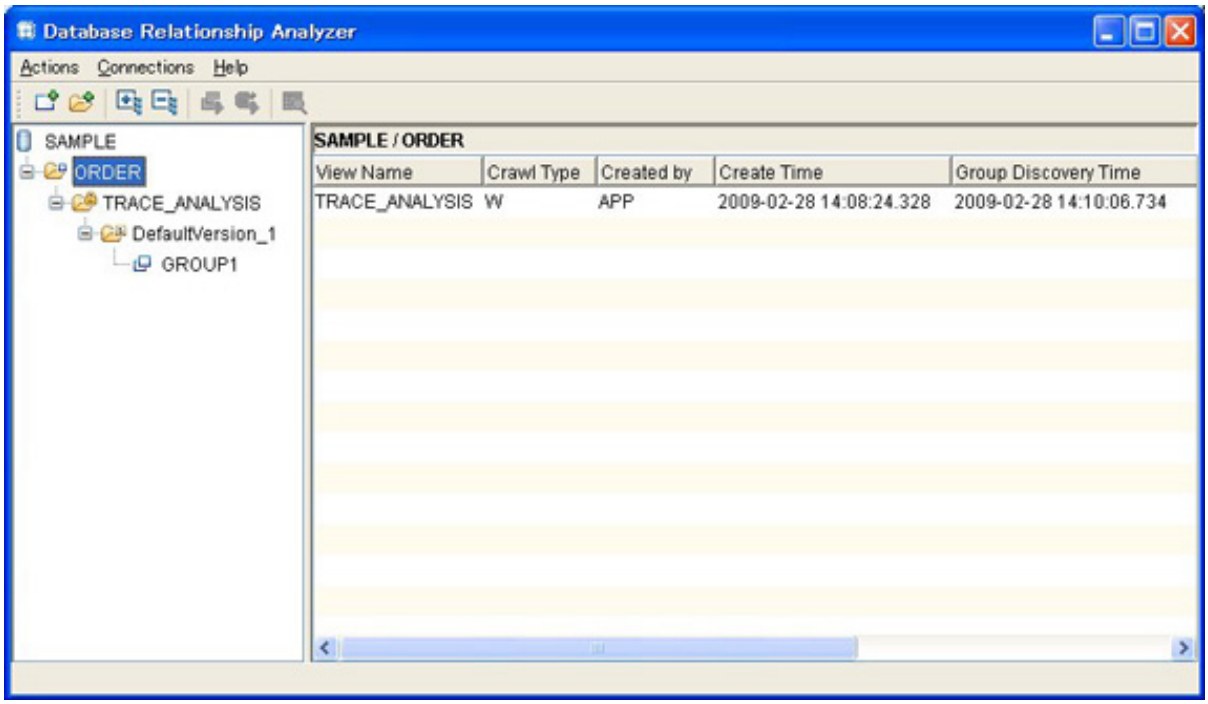
You can use the QSE framework to improve maintainability. For more information about the QSE framework, refer to the *An Object-Oriented framework for IBM Rational Functional Tester* item in the Resources section.

This article shows you how to automate testing more effectively by extending the QSE framework and utilizing Rational Functional Tester features, such as test data pool and verification points. This article also gives you some tips about how to test applications that support multiple languages.

Provide GUI control method

Consider testing the application shown in Figure 1.

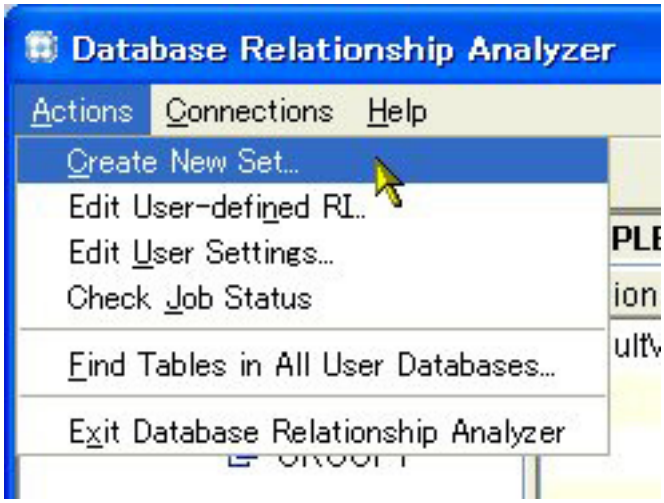
Figure 1. Application GUI



Click to enlarge

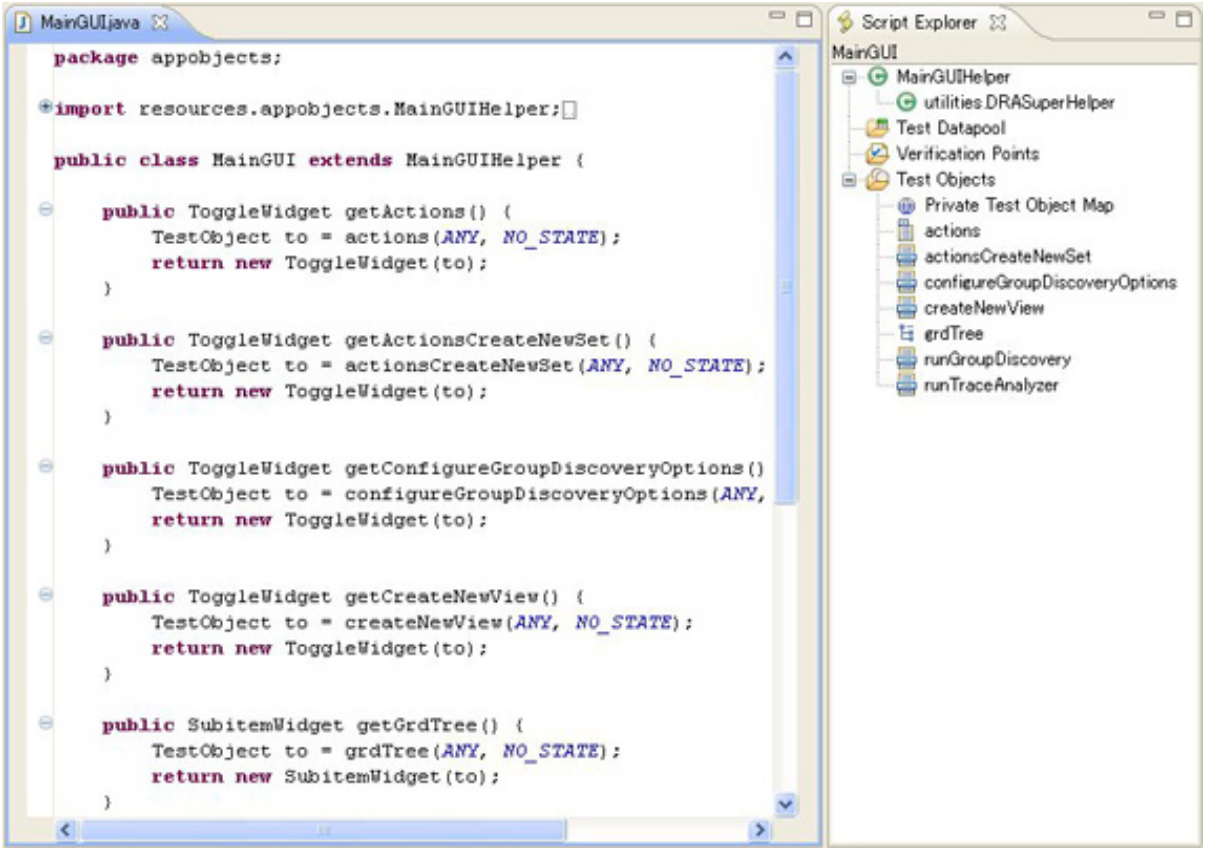
In this GUI, you can select the **Actions > Create New Set** menu item, as shown in Figure 2. Those objects can be obtained by the Test Object Map feature.

Figure 2. Application menus



With the jar file provided with the QSE framework, you can define the methods shown in Figure 3 and Listing 1 to get GUI objects obtained by the Test Object Map feature.

Figure 3. Test objects and program to get GUI objects



Click to enlarge

Listing 1. Program to get GUI objects

```
package appobjects;

import resources.appobjects.MainGUIHelper;
import ibm.widgets.*;
import ibm.widgets.ancestors.*;
import ibm.widgets.swt.*;

import com.rational.test.ft.*;
import com.rational.test.ft.object.interfaces.*;
import com.rational.test.ft.script.*;
import com.rational.test.ft.value.*;
import com.rational.test.ft.vp.*;

public class MainGUI extends MainGUIHelper {

  public ToggleWidget getActions() {
    TestObject to = actions(ANY, NO_STATE);
    return new ToggleWidget(to);
  }

  public ToggleWidget getActionsCreateNewSet() {
    TestObject to = actionsCreateNewSet(ANY, NO_STATE);
    return new ToggleWidget(to);
  }

  public ToggleWidget getConfigureGroupDiscoveryOptions() {
    TestObject to = configureGroupDiscoveryOptions(ANY, NO_STATE);
    return new ToggleWidget(to);
```

```
  }

  public ToggleWidget getCreateNewView() {
    TestObject to = createNewView(ANY, NO_STATE);
    return new ToggleWidget(to);
  }

  public SubitemWidget getGrdTree() {
    TestObject to = grdTree(ANY, NO_STATE);
    return new SubitemWidget(to);
  }

  public ToggleWidget getRunGroupDiscovery() {
    TestObject to = runGroupDiscovery(ANY, NO_STATE);
    return new ToggleWidget(to);
  }

  public ToggleWidget getRunTraceAnalyzer() {
    TestObject to = runTraceAnalyzer(ANY, NO_STATE);
    return new ToggleWidget(to);
  }

  // ----------------------------------------

  public void testMain (Object[] args)
  {
     // Unit testing can go here
     // (will be deleted next time ClasGenerator is run)
  }
}
```

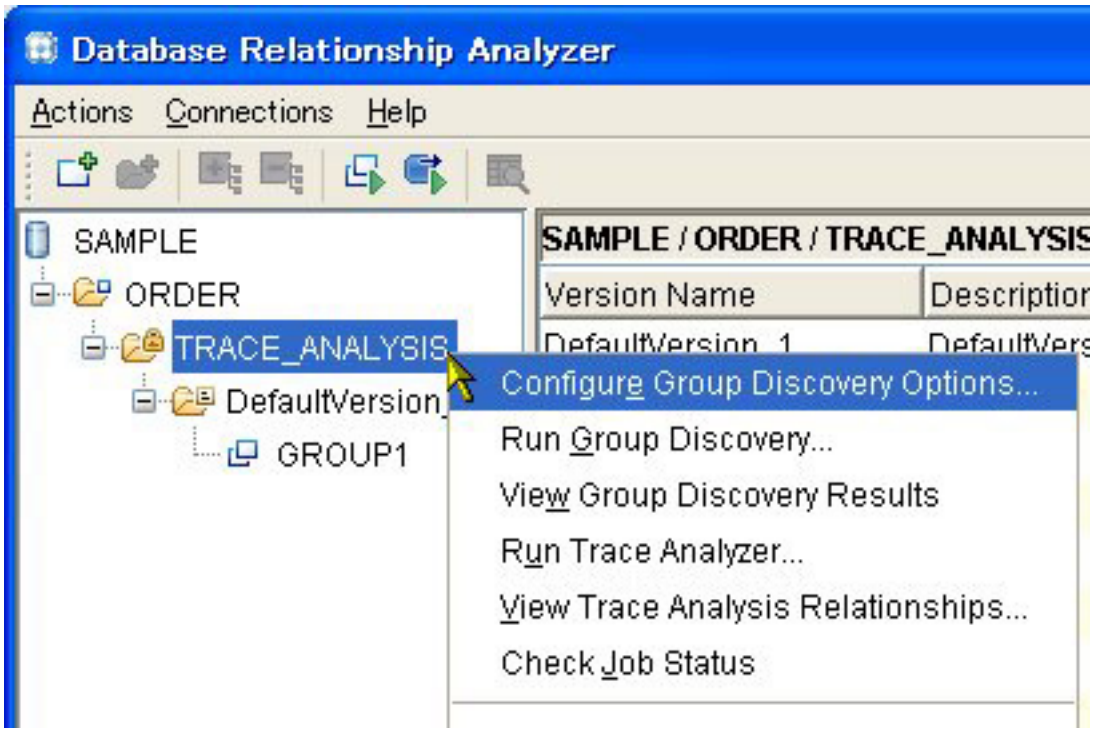
The program in Listing 2 simulates clicking the **Actions** menu, and then clicking the **Create New Set** sub-menu.

Listing 2. Example program to click a menu item

```
MainGUI mainGui = new MainGUI();
mainGui.getActions().click();
mainGui.getActionsCreateNewSet().click();
```

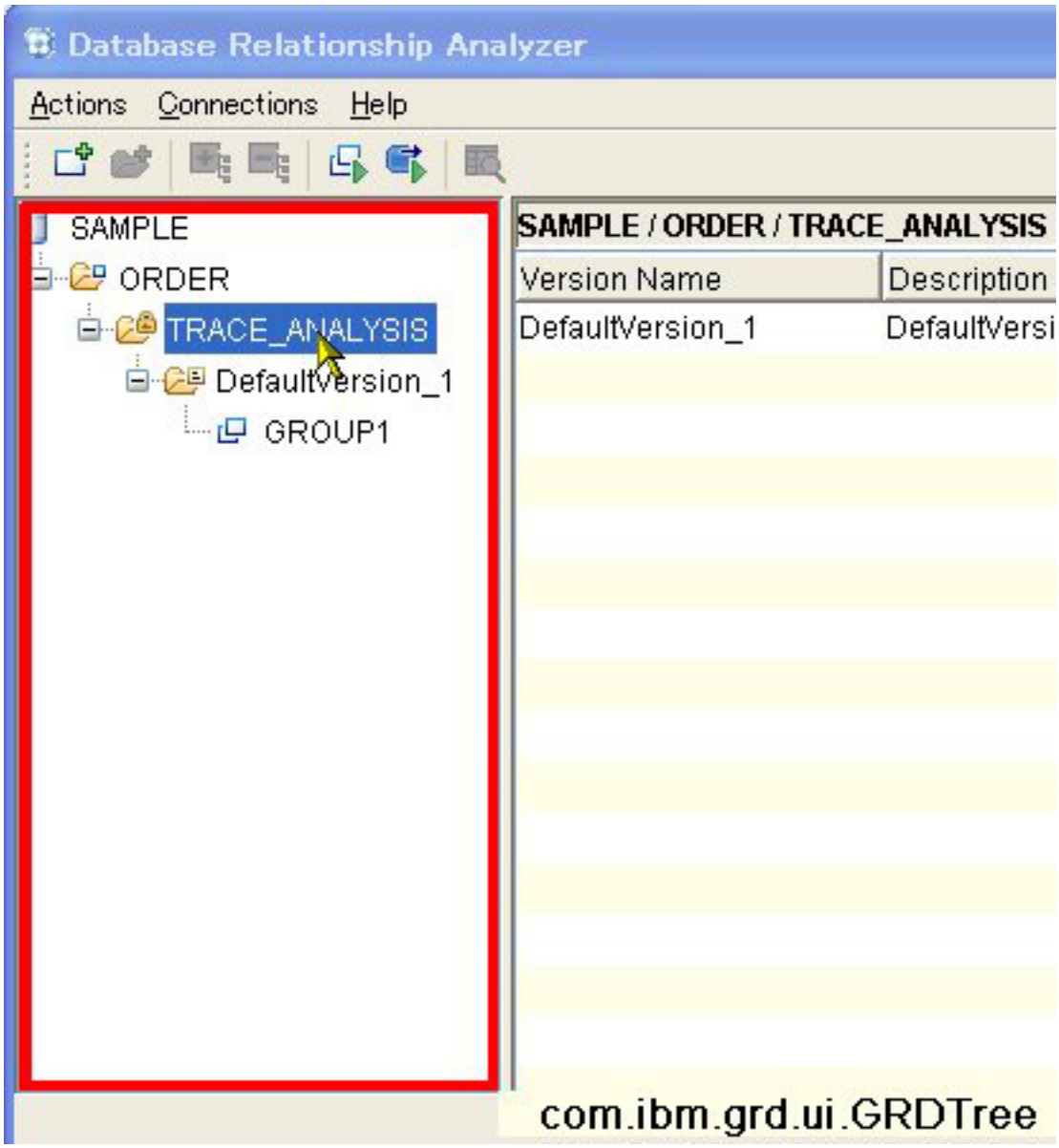
As this example shows, you can easily manipulate your GUI by using GUI objects. However, it can be difficult to find which method is used to execute a GUI action. Figure 4 shows an example: consider the case when you right-click the node of the tree.

Figure 4. Pull down menu of a tree node



In the case of a tree, you cannot get each tree node as an object. As shown in Figure 5, you need to get the tree object, and then use one of the methods to right-click a node of the tree.

Figure 5. Capture a tree object



If you don't know how a GUI action is executed with a GUI object (like in this example), you can use the recording feature, and then refer to the generated program. In this example, when you right-click the node while recording, you get a program like that shown in Listing 3.

Listing 3. Generated program to simulate right-clicking the node of the tree

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

        // Frame: Database Relationship Analyzer
        grdTree().click(RIGHT, atPath("SAMPLE->ORDER->TRACE_ANALYSIS"));
}
```

It's not efficient to write this kind of program in the tasks folder whenever you need it.

Instead, it's good to write a generic method in a class to simulate the above action, which inherits the GUI object class in the appobjects folder. This type of class doesn't need to be created as a Rational Functional Tester script: creating it as a Java™ class is enough. Create a folder, such as one called `apputils`, for storing this kind of classes. The program in Listing 4 shows a generic method for right-clicking the tree node. The `MainGUIUtil` class inherits the `MainGUI` class. The `MainGUI` class contains the GUI objects in the main window, and it is placed under the appobjects folder. The `grdTree` method previously used to get the tree object is replaced with that defined in the `MainGUI` class, which is the `getGrdTree` method in this example.

Listing 4. Example program to simulate right-clicking the node of the tree

```
public class MainGUIUtil extends MainGUI {
        public void rightClickView(String dbName, String setName, String viewName) {
                String s = dbName + "->" + setName + "->" + viewName;
                getGrdTree().click(RIGHT, atPath(s));
        }
```

By using methods like these, programs in the tasks folder can be implemented intuitively with GUI actions .

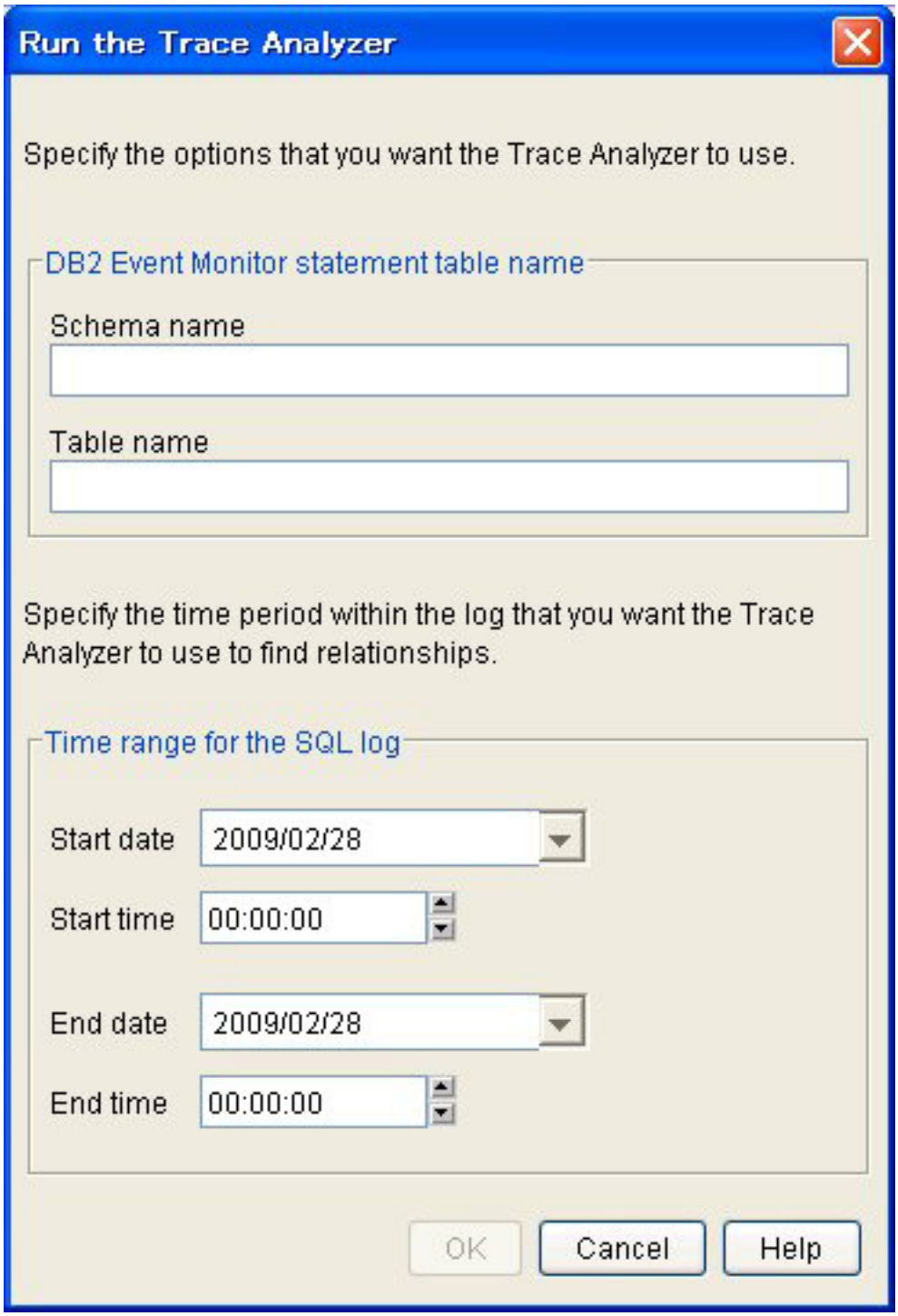
Effectively automate data input processing

With the QSE framework, the steps to input data are as follows.

1. Capture each input field to get its GUI object by using the Test Object Map feature.
2. Define methods to get the GUI objects.
3. Set input values by using the methods of the GUI objects.

You can effectively perform data input actions by using the Test Datapool feature. For example, Figure 6 shows a GUI with several input fields. Consider inputting data in this GUI.

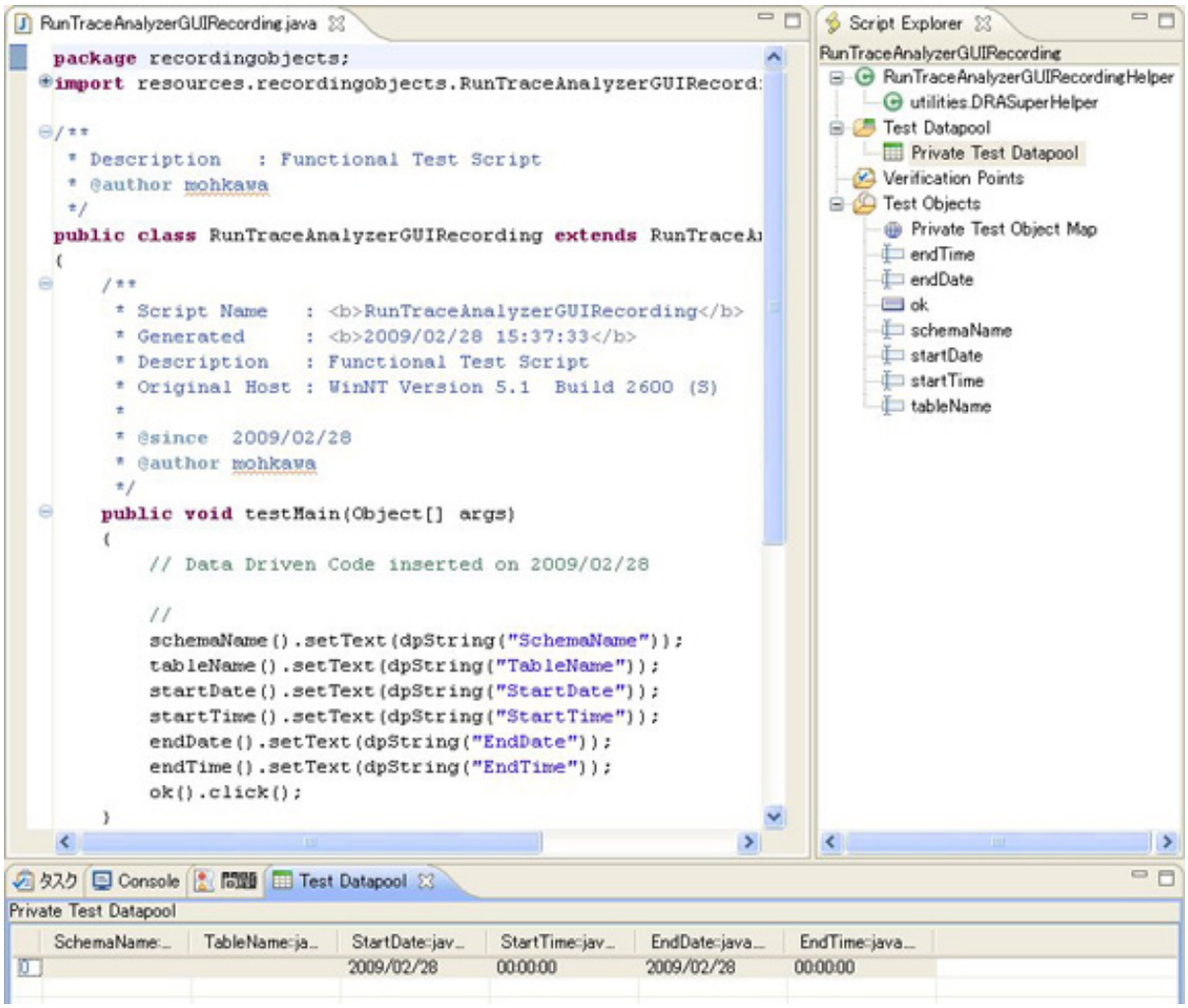
Figure 6. Application GUI with input fields



After opening this GUI, start recording, then capture this window by selecting **Insert Data Driven Commands**. After that, pause your recording, input dummy data to enable the **OK** button, resume recording, click **OK**, and then stop recording. When you do so, the program shown in Figure 7 and Listing 5 is generated. If you want to apply this technique to a wizard that consists of multiple panels, insert a data-driven

command in each panel so that you can parameterize all of the input fields.

Figure 7. Program generated with the Test Datapool feature



Click to enlarge

Listing 5. Generated program

```
package recordingobjects;
import resources.recordingobjects.RunTraceAnalyzerGUIRecordingHelper;
import com.rational.test.ft.*;
import com.rational.test.ft.object.interfaces.*;
import com.rational.test.ft.object.interfaces.SAP.*;
import com.rational.test.ft.object.interfaces.WPF.*;
import com.rational.test.ft.object.interfaces.dojo.*;
import com.rational.test.ft.object.interfaces.siebel.*;
import com.rational.test.ft.object.interfaces.flex.*;
import com.rational.test.ft.script.*;
import com.rational.test.ft.value.*;
import com.rational.test.ft.vp.*;

public class RunTraceAnalyzerGUIRecording extends RunTraceAnalyzerGUIRecordingHelper
{
  public void testMain(Object[] args)
  {
```

```
    // Data Driven Code inserted on 2009/02/28

    //
    schemaName().setText(dpString("SchemaName"));
    tableName().setText(dpString("TableName"));
    startDate().setText(dpString("StartDate"));
    startTime().setText(dpString("StartTime"));
    endDate().setText(dpString("EndDate"));
    endTime().setText(dpString("EndTime"));
    ok().click();
  }
}
```

If you define a method that contains this program, other programs can invoke the method. When you invoke the method, values defined in the test datapool are set in the appropriate fields. As you can see in Figure 7 and Listing 5, the `dpString` method is used for getting a value from the test datapool.

If you change the method to pass input values via the method arguments, the method can be used generically. In this example, if you use the test datapool values for the start time and the end time, and set other input values dynamically via the method arguments, the method is defined as shown in Listing 6.

Listing 6. Example program to input data dynamically

```
public void runTraceAnalyzer(String schema,
                        String table,
                        String startDate,
                        String endDate)
{
        schemaName().setText(schema);
        tableName().setText(table);
        startDate().setText(startDate);
        startTime().setText(dpString("StartTime"));
        endDate().setText(endDate);
        endTime().setText(dpString("EndTime"));
        ok().click();
}
```

Automate results verification

In case some results are displayed on GUI, it's easy to compare the actual values with the expected values by using the verification point feature.

Verify results statically

For example, if you expect to get the table described in Figure 8, capture the table as a verification point after you start recording, and then stop the recording. This produces the program described in Figure 9 and Listing 7.

Figure 8. Table containing some results

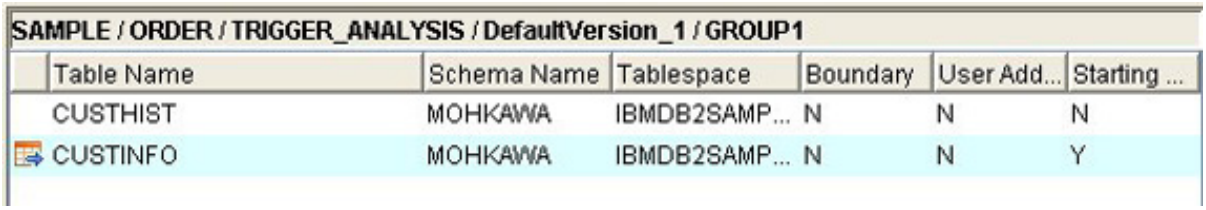
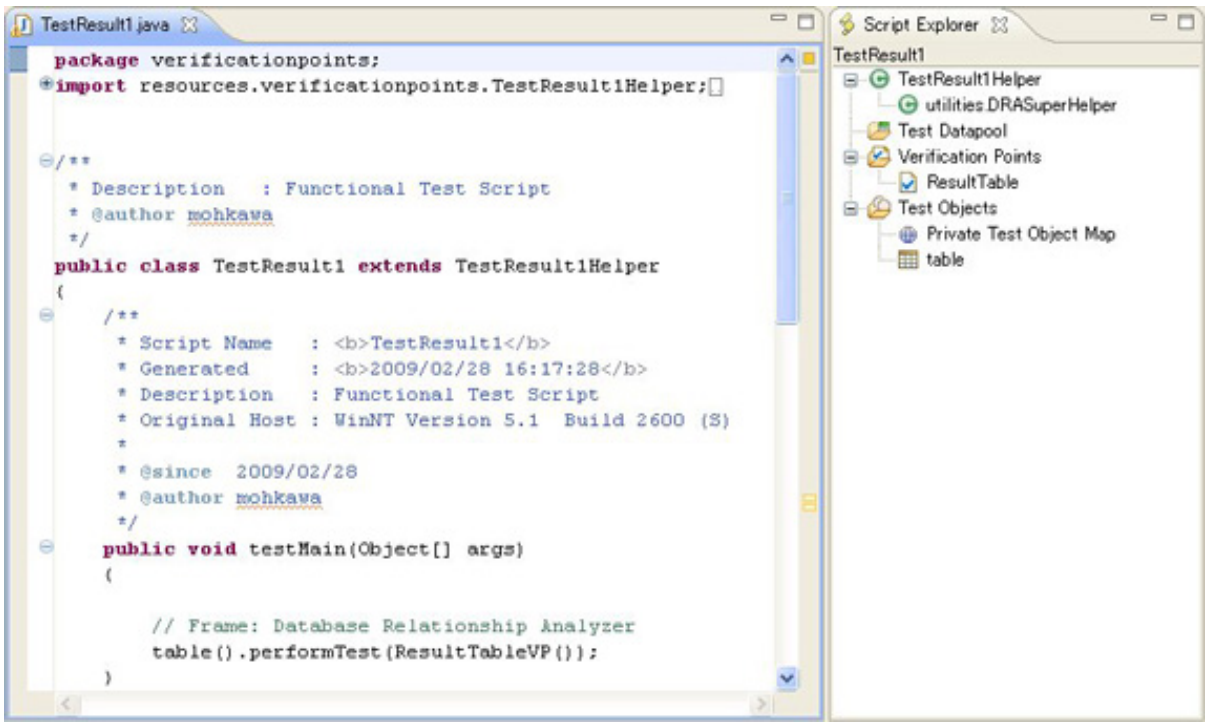


Figure 9. Program generated with the verification point feature



Click to enlarge

Listing 7. Generated program

```
package verificationpoints;
import resources.verificationpoints.TestResult1Helper;
import com.rational.test.ft.*;
import com.rational.test.ft.object.interfaces.*;
import com.rational.test.ft.object.interfaces.SAP.*;
import com.rational.test.ft.object.interfaces.WPF.*;
import com.rational.test.ft.object.interfaces.dojo.*;
import com.rational.test.ft.object.interfaces.siebel.*;
import com.rational.test.ft.object.interfaces.flex.*;
import com.rational.test.ft.script.*;
import com.rational.test.ft.value.*;
import com.rational.test.ft.vp.*;

public class TestResult1 extends TestResult1Helper
{
  public void testMain(Object[] args)
  {
    table().performTest(ResultTableVP());
  }
}
```

Copy the above program to create the method shown in Listing 8. If you invoke the

method, Rational Functional Tester automatically compares the actual values with the expected values.

Listing 8. Example method to verify result

```
public void verifyResult() {
        table().performTest(ResultTableVP());
}
```

Verify results dynamically

If you want to change the expected values dynamically, you can use the `IFtVerificationPoint vpManual(java.lang.String vpName, java.lang.Object expected, java.lang.Object actual).performTest method`. The `vpManual` method has three arguments. The first argument is to set a verification point name that is unique in the project, the second argument is to set an expected object, and the third argument is to set an actual object. If you want to compare values in a table like this example, you can use the `com.rational.test.ft.vp.impl.TestDataTable` class object in the second and third arguments of the `vpManual` method. You can create the expected object as shown in Listing 9. You can also specify the region to be compared, which is done with the `setComparisonRegions` method.

Listing 9. Example program to create a table as expected results

```
TestDataTable tbl = new TestDataTable();
tbl.setColumnHeader(0, "Table Name");
...
tbl.insert(new String[7], 0);
tbl.setCell(0, 1, "CUSTINFO");
...
TestDataTableRegions regions = new TestDataTableRegions();
regions.addRegion(TestDataTableRegion.allCells());
tbl.setComparisonRegions(regions.getRegions());
```

In the following example, you use the verification point object as the expected object, which was created by capturing the GUI table as a verification point in the previous example. Next, modify a value of the object. To get the actual values, you can use the `getTestData(String testDataType)` method of the `table` object obtained by the test object map, as shown in Listing 10. The `testDataType` argument has the value of the property `type` displayed in the verification point window. You can see it in Figure 10. In this case, this value is "visible contents".

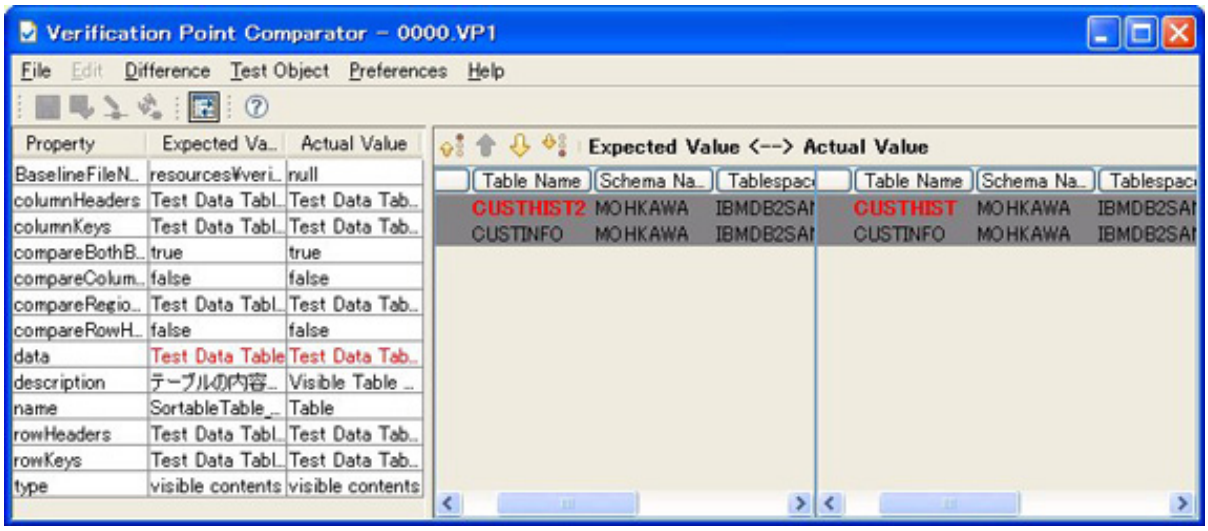
Listing 10. Example method to verify a table result

```
Public void verifyResult() {
        TestDataTable tbl = (TestDataTable)ResultTableVP().getBaselineData();
        tbl.setCell(0, 1, "CUSTHIST2");
        vpManual("VP1", tbl, table().getTestData("visible contents")).performTest();
}
```

In this program, "CUSTHIST2" is set in a cell, where it used to be "CUSTHIST". Running this method results in an error, and you can get the information shown in

Figure 10.

Figure 10. Detail result in a verification point (Table)



Click to enlarge

The previous example discusses the `table` object. The other objects that can be compared are shown in Table 1.

Table 1. Objects that can be compared

GUI Type	Class
Tree	com.rational.test.ft.vp.impl.TestDataTree
List	com.rational.test.ft.vp.impl.TestDataList
Menu bar	com.rational.test.ft.vp.impl.TestDataTree
Character string	String

In the case of the `tree` object, Listing 11 shows a program to add a node to a `tree` object captured as verification point.

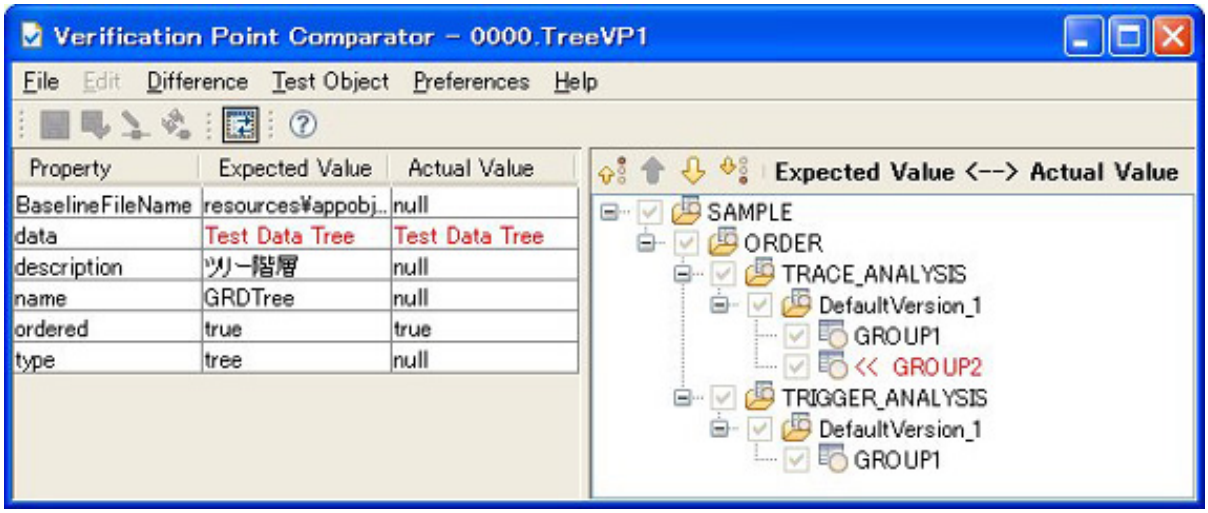
Listing 11. Example method to verify a tree result

```
public void verifyResult() {
        TestDataTree tree = (TestDataTree)GRDTreeVP().getBaselineData();
        TestDataTreeNodes nodes = (TestDataTreeNodes)tree.getTreeNodes();
        TestDataTreeNode node = (TestDataTreeNode)nodes.getRootNodes()[0];
        node = (TestDataTreeNode)node.getChild(0);
        node = (TestDataTreeNode)node.getChild(0);
        node = (TestDataTreeNode)node.getChild(0);

        TestDataTreeNode node2 = new TestDataTreeNode();
        node2.setNode("GROUP2");

        TestDataTreeNode[] n = new TestDataTreeNode[2];
        n[0] = (TestDataTreeNode)node.getChild(0);
        n[1] = node2;
        node.setChildren(n);
```

```
        vpManual("TreeVP1", tree, grdTree().getTestData("tree")).performTest();
}
```

When you run this program, you get the following information (Figure 11).

Figure 11. Detail result in a verification point (Tree)



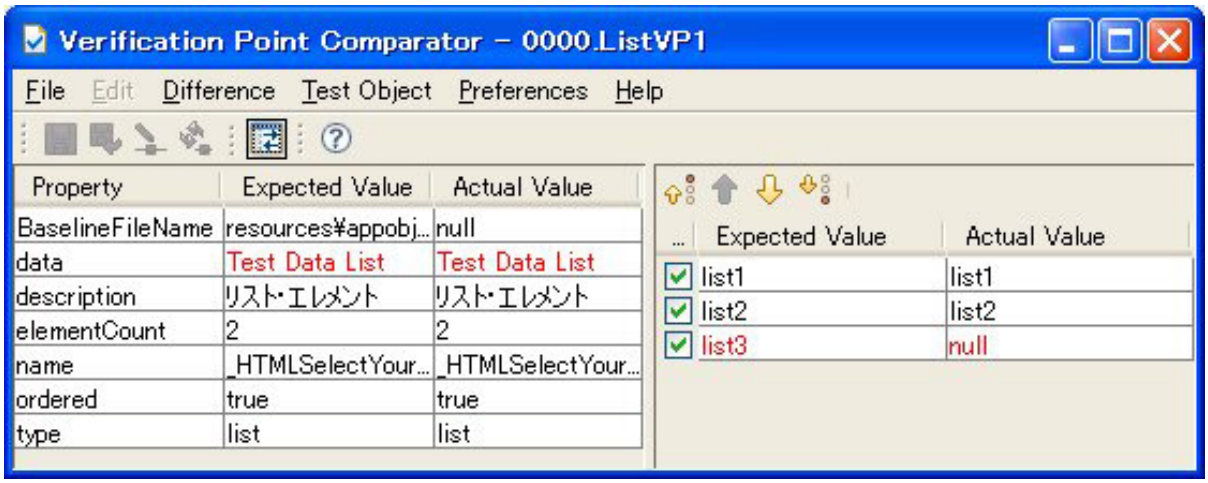
In the case of `list`, Listing 12 shows a program to add an item to a `list` object captured as verification point.

Listing 12. Example method to verify a list result

```
public void verifyResult() {
        TestDataList list = (TestDataList)TestListVP().getBaselineData();
        TestDataElementList elemList = (TestDataElementList)list.getData();

        TestDataElement elem = new TestDataElement();
        elem.setElement("list3");

        elemList.add(elem);

        vpManual("ListVP1", list, list().getTestData("list")).performTest();
}
```

When you run this program, you get the following information (Figure 12).

Figure 12. Detail result in a verification point (List)



Test an application that supports multiple languages

To test an application in multiple languages, you can use resource files that contain key and value pairs (`[KEY]=[VALUE]`), where the values are displayed in a GUI. An example of the resource file is one used in the `ResourceBundle` class of Java. The steps to configure this are as follows.

1. The following property in the ivory.properties file in Rational Functional Tester's bin directory is enabled (by default, it's commented out.)

```
# When enabled this property allows string look up in a localized string table,
 if available
rational.test.ft.services.enable_localization=true
```

2. Resource files of the application that consist of key and value pairs are copied under the resources folder of the Rational Functional Tester project after changing the base name of the file to the Rational Functional Tester project name. Of course it's OK to create the file instead. The resource file name format is:

   - [Rational Functional Tester project name]_[language].properties

   For example, in case the Rational Functional Tester project name is "DRAProject", the Japanese resource file name is:

   - DRAProject_ja.properties

3. Restart Rational Functional Tester.

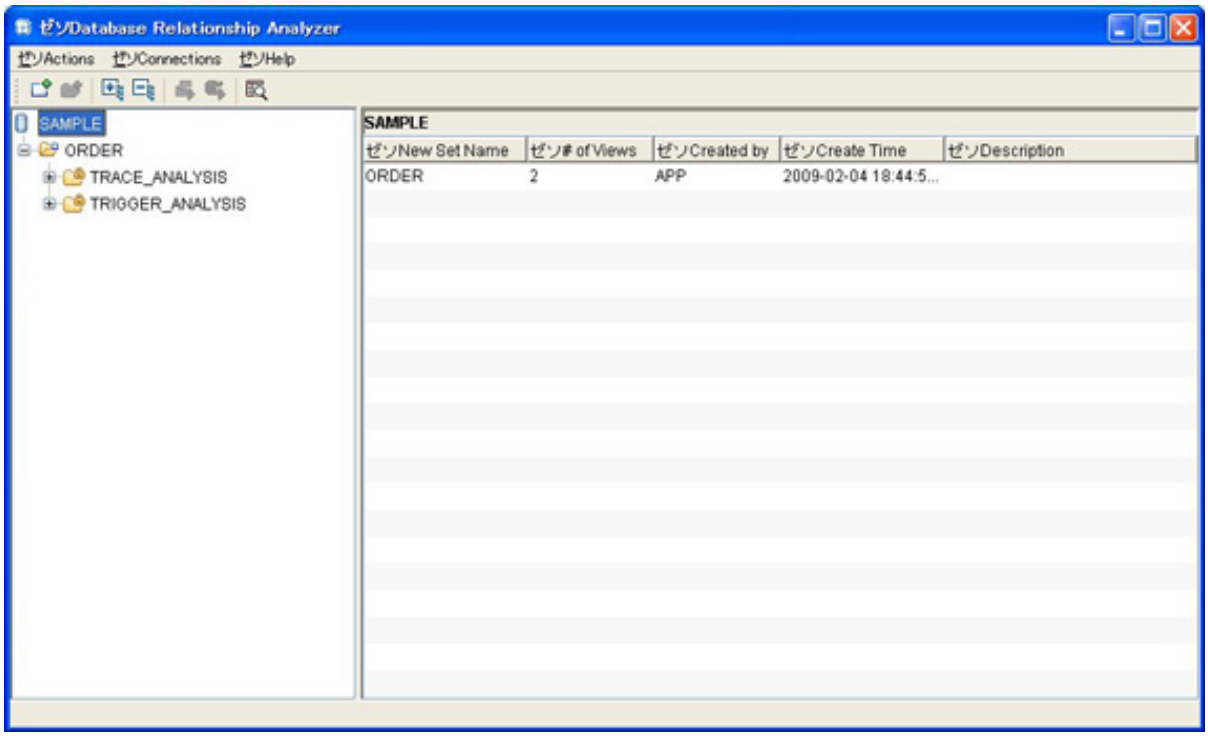
4. Open the test object map, and then change the property values that are GUI display values to key names of the resource files. Some property names, like the following, have GUI display values.

- accessibleContext.accessibleName
- .captionText
- name
- text

In the case of sub menu, some property value is concatenated with parent menu name separated by an underscore ("_"). In that case, change the weight to 0.

For example, copy the English resource file to the Japanese one, then add some Japanese characters in front of each value. You will do so in the GUI shown in Figure 13.

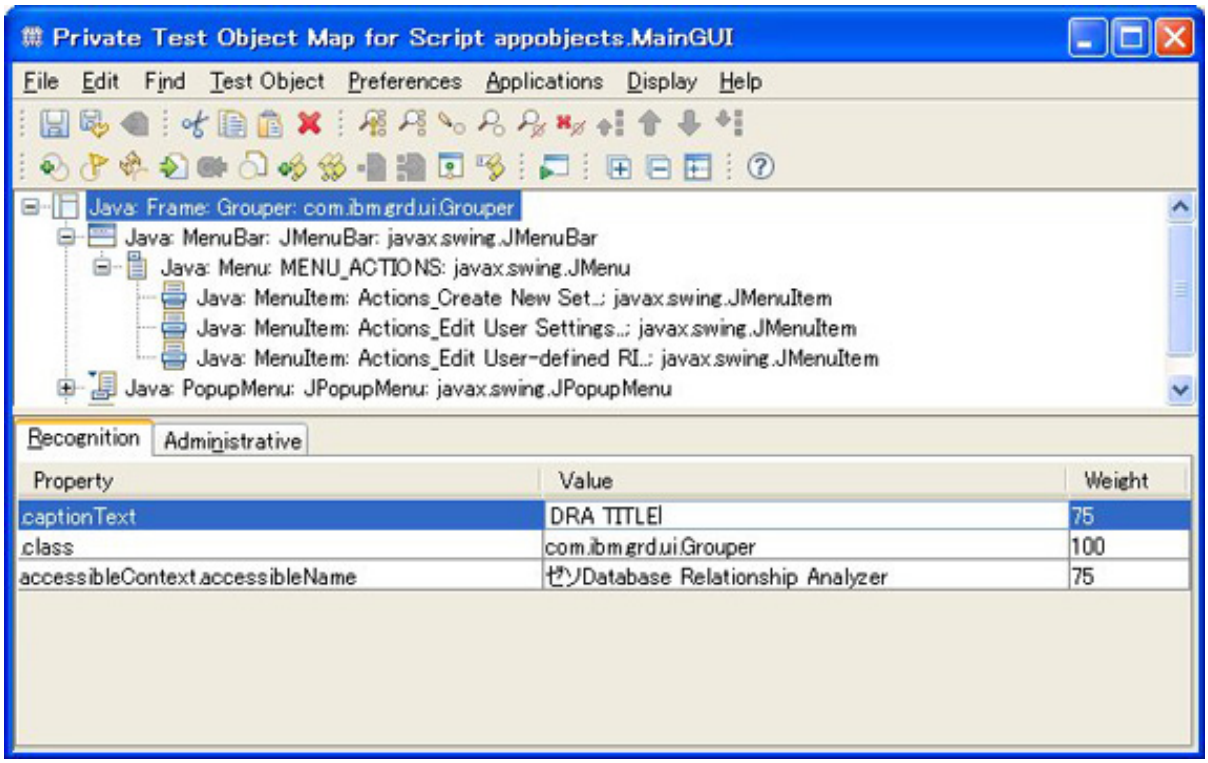
Figure 13. Application GUI for Japanese



Click to enlarge

Open the test object map, and then change the property values that are GUI display values to key names of the resource file, as shown in Figure 14.

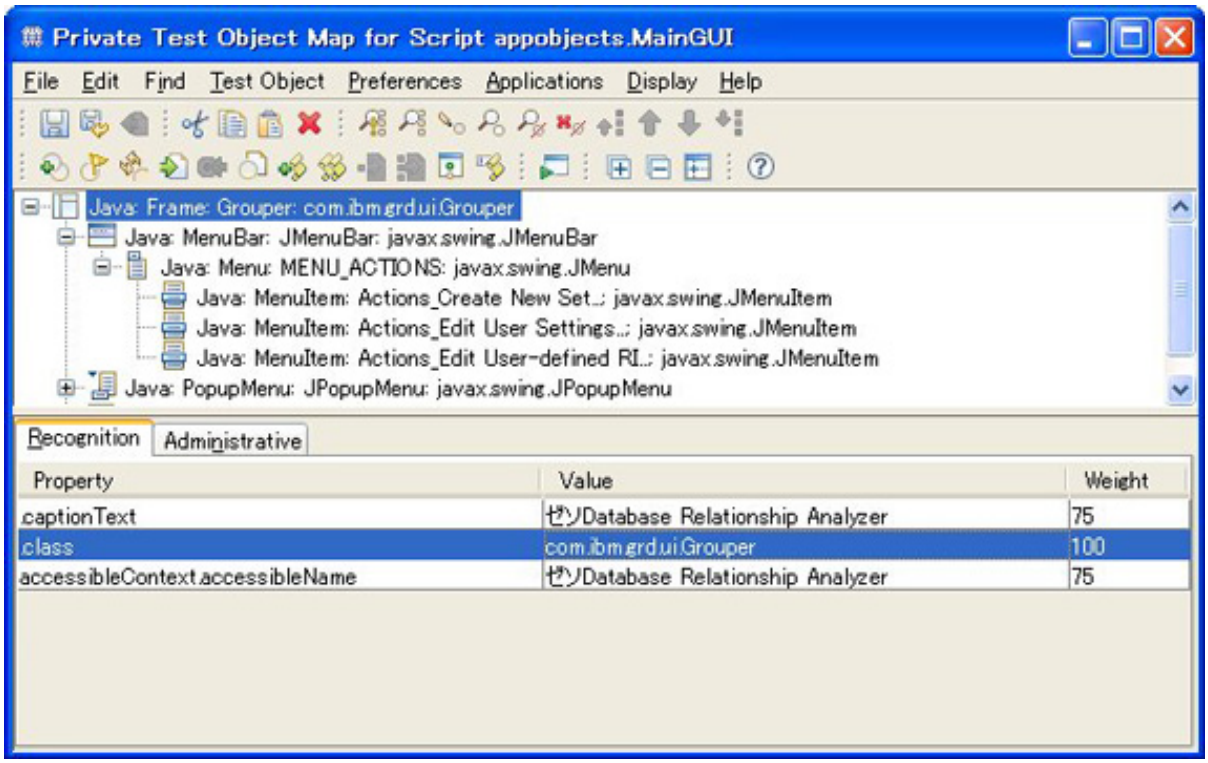
Figure 14. Input a key name of the resource file in the test object map



Click to enlarge

When you press the Enter key, the key name is changed to the actual value in the appropriate resource file, as shown in Figure 15. The key name is stored inside, and the value is dynamically changed based on the language setting. When you double-click the property value to switch to edit mode again, you can see the key name of the resource file. When you change some values of the resource file, and then restart Rational Functional Tester, you can see changed values in the test object map.

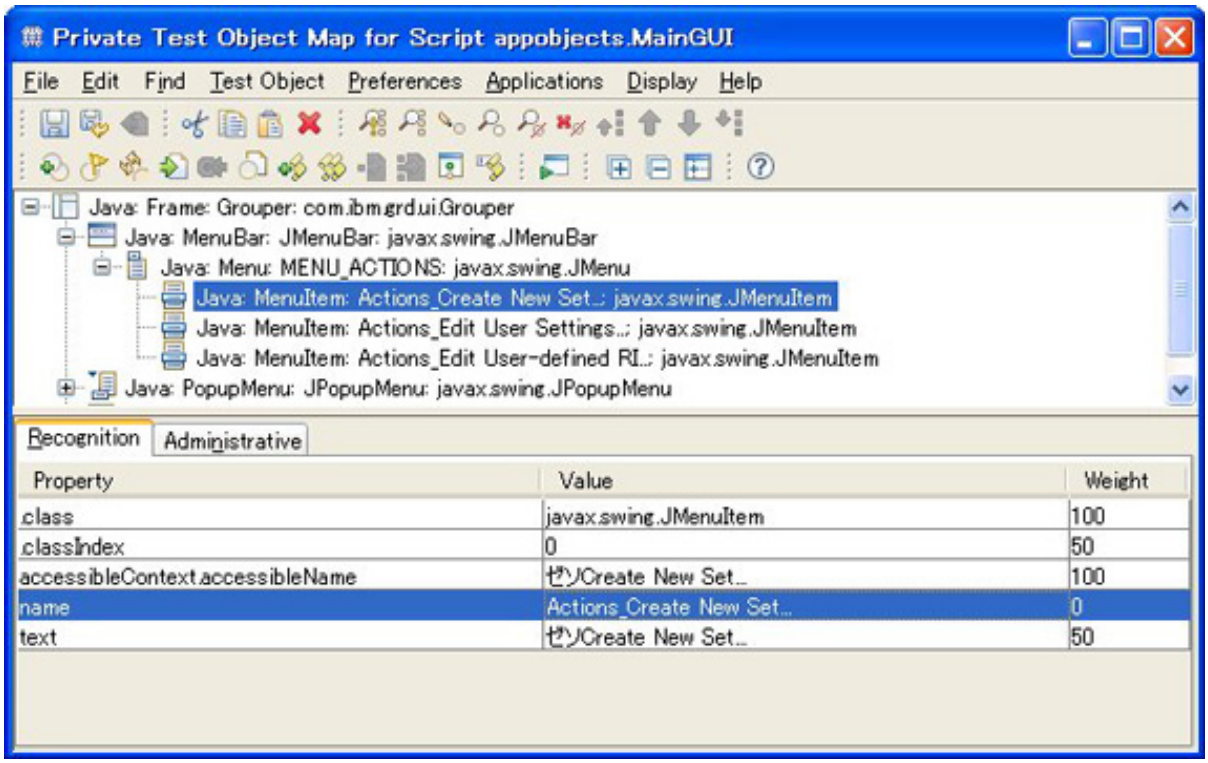
Figure 15. Automatically change the key name to the value in the resource file



Click to enlarge

The following `name` property has its value concatenated with the parent menu name, separated by an underscore ("_"). So its weight is changed to 0 (Figure 16).

Figure 16. Change the weight of a property for finding a GUI object



Click to enlarge

What you have learned

This article described what to do if you don't know which method is used to manipulate the GUI, and how to effectively use the test datapool and the verification point. When you don't know which method is used to manipulate the GUI, simulate the GUI action while recording. Next, refer to the generated program, and provide sub-classes (with understandable method names) as necessary. This allows them to be easily used from other programs. Using the test datapool and the verification point also enables you to input data and verify the test results.

Rational Functional Tester also provides a feature for applications that support multiple languages. This article explained some tips about that feature. Rational Functional Tester can use resource files for multiple languages that contain key and value pairs for GUI display values, so that you can test applications that support multiple languages as well.

Resources

Learn

- Explore the Rational Functional Tester Information Center, where you can also take a short video tour.
- Investigate Rational Functional Tester Plus, which is a software application testing bundle.
- Visit the Rational Functional Tester area on developerWorks for introductory to in-depth information.
- An Object-Oriented framework for IBM Rational Functional Tester.
- Using IBM Rational Functional Tester to automate testing of globalized applications.
- Browse the technology bookstore for books on these and other technicaltopics.
- Visit the Rational software area on developerWorks for technical resources and best practices for Rational Software Delivery Platform products.
- Subscribe to the IBM developerWorks newsletter, a weekly update on the best of developerWorks tutorials, articles, downloads, community activities, webcasts and events.
- 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 newsletter for articles on the concepts behind effective software development.

Get products and technologies

- Try Rational Functional Tester. The trial download is free.
- Download trial versions of IBM Rational software.

Discuss

- Get involved in the developerWorks Functional and GUI Testing discussion forum where you can discuss and ask questions about Rational Functional Tester and general testing topics.

About the authors

Masahiro Ohkawa
Masahiro Ohkawa is a member of the database tooling development team in the Yamato Software Development Laboratory (YSL), IBM Japan.

Gou Nakashima
Gou Nakashima is a member of the database tooling development team in the Yamato Software Development Laboratory (YSL), IBM Japan.

Trademarks