IBM Rational Functional Tester
Learning Guide

Index

1) INTRODUCTION

Need For Automation

Speed - Automation scripts run very fast when compared to human users.

Reliable - Tests perform precisely the same operations each time they are run, thereby eliminating human error.

Repeatable - We can test how the application reacts after repeated execution of the same operations.

Programmable - We can program sophisticated tests that bring out hidden information.

Comprehensive - We can build a suite of tests that covers every feature in our application.

Reusable - We can reuse tests on different versions of an application, even if the user interface changes.

Automation Rules

Testing tools are usually very expensive. The Test Manger or the Test lead has to think twice before going in for automating his test effort. There are also some widely held false notions or beliefs about automation tools. In general, the golden rules that should be kept in mind in the regard are:

1. Product feature understanding, test planning, test case documentation, test bed setup, defect tracking, progression tests are all done in manual mode
2. Test automation success depends on robustness of the test cases not on the test tool.
3. Every manual step has commands in test script
4. If it is not worth, DO NOT automate.
5. Do not build application logic in your test script
6. Always have a common known base state for test cases
7. One test script must address one test case

Rational Functional Tester Features

The following are the key features of RFT.

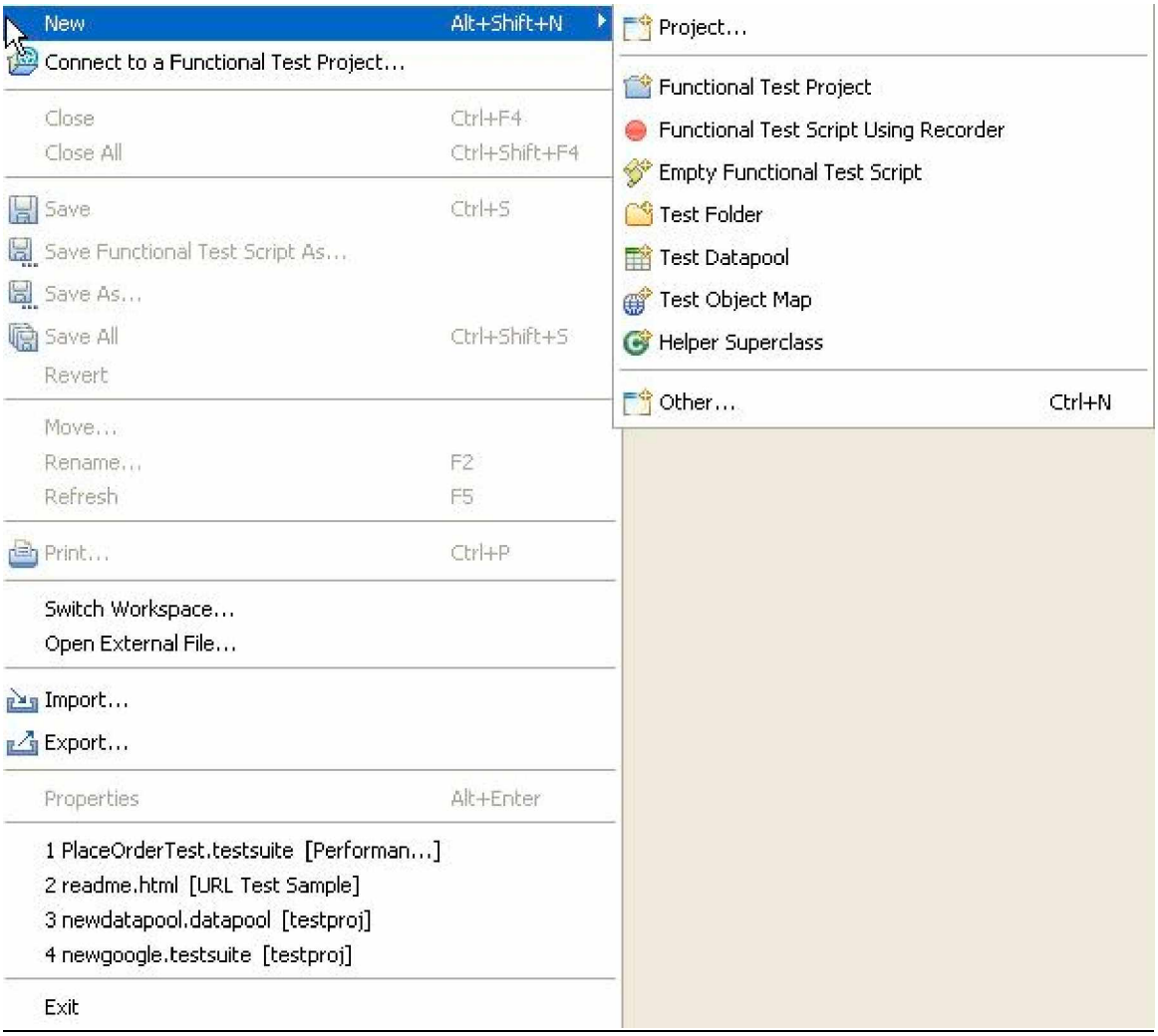
Use Add-ins to support multiple environments

- Projects
- Test Object Inspector
- Test Object Map
- Recording test scripts
- Replay test scripts
- Debug scripts
- Java Scripting
- Create Verification points – GUI, Bitmap, Menu
- Databases sample scripts
- Data Pool the test cases
- Suite (Batch Run)

2) RFT MENU ITEMS

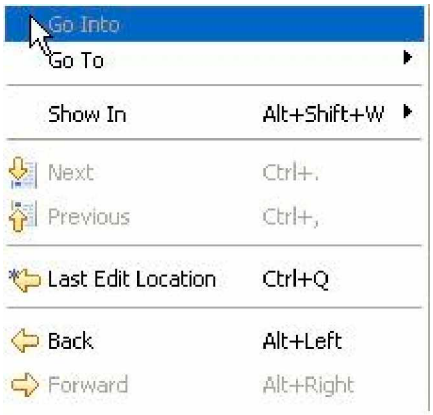
RFT has the following top level menus: File, Edit, Navigate, Search, Project, Script, Configure, Run, Window, Help

File Menu Items



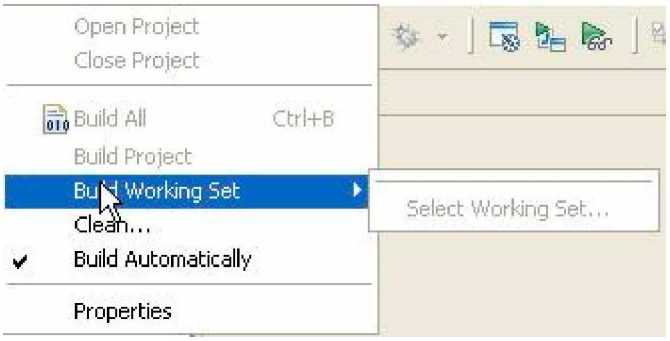
Edit Menu Items

Navigate Menu Items



Search Menu Items

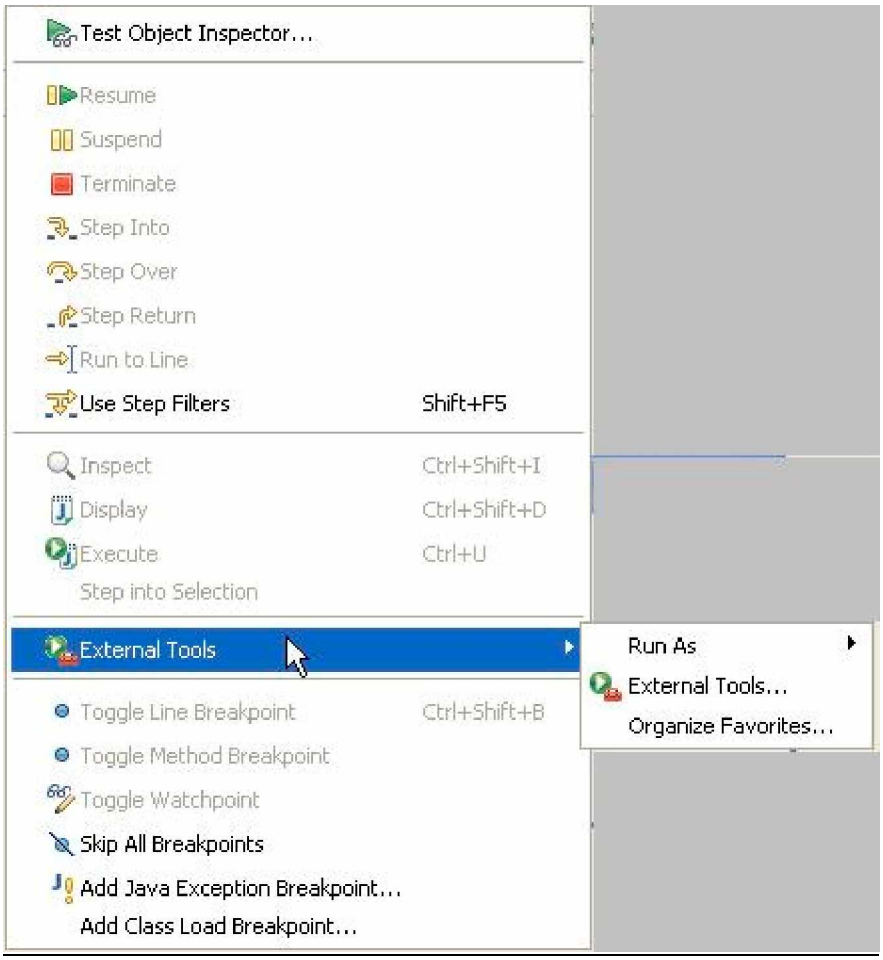
Project Menu Items



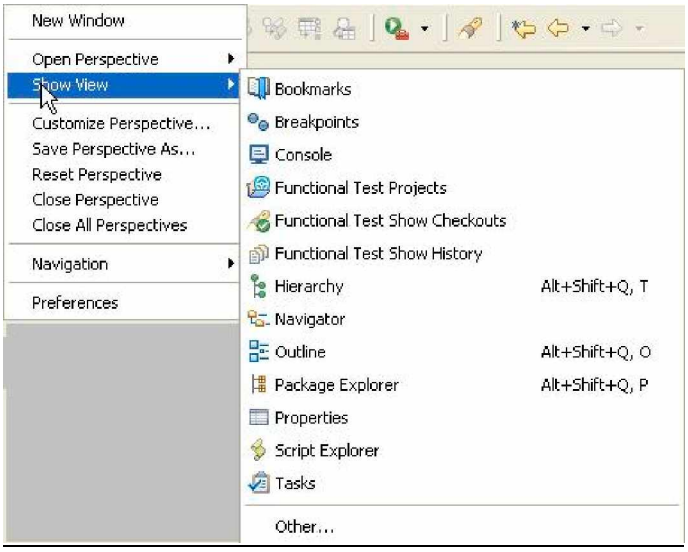
Script Menu Items

Configure Menu Items

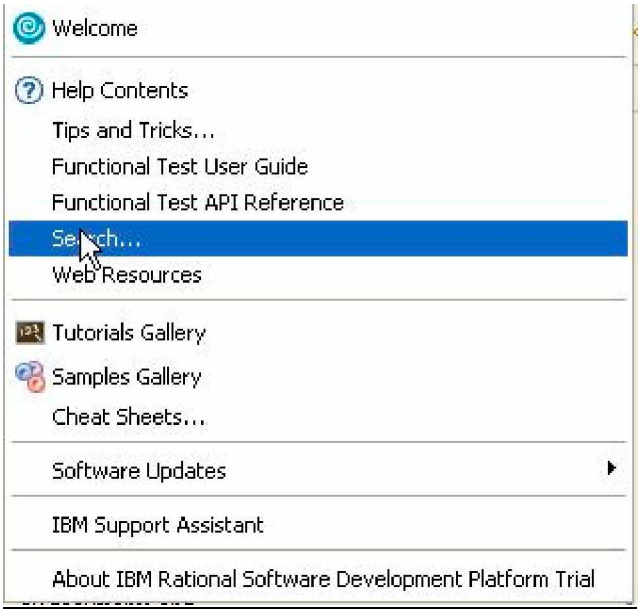
Run Menu Items



Window Menu Items



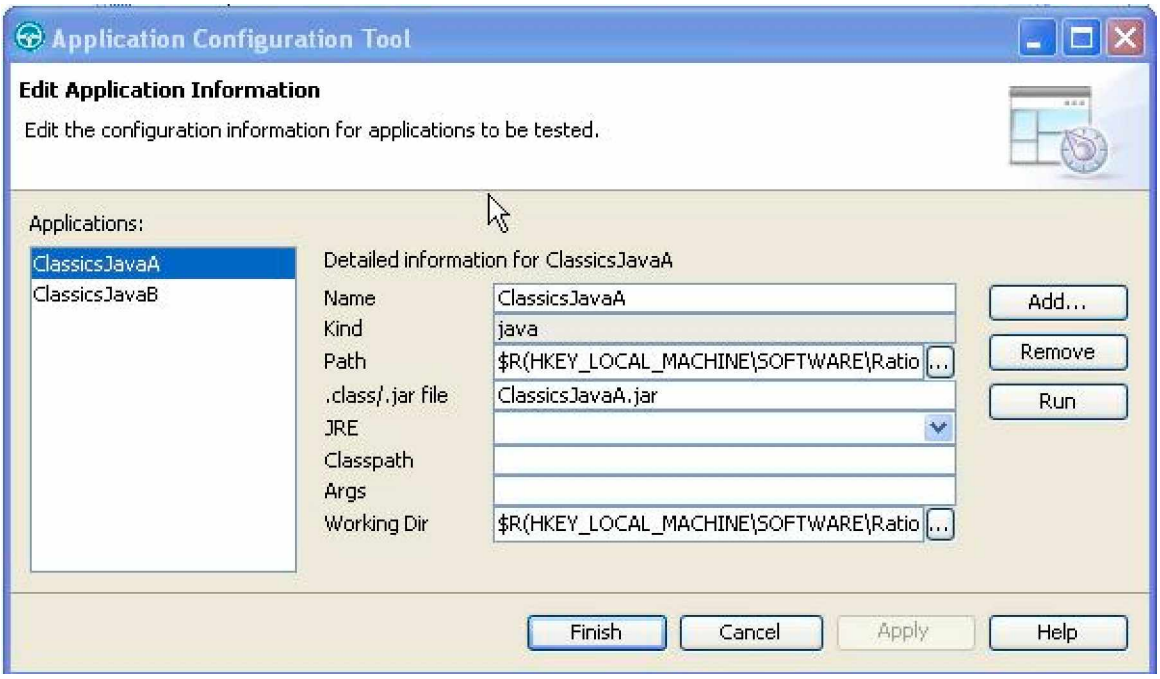
Help Menu Items



Introduction to Rational Functional Tester

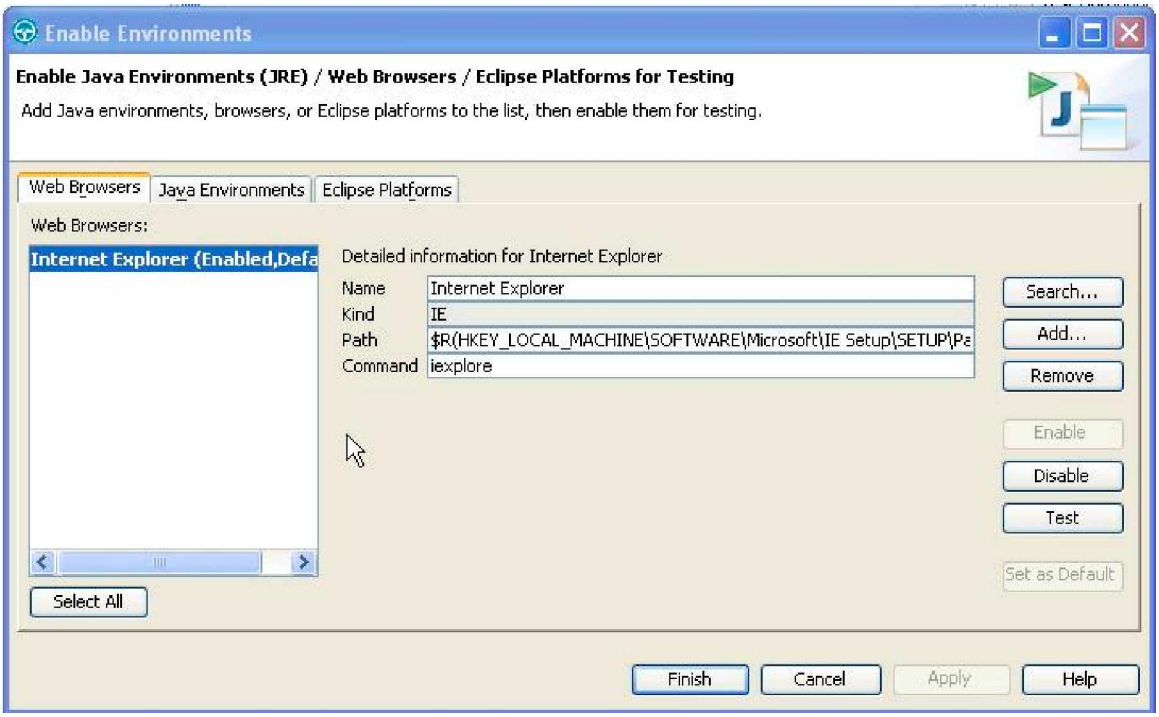
In RFT, Project needs to be created on which multiple scripts can be generated with record reply or with scripting. Rational supports java and vb scripting, Rational can identify windows based objects only on installing .net frame work.

Before staring of Rational Functional tester, the test application needs to be configured for easy handling. This can be done using Configure tab. This will enable Functional tester to start the application to test during the recording event. Also this makes the functional tester more reliable.



Before testing any web based applications (HTML), care should be taken to invoke the appropriate browsers. This can be done using Configure->enable environment. By default, while

installing IE is enabled, In case of Netscape, Mozilla, these browsers needs to be enabled using the enable function provided in the enable environment.



Test Object Inspector

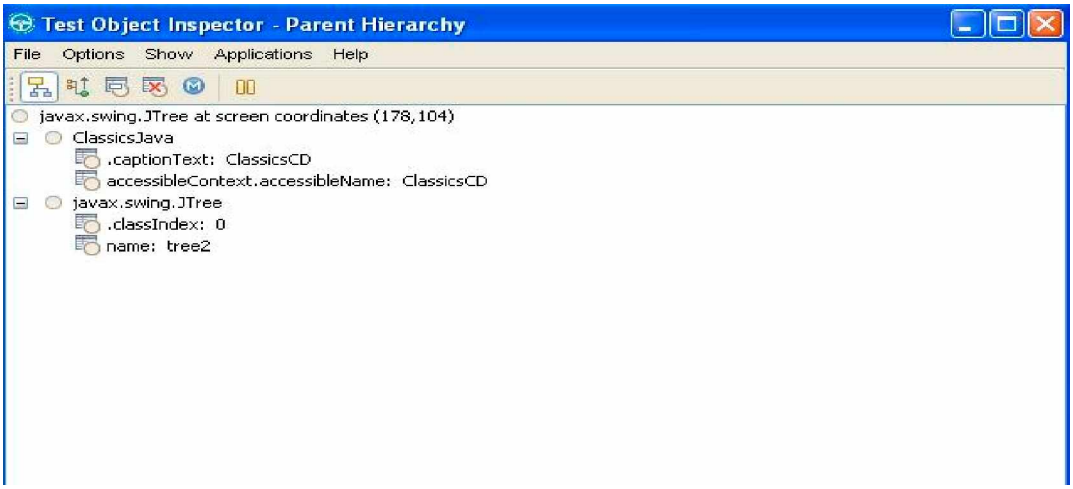
Test object inspector helps to view the properties of the test objects, It displays the information such as parent hierarchy, inheritance hierarchy, test object properties, non value properties, and method that can be used for those objects.

Windows objects can be viewed in the test object inspector on installation .net frame work version 1.1.

Default it works well with all java applications.

Before doing any automation activities, spend 3-4 hours to inspect every object in the application to know better about the object properties. We can find out both visible and invisible properties.

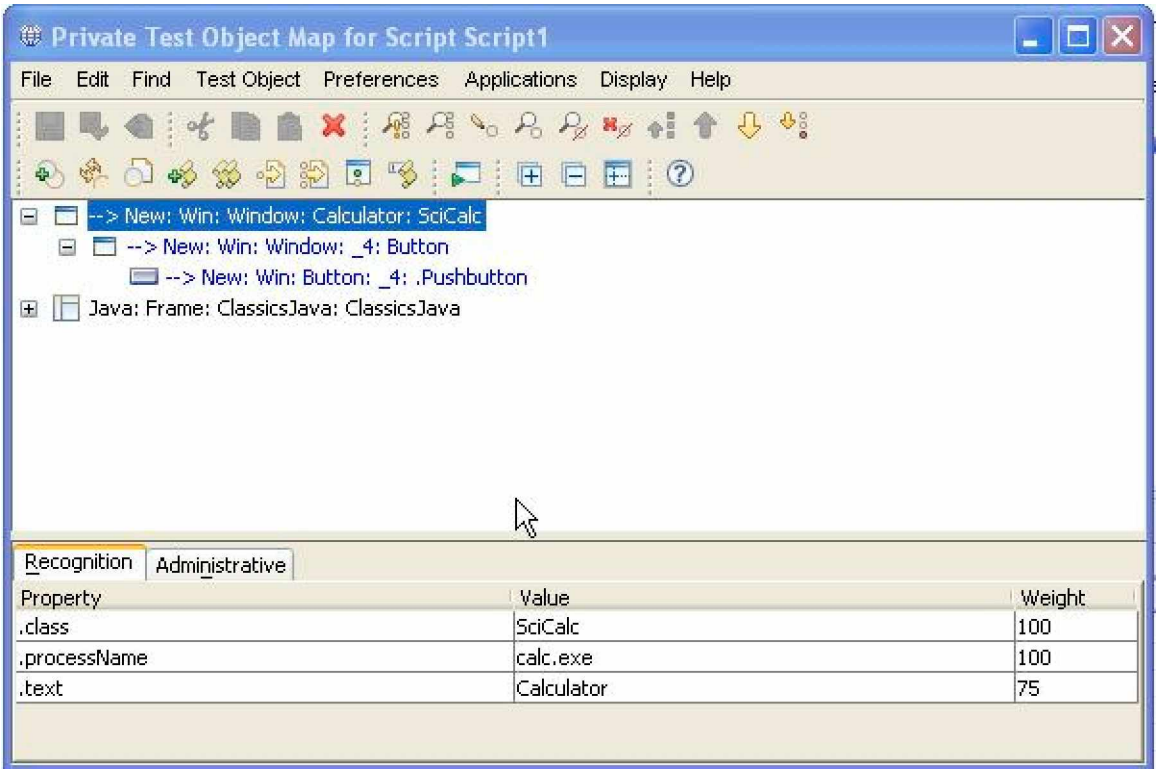
Test Object Inspector can be invoked from Run-Test Object Inspector



Test Object Map

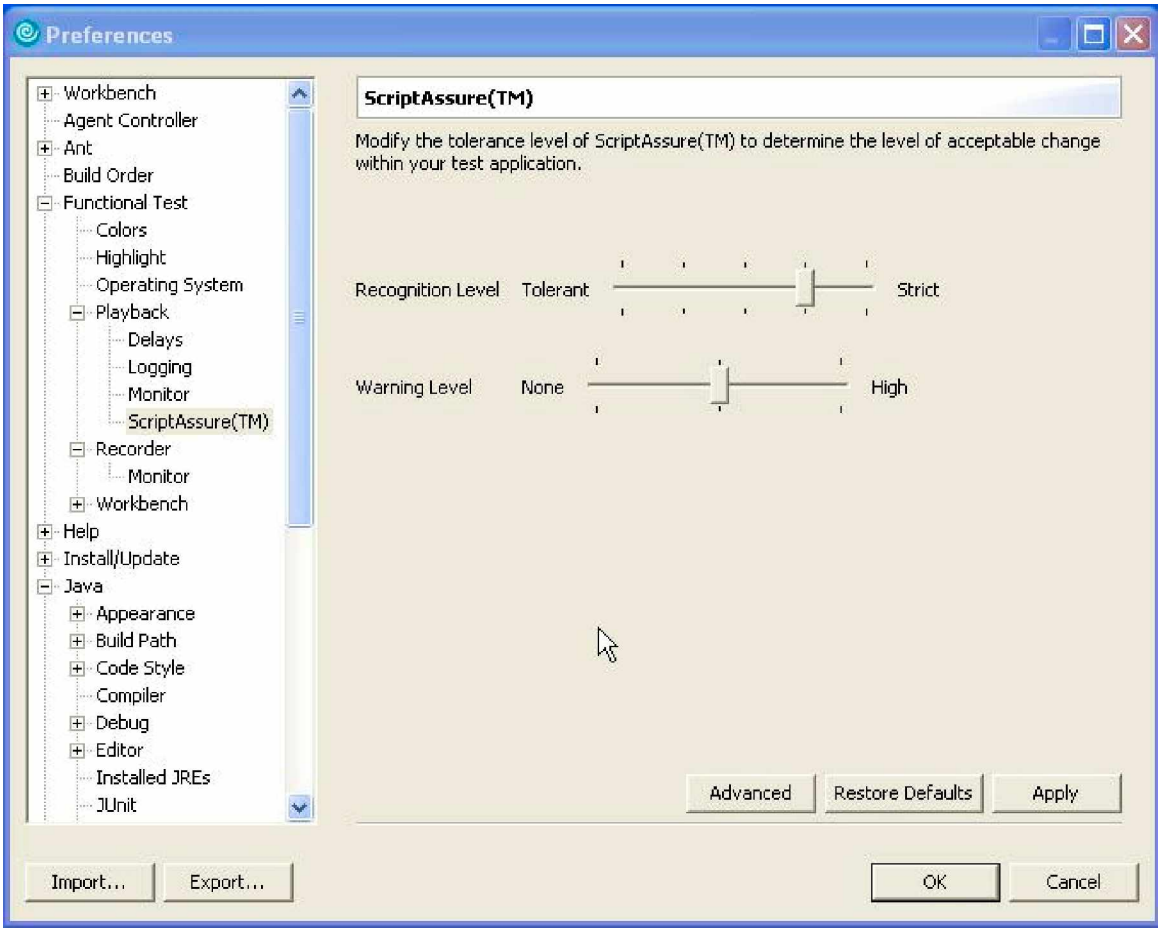
While recording the script, Functional tester creates a test Object Map. It can private, associated with only the script or shared between scripts. Merging of multiple Test object map is also possible.

In a **recording** session: Rational Functional Tester generates script code into a script file. The recognized mappings of logical to physical names of classes and objects in the application Rational Functional Tester stores in a **Test object map** file that Rational Functional Tester uses later when it runs scripts to control the application under test.



Objects to be recognized can be added directly to the existing Test object map using the menu available in the Test Object Map window.

Recognition tab contains the recognition data used by Functional test. Recognition properties are those which are established during recording. These are the properties used by the Functional tester to recognize the objects.



For Functional Tester to recognize an object in the application under test, the object properties must match the properties recorded in the test object map. By default, Functional Tester can find the object if one or two properties do not match.

If objects in the application under test have changed, you can still play back scripts in Functional Tester by using the Script Assure Feature to control object-matching sensitivity.

Object recognition during recording is controlled by the Test Object Map.

3) Recording

Recording is used to record test pre-requisites and test steps.

- Commands are based on GUI objects
- Test Object map is the key repository for CS commands to work
- All commands will appear as object.action()
- Example: object.click(), object.select("Item name")
- Every test step must have a corresponding command while recording
- Most used objects are: window, menu, toolbar, edit box, list box, check button, radio button, push button, tab, grid, scroll bar

Context sensitive command Types

- **Action commands** are recorded – they trigger events on GUI objects. These are the ones that are generated while recording example object.click()
- **Get commands** provide some information about the GUI objects and they return results in a variable. Commands will be like object.getProperty("Propertyname")
- **Verification point** commands verify certain attributes of objects. Commands will be like objectVP.performtest()
- **WaitforExistance** commands make RFT to wait for certain objects to appear. Commands will be like object.waitforexistance("ObjectName")

Sample Record Script

```
import resources.order1Helper;

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.siebel.*;
import com.rational.test.ft.script.*;
import com.rational.test.ft.value.*;
import com.rational.test.ft.vp.*;

/**
 * Description   : Functional Test Script
 * @author Administrator
 */
public class order1 extends order1Helper
{
	/**
	 * Script Name   : <b>order1</b>
	 * Generated     : <b>Apr 6, 2007 10:14:44 AM</b>
	 * Description   : Functional Test Script
	 * Original Host : WinNT Version 5.0  Build 2195 (S)
	 *
	 * @since  2007/04/06
	 * @author Administrator
	 */
	public void testMain(Object[] args)
	{

		// Frame: ClassicsCD
		jmb().click(atPath("Order"));
		jmb().click(atPath("Order->Place New Order..."));

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

		// Frame: Place an Order
		cardNumberIncludeTheSpacesText().click(atPoint(24,11));
		placeAnOrder().inputChars("1234");
		nameText().click(atPoint(27,13));
		placeAnOrder().inputChars("abc");
		phoneText().click(atPoint(17,12));
		placeAnOrder().inputChars("1234");
		placeOrder().click();

		//
		ok2().click();
```

```
            // Frame: Place an Order
            streetName().click(atPoint(34,10));
            placeAnOrder().inputChars("mint st");
            placeOrder().click();

            //
            ok2().click();

            // Frame: Place an Order
            cityStateZipText().click(atPoint(18,10));
            placeAnOrder().inputChars("chn,tn,18");
            placeOrder().drag();

            //
            ok2().click();

            // Frame: Place an Order
            cancel().click();



            // Frame: ClassicsCD
            jmb().click(atPath("Admin"));
            jmb().click(atPath("Admin->Products..."));

            // Frame: Administration
            rememberPassword().clickToState(SELECTED);
            cancel2().click();

            // Frame: ClassicsCD
            jmb().click(atPath("Help"));
            jmb().drag(atPath("Help->About Classics Java"));

            // Frame: About ClassicsCD
            okHelp().click();
        }
    }
```

4) Replay

- v One can replay all CS statements from top
- v We can set the start position for execution using arrow mark and start execution from that point
- v Step by step execution is possible
- v Setting break points and executing is possible
- v Fixed delays can be introduced between execution of CS statements

5) Simple Java Scripting

Coding is used to enable better testing and not to show the programming skills of a tester. The end goal is to have the product with zero defects and not a test script with zero defects.

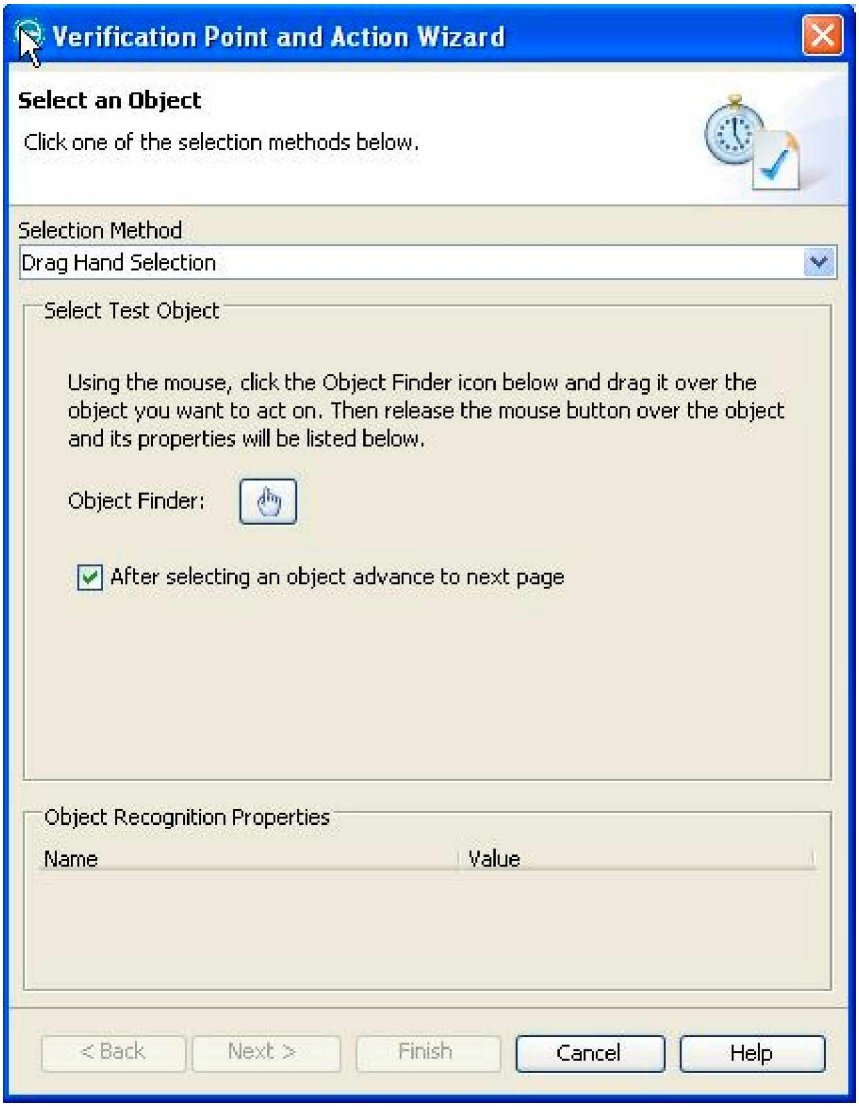
- Variables cannot be used as variants – need to declare
- Variable scopes are auto, public
- Arrays can be defined and grown/shrunk dynamically
- Different elements of array can have different types of values
- If-then-else is used as control structures
- For, do-while and while-do statements are used as loops
- Built-in arithmetic functions like sqrt, log, exp are available
- Built-in string functions like length, toupper, tolower, substr are available
- Comments are prefixed with /* ….*/ , // symbol

```
import resources.ScriptsHelper;
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.*;

/**
 * Description   : Functional Test Script
 * @author User
 */
public class Scripts extends ScriptsHelper
{
	/**
	 * Script Name   : <b>Variables</b>
	 * Generated     : <b>Jun 12, 2007 3:22:35 PM</b>
	 * Description   : Functional Test Script
	 * Original Host : WinNT Version 5.1  Build 2600 (S)
	 *
	 * @since  2007/06/12
	 * @author User
	 */
	 int add(int a2,int b2)// method declaration
				// int is the return type of the method

		{
		 return  a2+b2;     // method return value


		}


	public void testMain(Object[] args)
	{
		// TODO Insert code here

		String a = "piyush"; //String Constant Declaration
```

```
System.out.println(a); //Printing the value of a varialbe

int b = 20; // Integer Declaration
int c = 30; //Integer Declaration

int d = b+c; // arithmetic operations
int e = b*c;
int f = c/b;
int g = c%b;
System.out.println(d);
System.out.println(e);
System.out.println(f);
System.out.println(g);

String s1="Welcome ";
String s2="To the world of test automation";
String s = s1+s2;// String concatination opeartion
                //'+' used when both variables are Strings
String S="Shree ";
int i = 4;
int j= 20;

String k = String.valueOf(i).concat(String.valueOf(j));
//String.valueOf will convert the variable i to String
//Concat Is the method used to concatinate two strings.


String l = S.concat(k);
System.out.println(l);

// For Loop

for(int a1=0;a1<=10;a1++)
{
        System.out.println("This is "+a1);
}

int b1=1;
while(b1<10) // while Statement
                // Less than operator(logical Operator)
{
        System.out.println("I am in loop "+b1);
        b1+=b1; // Post Increment
}


int p=1;

do
{
        System.out.println("do it");
        p+=p;
```

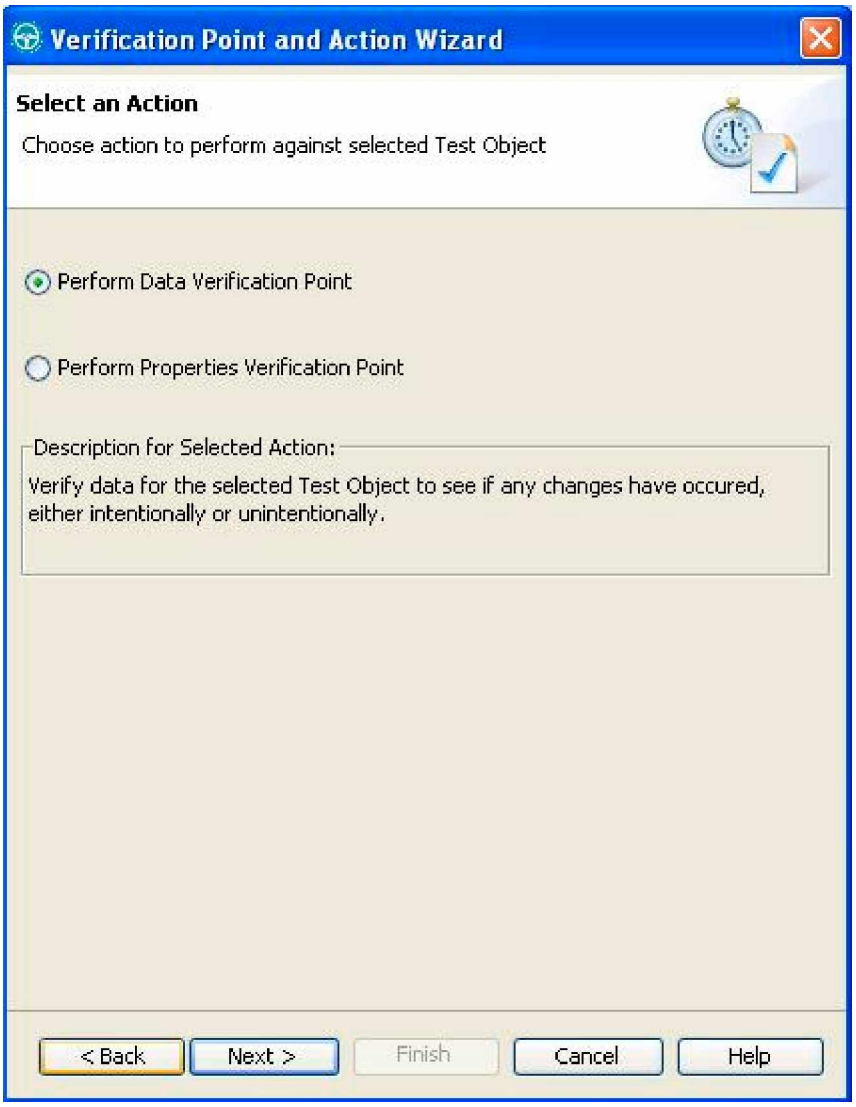
```
        }while(p<10);

    int c1=20;
    int d1=20;

    if(c1==d1)// Equal To operator(logical Operator)
    {
        System.out.println("All are equal");
    }

    else
    {
        System.out.println("All are not equal");
    }


   int val=add(2,3);//method calling
   System.out.println(val);

   String Testtool = "Rational Functinal Tester";
      char[] mfl = Testtool.toCharArray();
      for (int pos = 0; pos < mfl.length; pos++)
      {
       char current = mfl[pos];
        if (current != ' ')
        {
          System.out.print(current);
        }
        else {
System.out.print('.');
        }
      }
    }




 }
```

6) Verification Points



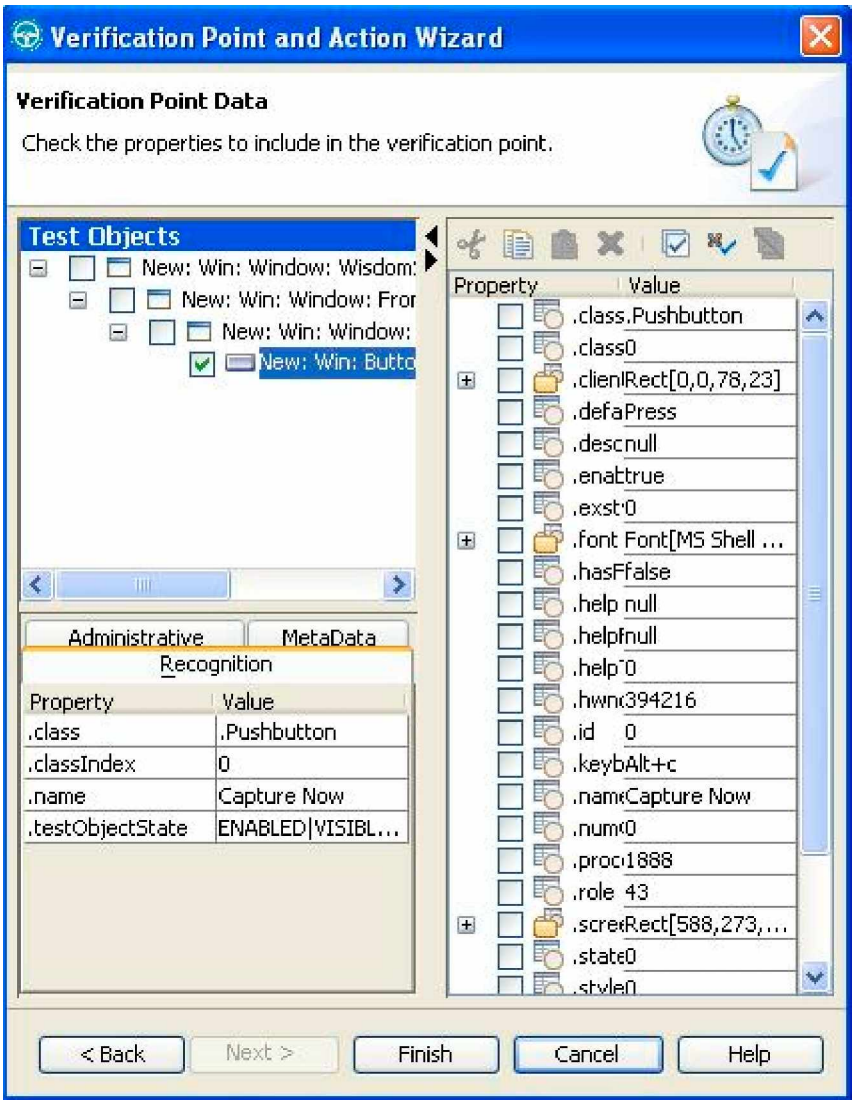
7) Properties Verification Points

This is used to test the properties of an object in the application. When we create this verification point we can use the data pool value also to check variable data.



Data Verification Point

- **Data Verification Point (List):** To test the list data in the application. When recorded, a baseline of the list data is created and that will be compared with run time data in the list box.
- **Data Verification Point (Menu Hierarchy):** To the data in the menu hierarchy of application. When recorded, a baseline of the menu hierarchy is created and that will be compared with run time menu data.
- **Data Verification Point (State):** To test the state of the radio button or check box in the application. When recorded, a baseline of the state data is created and that will be compared with run time value for that object.
- **Data Verification Point (Table):** To test the table contents or selected table cells in the application. Table content test the entire contents of the table. Selected will test only the cells selected.
- **Data Verification Point (Text):**To test the textual data in the application. For example, to test text in a text area, or the text on an object. When recorded, a baseline of the textual data will be created.
- **Data Verification Point (Tree Hierarchy):** To test the tree hierarchy in the application. Important point - One of the values listed on the object recognition properties grid should be ‘tree’ . We can test the entire tree hierarchy or selected tree hierarchy.



8) Database Scripts

v DSN is required for database checkpoint to work

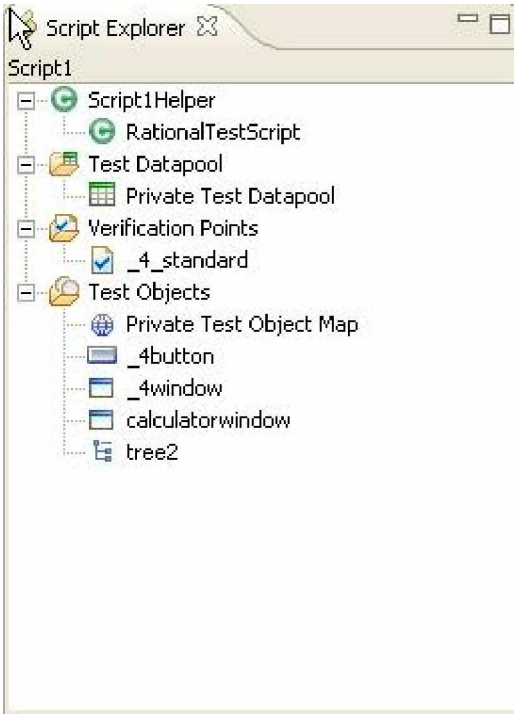
Sample Script for Database

```
import resources.databaseHelper;

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.siebel.*;
import com.rational.test.ft.script.*;
import com.rational.test.ft.value.*;
import com.rational.test.ft.vp.*;
import java.sql.*;

/**
 * Description   : Functional Test Script
```

```
 * @author Administrator
 */
public class database extends databaseHelper
{
	/**
	 * Script Name   : <b>database</b>
	 * Generated     : <b>Apr 6, 2007 12:35:38 PM</b>
	 * Description   : Functional Test Script
	 * Original Host : WinNT Version 5.0  Build 2195 (S)
	 *
	 * @since  2007/04/06
	 * @author Administrator
	 */
	public void testMain(Object[] args)
	{
		Connection con; // The connection to the database.
//		 The following code can throw errors, so they must be caught.

		try{

//		 First, tell Java what driver to use and where to find it.
		Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

//		 Next, create a connection to your data source.
//		 Specify that you are using the ODBC-JDBC Bridge.
//		 And specify the data source from ODBC.
		con = DriverManager.getConnection("jdbc:odbc:QT_Flight32");
//		 Create an SQL statement.
		Statement stmt = con.createStatement();

//		 Execute some SQL to create a table in your database.
//		 If the table already exists, an exception is thrown!
		//stmt.executeUpdate("CREATE TABLE COFFEES " +
		//"(COF_NAME VARCHAR(32), SUP_ID INTEGER, PRICE FLOAT, " +
		//"SALES INTEGER, TOTAL INTEGER)");

		ResultSet rs = stmt.executeQuery("select * from orders");

		while (rs.next())
		{
			String s = rs.getString("Customer_Name");
			int i = rs.getInt("Flight_Number");
			int o = rs.getInt("Order_Number");
			System.out.println(o + " : " + s + " : " + i);

		}

		con.close();

		}
//		 Catch any exceptions that are thrown.
		catch(ClassNotFoundException e){

			System.out.println(e.toString());
		}
		catch(SQLException e){

			System.out.println(e.toString());
		}

	}
}
```

9)Data Pool Tests

- Population of master data or running the same test using multiple data would require this feature
- Record the test steps for one transaction
- Associate the Data table using script explorer and right click on the test Data Pool and click on associate with data pool
- Create the data table with different sets of data to be used for the same script or function
- Use 'Find literals and Replace with Datapool reference' command from Script menu to map screen fields and data table fields
- Rational Functional Tester will create code section to drive the script using multiple data
- Commands are: dpString, dpBoolean, dpByte, dbChar,dbInt, etc.



Suite of Scripts

You can use call script method for calling multiple scripts from a script

```
Public class RegreessionSuite extends RegressionSuiteHelper
{
Public void testmain (object[] args)
{
        CallScript ("test1");
        Callscript ("test2");
        Callscript ("test3");
}
}
```

Note: Do not call the testMain method from another functional test Script

10) Coding Standards

1. All variables will follow Pascal Casing.
2. All functions will follow Pascal Casing.
3. Public variables will have a prefix "g". (gFirstName)
4. Extern variables will have a prefix "e". (eLastName)
5. Auto and Static variables will not have any prefixes. (Age, DoorNum)
6. Variables containing integer values will have a prefix "i". (iAge)
7. Variables containing real values will have a prefix "f". (fVelocity)
8. Variables containing string values will have a prefix "s". (sState)
9. Scope prefix will come first and then the variable type prefix. (giAge)
10. Only one statement will be coded in one line.
11. One tab indentation (4 spaces) will be used for every level of control structure.
12. All functions will have a prefix "fn". (fnGetStateName)
13. All function names must start with a verb.
14. Every function will have a header as follows:

```
#######################################################
# Name: <function name>
# Parameters: <Describe every parameter name and type and usage>
# Return value: <Describe the return value>
# Description: <Describe the logic of the function in brief>
#######################################################
```

15. A space must be given before and after every operator.
16. All constants must be typed in uppercase letters (gTIMEOUT)
17. Between functions, there must be 2 blank lines.

11) Scripting Guidelines

1. The function that drives a test case must have the name as the test case ID.
2. Avoid nesting of control structures more than 2 levels.
3. Whenever a GUI object is used in verification points, its enabled status must be checked.
4. Avoid usage of ActiveX_Set_Info commands.
5. Ensure every file that is opened, is closed soon after the operations are complete.
6. Ensure every DSN that is opened, is closed soon after the operations are complete.
7. The column names for the data pool test .xls file must be in uppercase letters.
8. Avoid hard coding.
9. Every loop must be preceded with comment lines.
10. To the extent possible, use one test object map file for the whole project.
11. Always check the return code of file operation commands.
12. Always check the return code of database operation commands.

13. DO NOT IMPLEMENT APPLICATION LOGIC INSIDE CODE, to calculate the expected results. Manually find out the expected results in the test cases and use verification points directly.