Open2Test Test Automation Framework for SilkTest - Quick Start Guide

Version 1.0

January 2010

TABLE OF CONTENTS

1 Purpose of the Document

This document drives the settings and describes how to get started with keyword-driven scripting in SilkTest.

2 Enabling Extension

Enabling Extensions is performed to identify/recognize the objects in Application Under Test (AUT) by SilkTest.

An extension is a file that serves to extend the capabilities of, or data available to, a more basic program. SilkTest provides extensions for testing applications that use non-standard controls in specific development and browser environments.

Extensions can be enabled using any of the following methods:

2.1 Using Basic Workflowbar

1. To enable the extensions, launch the Application Under Test (AUT), then click on "Enable Extensions" in the "Basic Workflowbar" of SilkTest.

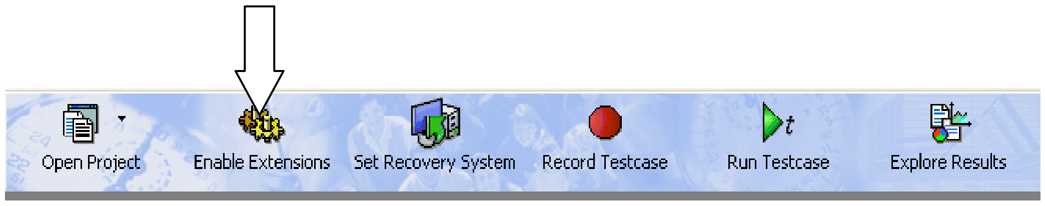
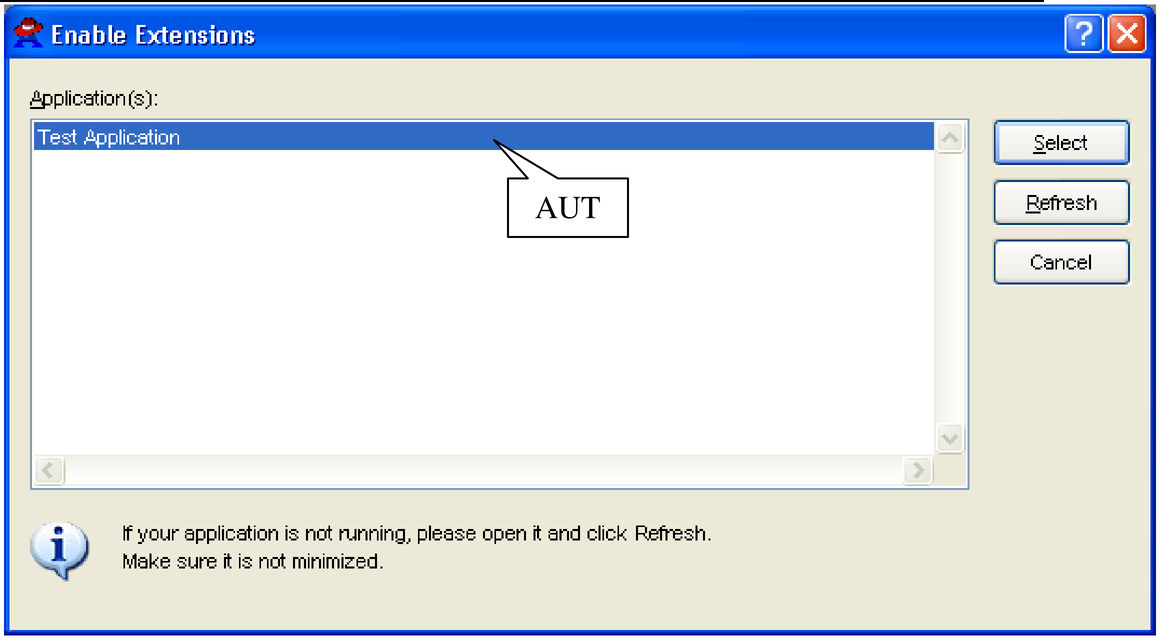


Figure 1Basic Workflowbar

2. Enable Extensions dialog is displayed with a list of all the opened applications in the "Application(s)" list.



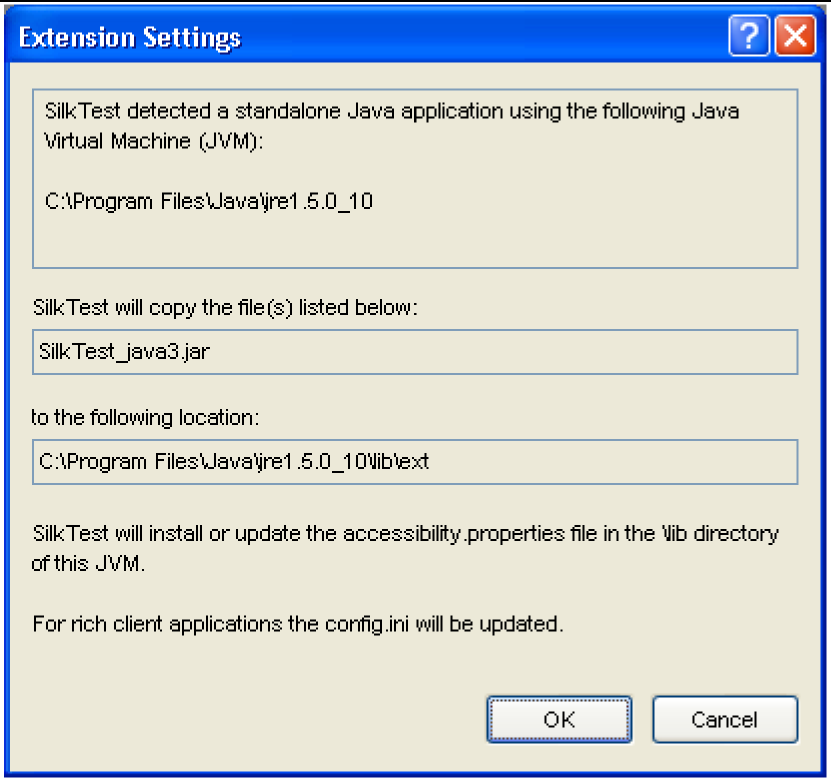
If your application is not running, please open it and click Refresh.
Make sure it is not minimized.

Figure 2Enable Extensions Dialog

Note: If the AUT is minimized or not visible in the application(s) section, maximize the AUT and click on the "Refresh" button in the Enable Extensions dialog.

3. Select the required application from the application(s) list, then click on the "Select" button.

4. Click on the "OK" button in the Extension Settings dialog.



SilkTest detected a standalone Java application using the following Java Virtual Machine (JVM):

C:\Program Files\Java\jre1.5.0_10

SilkTest will copy the file(s) listed below:

SilkTest_java3.jar

to the following location:

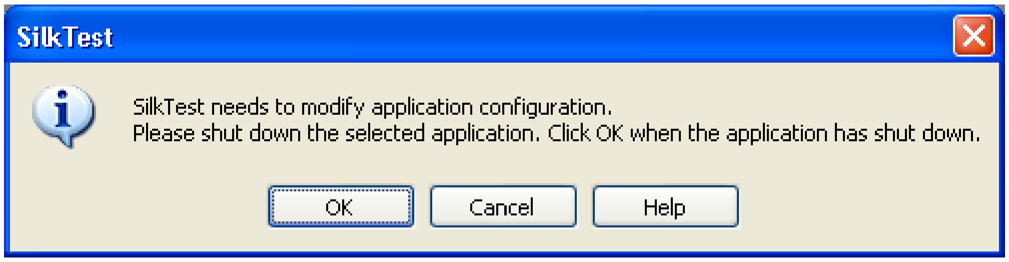
C:\Program Files\Java\jre1.5.0_10\lib\ext

SilkTest will install or update the accessibility.properties file in the \lib directory of this JVM.

For rich client applications the config.ini will be updated.

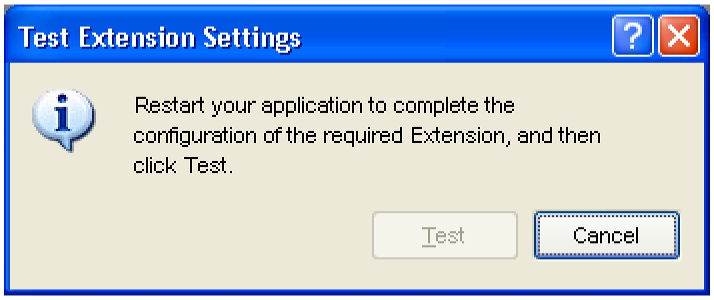
Figure 3Extension settings Dialog

5. Close the application, and then click on the "OK" button.



SilkTest needs to modify application configuration.
Please shut down the selected application. Click OK when the application has shut down.

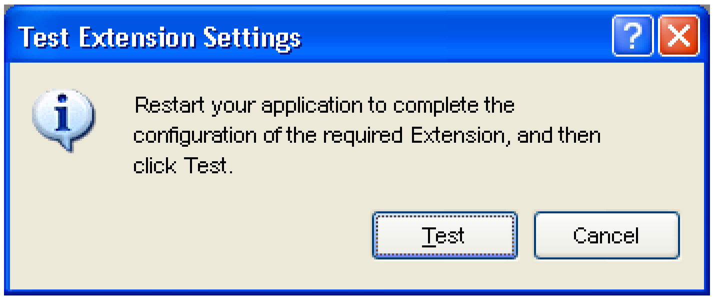
The Test Extension Settings dialog is displayed after closing the application.



Restart your application to complete the configuration of the required Extension, and then click Test.

Note: The "Test" button in the Test Extension Settings dialog gets enabled once the application (AUT) is relaunched.

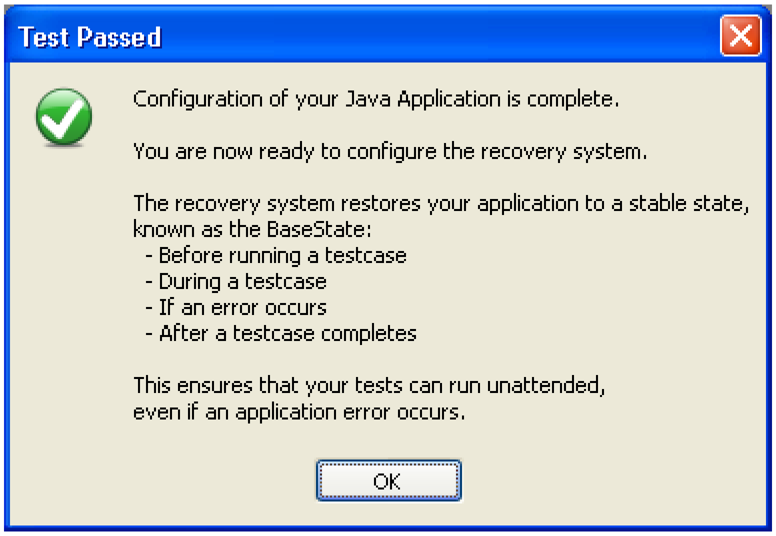
6. Relaunch the application, and then click on the "Test" button.



Test Extension Settings

Restart your application to complete the configuration of the required Extension, and then click Test.

7. Click on the "OK" button in the dialog below once the extensions process is complete.



Test Passed

Configuration of your Java Application is complete.

You are now ready to configure the recovery system.

The recovery system restores your application to a stable state, known as the BaseState:
- Before running a testcase
- During a testcase
- If an error occurs
- After a testcase completes

This ensures that your tests can run unattended, even if an application error occurs.

2.2 Using the Options

1. Select Options → Extensions, as shown below.

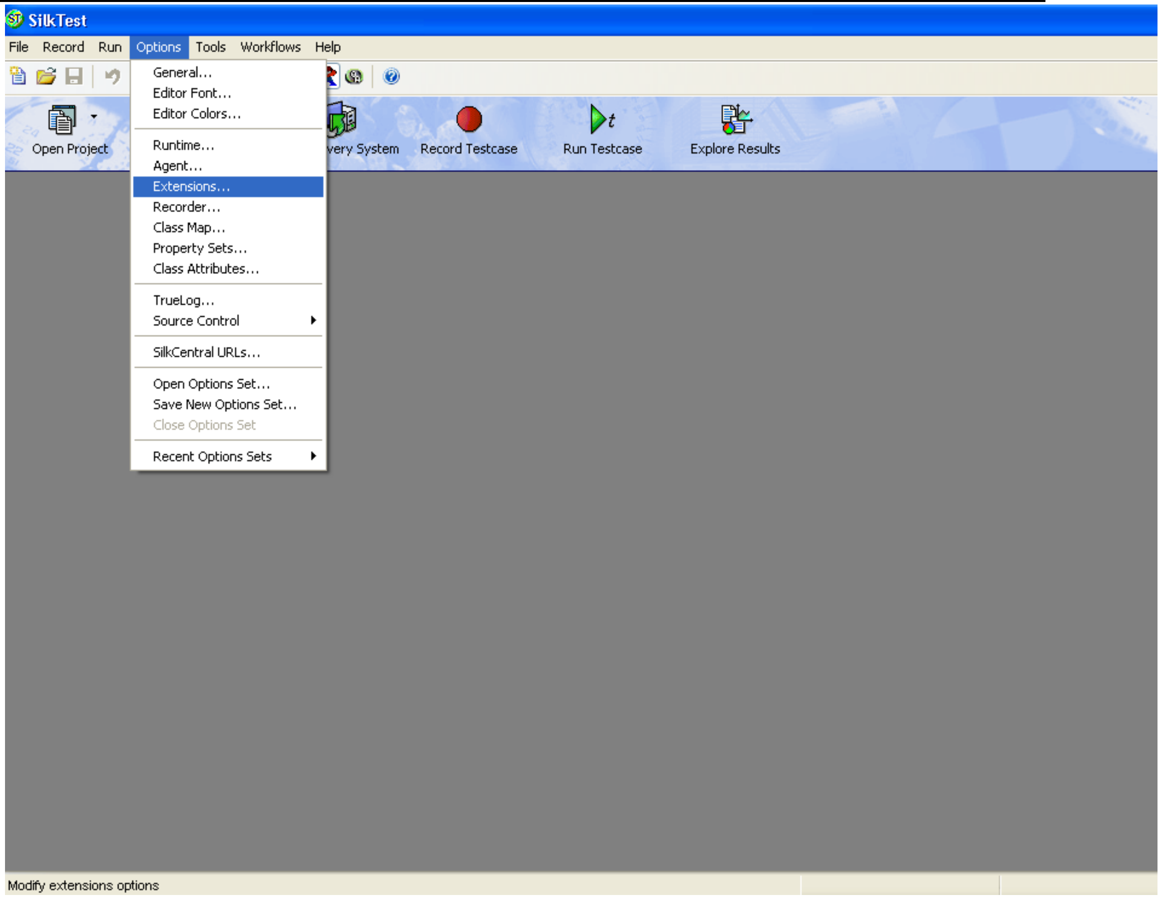
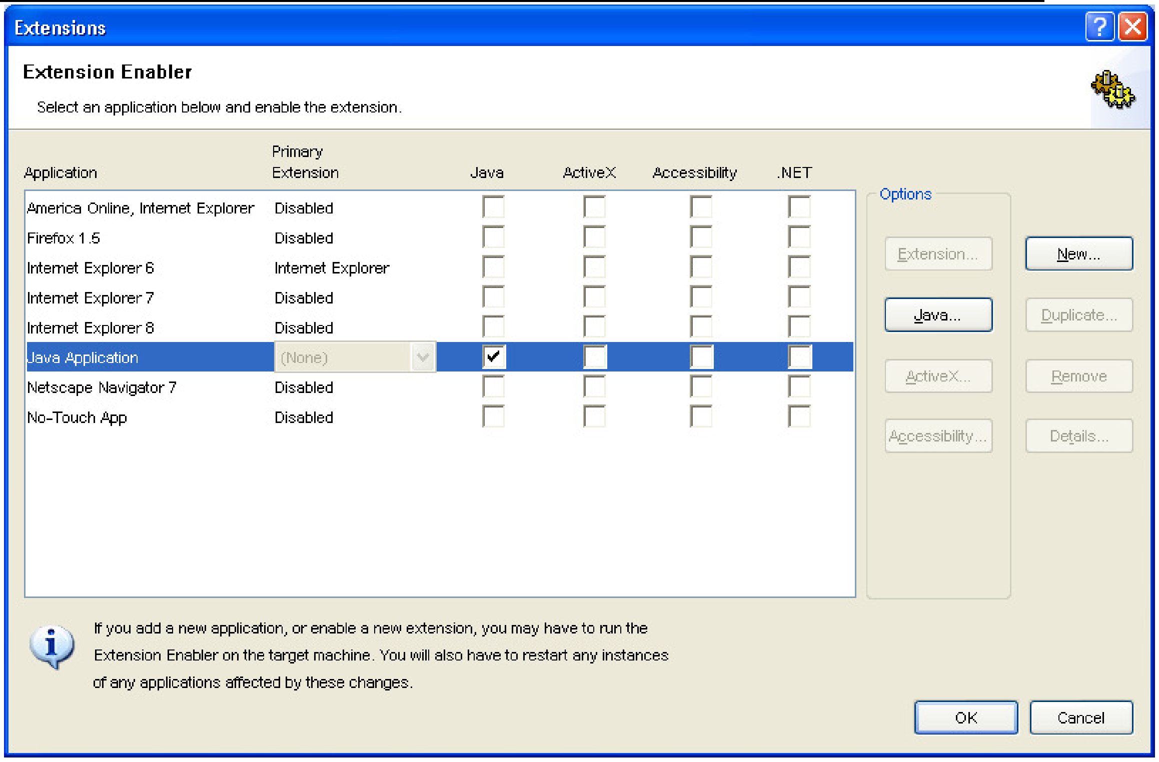


Figure 4SilkTest Host Options Dialog

2. Select the type of the AUT (e.g., Java, .Net, etc) from the "Extension Enabler" dialog, then click on the "OK" button.



Application	Primary Extension	Java	ActiveX	Accessibility	.NET
America Online, Internet Explorer	Disabled	☐	☐	☐	☐
Firefox 1.5	Disabled	☐	☐	☐	☐
Internet Explorer 6	Internet Explorer	☐	☐	☐	☐
Internet Explorer 7	Disabled	☐	☐	☐	☐
Internet Explorer 8	Disabled	☐	☐	☐	☐
Java Application	(None)	✓	☐	☐	☐
Netscape Navigator 7	Disabled	☐	☐	☐	☐
No-Touch App	Disabled	☐	☐	☐	☐

If you add a new application, or enable a new extension, you may have to run the Extension Enabler on the target machine. You will also have to restart any instances of any applications affected by these changes.

Figure 5Extensions Enabler Dialog

Once the enabling extension is completed, the recovery system may need to be set.

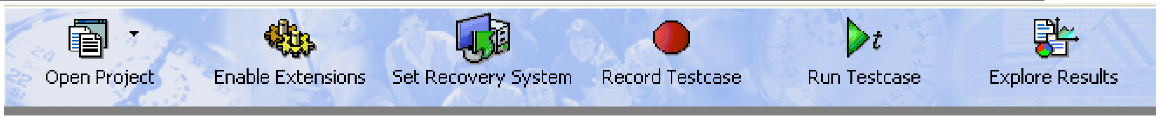
2.3 Setting Recovery System

The recovery system is used to set the default base state for the application.

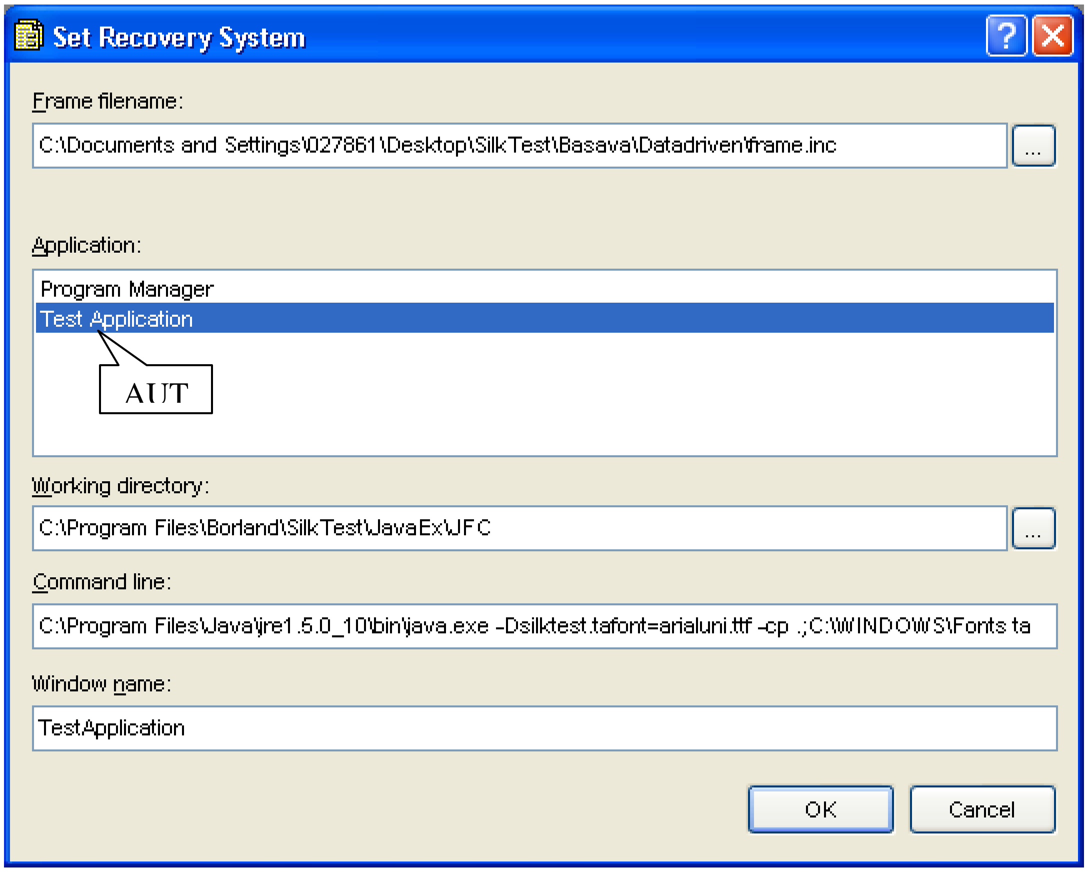
By default, the recovery system carries out four tasks for client-server applications:

- Ensures that the application or browser is running
- Restores the application if it is minimized
- Sets the application active
- Closes any application child windows or dialog boxes

1. To set the recovery system, launch AUT, then click the “Set Recovery System” icon on the “Basic Workflowbar” of the SilkTest Host.



The Set Recovery System dialog is displayed with the list of the open applications in the "Application" section.



Frame filename: C:\Documents and Settings\027861\Desktop\SilkTest\Basava\Datadriven\frame.inc

Application:
Program Manager
Test Application

Working directory: C:\Program Files\Borland\SilkTest\JavaEx\JFC

Command line: C:\Program Files\Java\jre1.5.0_10\bin\java.exe -Dsilktest.tafont=arialuni.ttf -cp .;C:\WINDOWS\Fonts ta

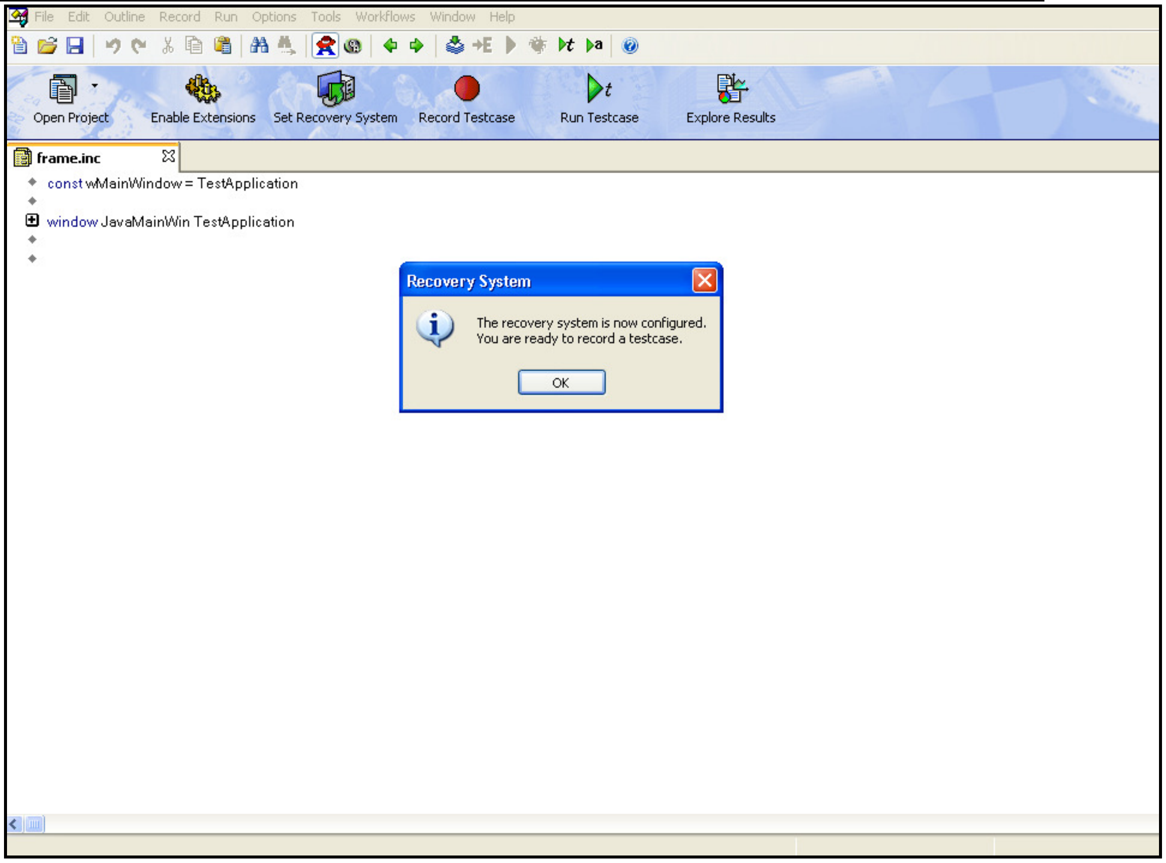
Window name: TestApplication

Figure 6Set Recovery System Dialog

2. Select the application from the application section, then click on the "OK" button.

3. Click on the "OK" button in the recovery system completion popup message dialog.

New "frame.inc" file is created.



```
const wMainWindow = TestApplication

window JavaMainWin TestApplication
```

The recovery system is now configured.
You are ready to record a testcase.

3. Click on the "OK" button in the popup message window.

3 Creating New Test

A new test in SilkTest can be created from SilkTest Host.

There are two types of test files: Script files, and data-driven files.

3.1 Scripts File Creation in SilkTest

To create a new script file,

1. Select File → New

   OR

   Click on the toolbar icon.

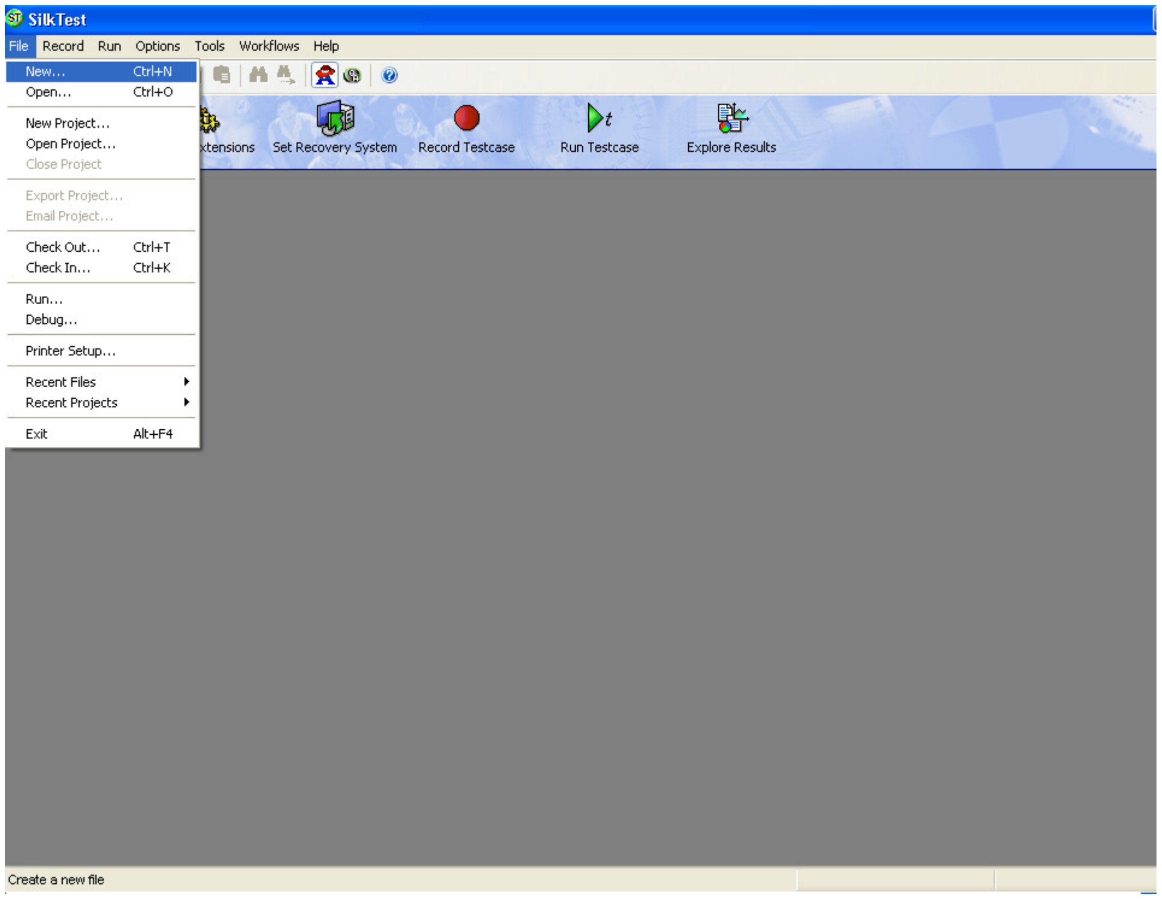
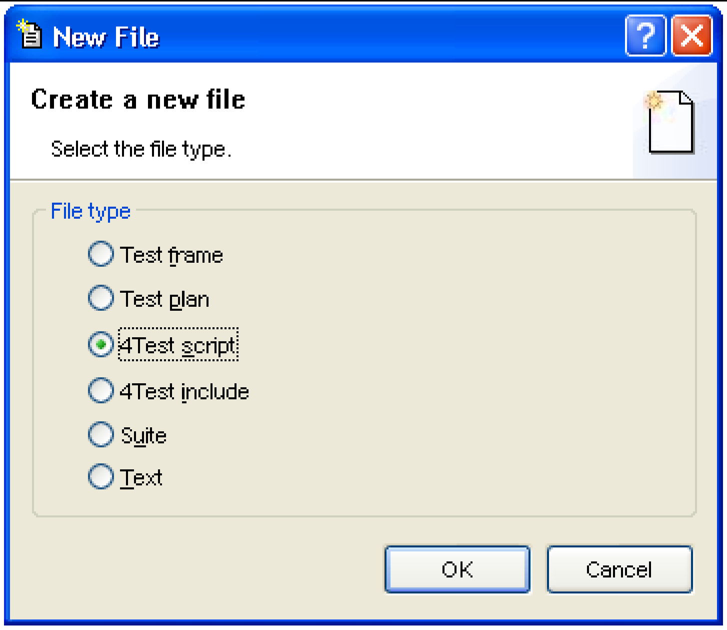


Figure 7New File Creation

2. Select the "4Test script" option and click on the "OK" button in the New File dialog.


New File

Create a new file

Select the file type.

File type

- Test frame
- Test plan
- [✓ 4Test script]
- 4Test include
- Suite
- Text

Figure 8New File Selection Dialog

A blank script file will be created.

The file can then be saved in the preferred location.

3.2 Creating New Data-Driven Test

Note: Before creating a data-driven test file, a 4Test script file and a keyword script file or keyword script template should be prepared.

1. Select Workflows → Data Driven

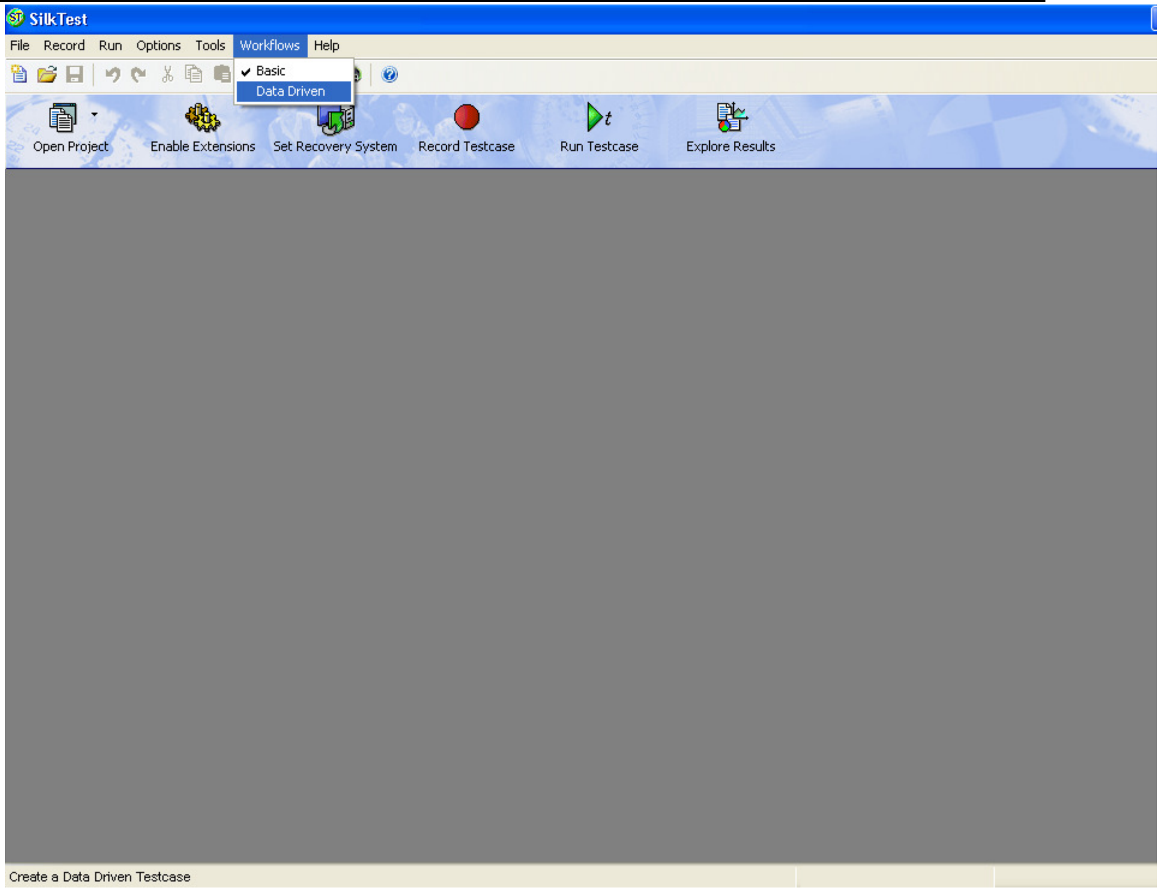


Figure 9SilkTest Workflows

2. Click on File → Open or click on toolbar icon.

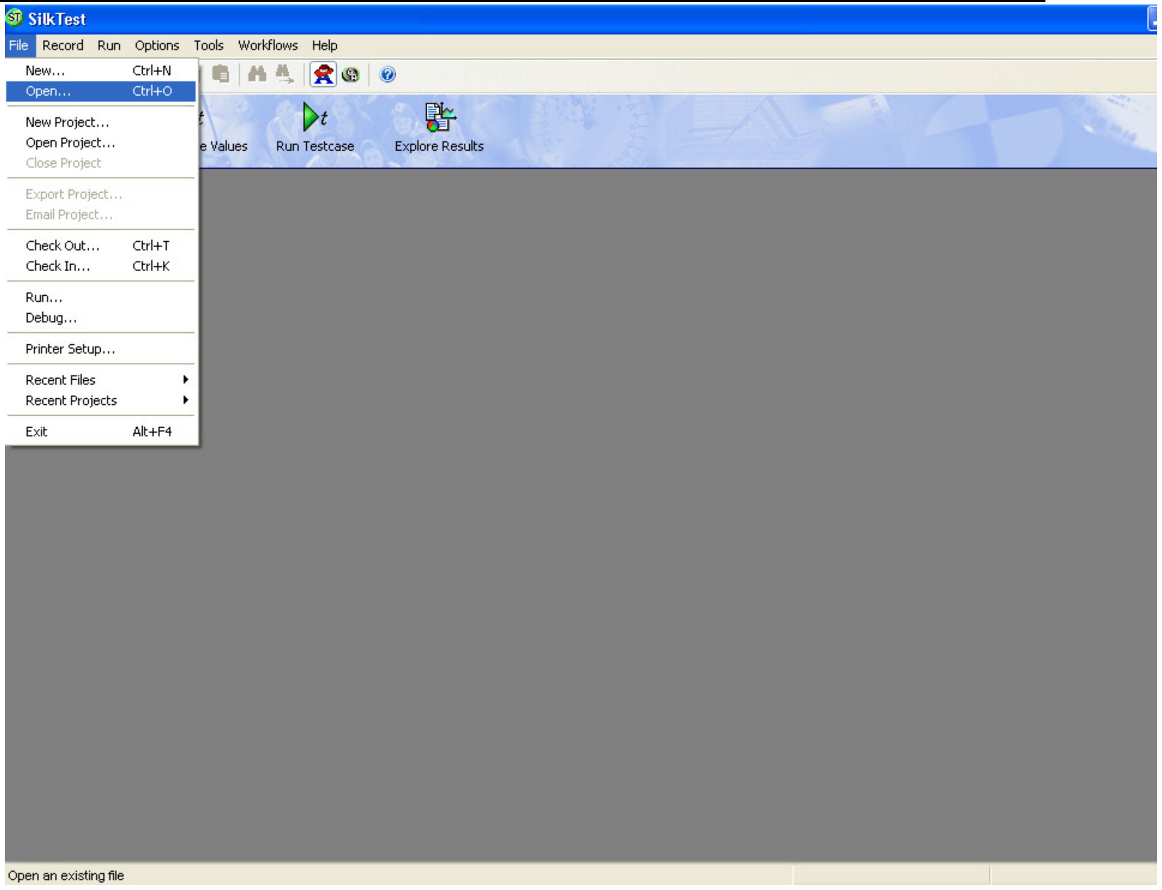


Figure 10 **Opening an Existing File**

3. Select an existing script (.t) file and then click on the "Open" button.

Note: If the script file is not created or doesn't exist, then create a new script file as explained in section 3.1

4. Now click on the "Data Drive Testcase" icon in the "Data Driven Workflowbar" of SilkTest.

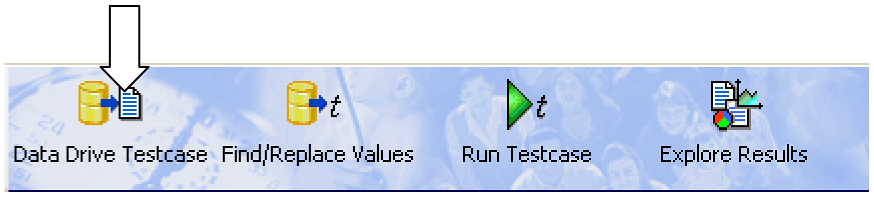
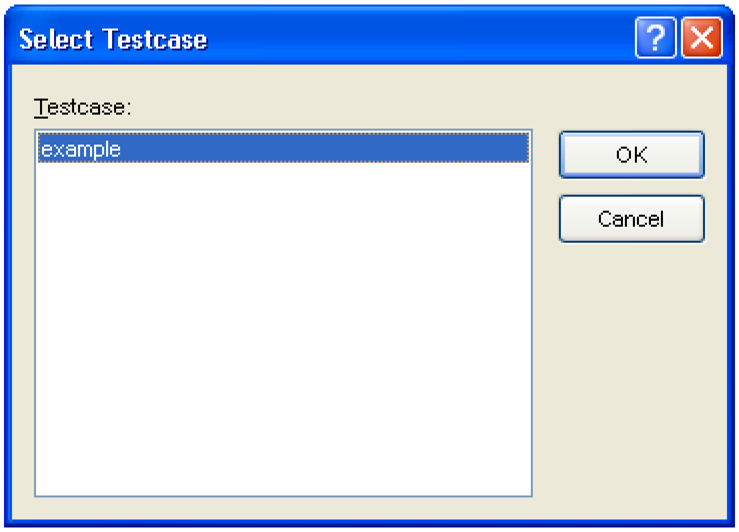


Figure 11 **Data-Driven Workflowbar**

5. Select the testcase name from the Testcase section in the "Select Testcase" dialog, then click on the "OK" button.

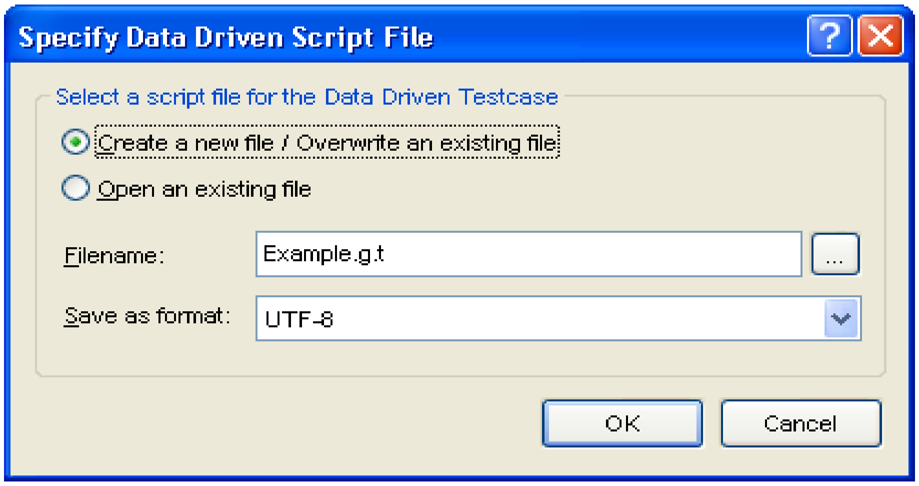


Testcase:

example

Figure 12 **Select Testcase Dialog**

6. Specify data-driven script file in the Specify Data Driven Script File dialog, then click on the "OK" button.



Select a script file for the Data Driven Testcase

(●) Create a new file / Overwrite an existing file

( ) Open an existing file

Filename: Example.g.t

Save as format: UTF-8

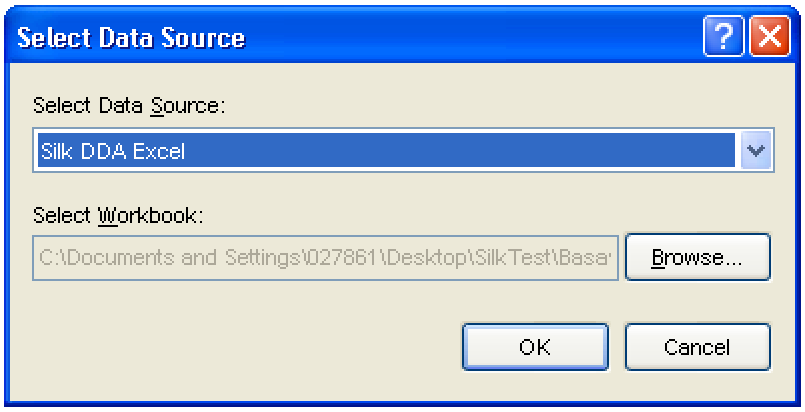
Figure 13 **Specify Data-Driven Script File Dialog**

Note: The Filename will be displayed by default (i.e., the same name as the ".t" file with an extension ".g.t").

7. Select a data source file and then click on the "Browse" button in the select data source dialog.

8. Browse for the keyword scripts Excel sheet path, click on "Open", and then click on the "OK" button in the select data

source dialog.



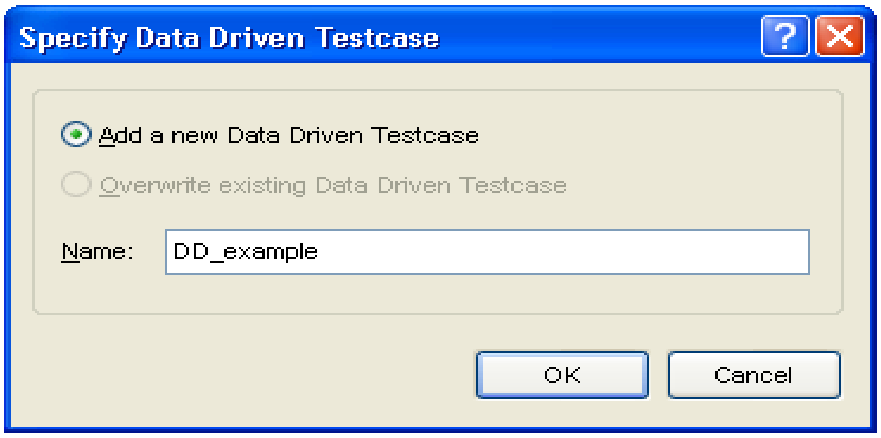
Select Data Source: Silk DDA Excel

Select Workbook: C:\Documents and Settings\027861\Desktop\SilkTest\Basa

Figure 14 **Select Data Source Dialog**

Note: Silk DDA Excel data source file is used for keyword scripting.

9. Click on the "OK" button in the Specify Data Driven Testcase dialog.



(●) Add a new Data Driven Testcase
( ) Overwrite existing Data Driven Testcase

Name: DD_example

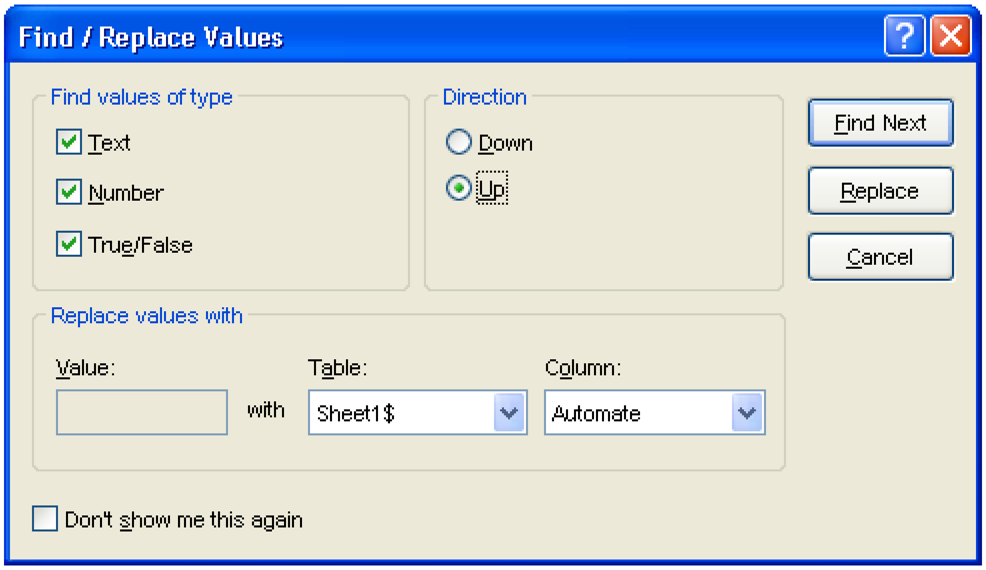
Figure 15 **Select Data Driven Testcase Dialog**

Note: The name of the data-driven testcase is displayed by default (i.e., it will be same as the script (.t) file prefixed with "DD_").

10. Click on the "Cancel" button in the "Find/Replace Value" dialog.

A new data-driven test is generated after clicking on the "Cancel" button in the "Find/Replace Value" dialog.

Note: The sheet names and column names of the keyword script Excel file will be displayed in the table and column droplists. This is a validation of the set-up.



Find values of type: Text [✓], Number [✓], True/False [✓]

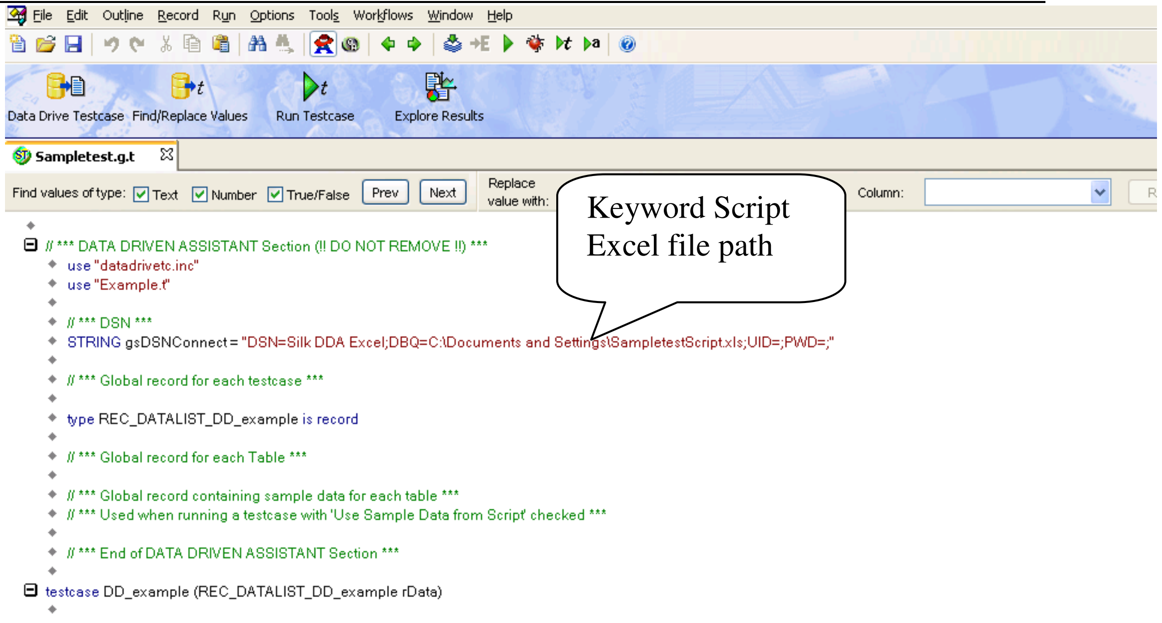
Direction: Down [✓ Up]

Table: Sheet1$

Column: Automate

Don't show me this again [ ]

Figure 16 **Find/Replace Values Dialog**


File Edit Outline Record Run Options Tools Workflows Window Help
Data Drive Testcase
Find/Replace Values
Run Testcase
Explore Results
Sampletest.g.t
Find values of type: Text Number True/False Prev Next Replace value with:
Column:
Keyword Script Excel file path

```
// *** DATA DRIVEN ASSISTANT Section (!! DO NOT REMOVE !!) ***
    use "datadrivetc.inc"
    use "Example.t"

    // *** DSN ***
    STRING gsDSNConnect = "DSN=Silk DDA Excel;DBQ=C:\Documents and Settings\SampletestScript.xls;UID=;PWD=;"

    // *** Global record for each testcase ***

    type REC_DATALIST_DD_example is record

    // *** Global record for each Table ***

    // *** Global record containing sample data for each table ***
    // *** Used when running a testcase with 'Use Sample Data from Script' checked ***

    // *** End of DATA DRIVEN ASSISTANT Section ***

testcase DD_example (REC_DATALIST_DD_example rData)
```

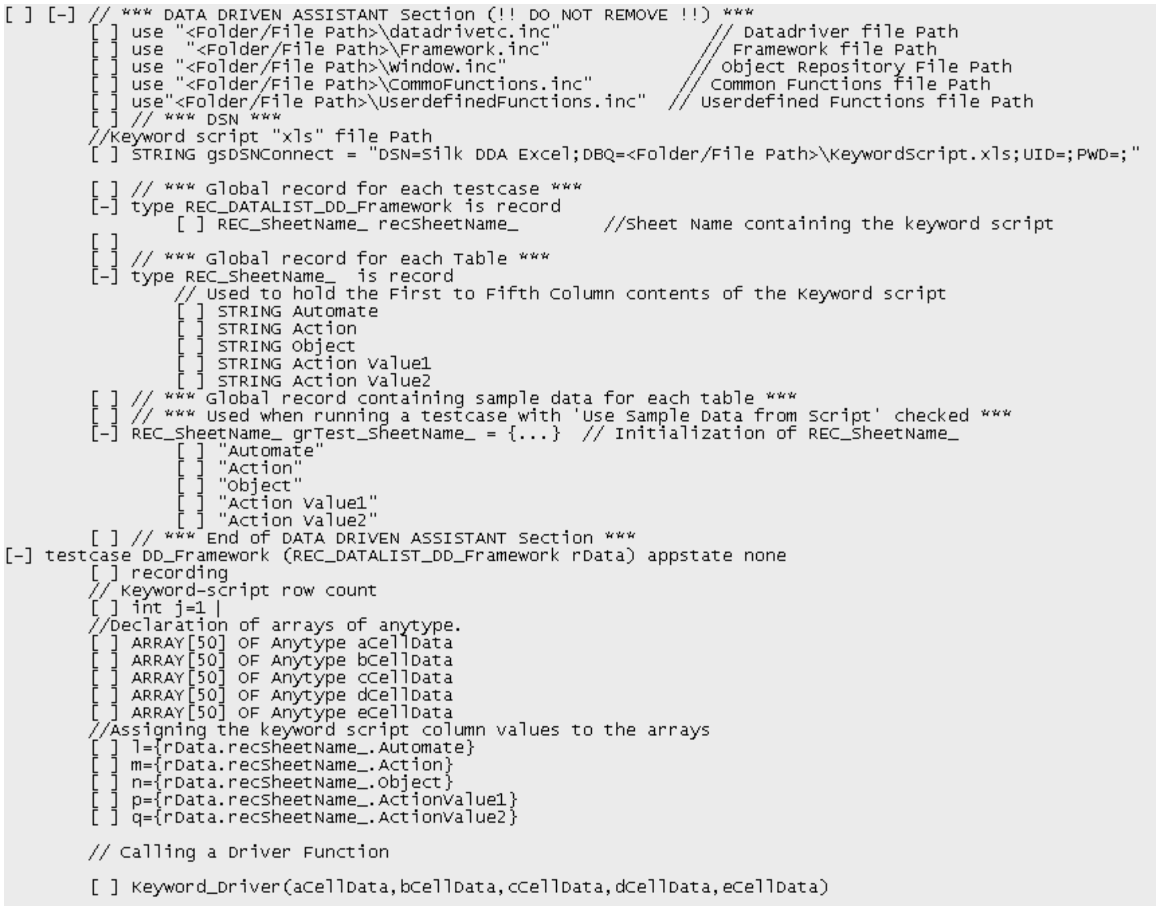
Figure 17 **New Data-Driven Test file**

4 Test Settings for Keyword-Driven Scripting

The data-driven testing approach available in SilkTest is used for achieving the keyword-driven approach.

With the keyword-driven approach, the entire script is developed with keywords. The script is developed in a spreadsheet that is interpreted by the main driver script, which then uses the function library to execute the complete script.

The following test setting needs to be done before proceeding with the keyword-driven scripting.



```
[ ] [-] // *** DATA DRIVEN ASSISTANT Section (!! DO NOT REMOVE !!) ***
        [ ] use "<Folder/File Path>\datadrivetc.inc"                    // Datadriver file Path
        [ ] use  "<Folder/File Path>\Framework.inc"                    // Framework file Path
        [ ] use "<Folder/File Path>\Window.inc"                       // Object Repository File Path
        [ ] use "<Folder/File Path>\CommoFunctions.inc"             // Common Functions file Path
        [ ] use"<Folder/File Path>\UserdefinedFunctions.inc"  // Userdefined Functions file Path
        [ ] // *** DSN ***
        //Keyword script "xls" file Path
        [ ] STRING gsDSNConnect = "DSN=Silk DDA Excel;DBQ=<Folder/File Path>\KeywordScript.xls;UID=;PWD=;"

        [ ] // *** Global record for each testcase ***
        [-] type REC_DATALIST_DD_Framework is record
                [ ] REC_SheetName_ recSheetName_        //Sheet Name containing the keyword script
        [ ]
        [ ] // *** Global record for each Table ***
        [-] type REC_SheetName_  is record
                // Used to hold the First to Fifth Column contents of the Keyword script
                [ ] STRING Automate
                [ ] STRING Action
                [ ] STRING Object
                [ ] STRING Action Value1
                [ ] STRING Action Value2
        [ ] // *** Global record containing sample data for each table ***
        [ ] // *** Used when running a testcase with 'Use Sample Data from Script' checked ***
        [-] REC_SheetName_ grTest_SheetName_ = {...}  // Initialization of REC_SheetName_
                [ ] "Automate"
                [ ] "Action"
                [ ] "Object"
                [ ] "Action Value1"
                [ ] "Action Value2"
        [ ] // *** End of DATA DRIVEN ASSISTANT Section ***
[-] testcase DD_Framework (REC_DATALIST_DD_Framework rData) appstate none
        [ ] recording
        // Keyword-script row count
        [ ] int j=1 |
        //Declaration of arrays of anytype.
        [ ] ARRAY[50] OF Anytype aCellData
        [ ] ARRAY[50] OF Anytype bCellData
        [ ] ARRAY[50] OF Anytype cCellData
        [ ] ARRAY[50] OF Anytype dCellData
        [ ] ARRAY[50] OF Anytype eCellData
        //Assigning the keyword script column values to the arrays
        [ ] l={rData.recSheetName_.Automate}
        [ ] m={rData.recSheetName_.Action}
        [ ] n={rData.recSheetName_.Object}
        [ ] p={rData.recSheetName_.ActionValue1}
        [ ] q={rData.recSheetName_.ActionValue2}

        // Calling a Driver Function

        [ ] Keyword_Driver(aCellData,bCellData,cCellData,dCellData,eCellData)
```

Figure 18 **Sample Keyword-Driven Test**

4.1 Data-Driven Assistance Section

The following file paths need to be mentioned in the Data-Driven Assistance Section:

1. Datadriver File – Specify the path of the file where the datadriver script is stored.

2. Framework File – Specify the path of the file where the framework script is placed.

3. Window Declaration File – Specify the path of the file where

the object repository is placed.

4. Common Function and User-Defined Functions - Specify the path of the common functions and user-defined functions associated with the framework script.

4.2 DSN (Data Source Name)

Mention the path where the keyword script excel file is placed. Also mention the username and password, if applicable in the DBQ path.

Note: DSN file name will be displayed by default, which will be the data source file name that you selected while creating a data-driven script.

4.3 Global Record for Each Testcase

Mention the sheet name used for driving the keyword script in the keyword script excel file (i.e., after creating a new data-driven script file, the following script will be generated by default in the Global Record for each testcase section).

type REC_DATALIST_DD_example is record

Below this line, mention the following as shown in figure 4.1 above.

REC_<SheetName>_ rec<SheetName>_

Note: SheetName will be the name of the tab containing the script in the keyword script Excel file.

4.4 Global Record for Each Table

The columns of the keyword script Excel file need to be declared in this section (i.e., all the columns which are used for keyword scripting).

A sample script is given below:

```
[-] type REC_SheetName_ is record
        [ ] STRING <Column A name>
        [ ] STRING <Column B name>
        [ ] STRING <Column C name>
        [ ] STRING <Column D name>
        [ ] STRING <Column E name>
```

4.5 Global Record Containing Sample Data for Each Table

The column names used in the keyword script Excel file need to be declared in this section:

A sample script is given below:

```
[+] REC_<SheetName>_ grTest_<SheetName>_ = {...}
[ ] "Automate"                  // First Column name of the keyword script excel
[ ] "Action"                    //Second Column name of the keyword script excel
[ ] "Object"                 //Third Column name of the keyword script excel
[ ] "Action Value1"          // Fourth Column name of the keyword script excel
[ ] "Action Value2"     // Fifth Column name of the keyword script excel
```

4.6 Calling Keyword Script

The following line will be generated after creating a new data-driven test (.g.t) file.

[-] testcase DD_example (REC_DATALIST_DD_example rData) appstate none

Note: "DD_example" is the name of the data-driven testcase.

A sample script is given below:

```
[-] testcase DD_example (REC_DATALIST_DD_example rData) appstate none
      [ ] recording
      //Specifies the first line of the keyword script, and it's the counter of the keyword script.
      [ ] Int j=1
      //Declaration of arrays of anytype
      [ ] ARRAY[50] OF Anytype aCellData
      [ ] ARRAY[50] OF Anytype bCellData
      [ ] ARRAY[50] OF Anytype cCellData
      [ ] ARRAY[50] OF Anytype dCellData
      [ ] ARRAY[50] OF Anytype eCellData

      // Assigning of each column values to carry for particular row
      [ ] l={rData.rec<SheetName>_.<Column A Name>}
      [ ] m={rData.rec<SheetName>_.<Column B Name>}
      [ ] n={rData.rec<SheetName> _.<Column C Name>}
      [ ] p={rData.rec<SheetName> _.<Column D Name>}
      [ ] q={rData.rec<SheetName>_.<Column E Name>}

      //Keyword driver function is called here
      [     ]     Keyword_Driver(aCellData,bCellData,cCellData,dCellData,
      eCellData)       // Call to Framework.
```

5 Managing Object Repository and Other Files

SilkTest learns the interface of an application to be able to work with it. It does this by interpreting the objects and recognizing them based on the class, properties, and methods that uniquely identify them. During testing, Silk Test interacts with the objects to submit operations to the application automatically, simulating the actions of a user, and then verifies the results of each operation. The simulated user, SilkTest, is said to be driving the application.

The repository includes descriptions of the GUI objects that comprise the application. Based on the properties and methods SilkTest associates with these objects, SilkTest can recognize the actions performed on them and intelligently record those actions into test script using the 4Test language.

The objects from the window declaration repository will be uploaded and made available for the tests either by using them in the “DATA DRIVEN ASSISTANT Section” of the data-driven test (.g.t) file or will be uploaded and made available for the tests.

Follow the steps below to upload the object repository and other files (Common functions, Userdefined functions or any “.inc” files).

1. Click on Options > Runtime

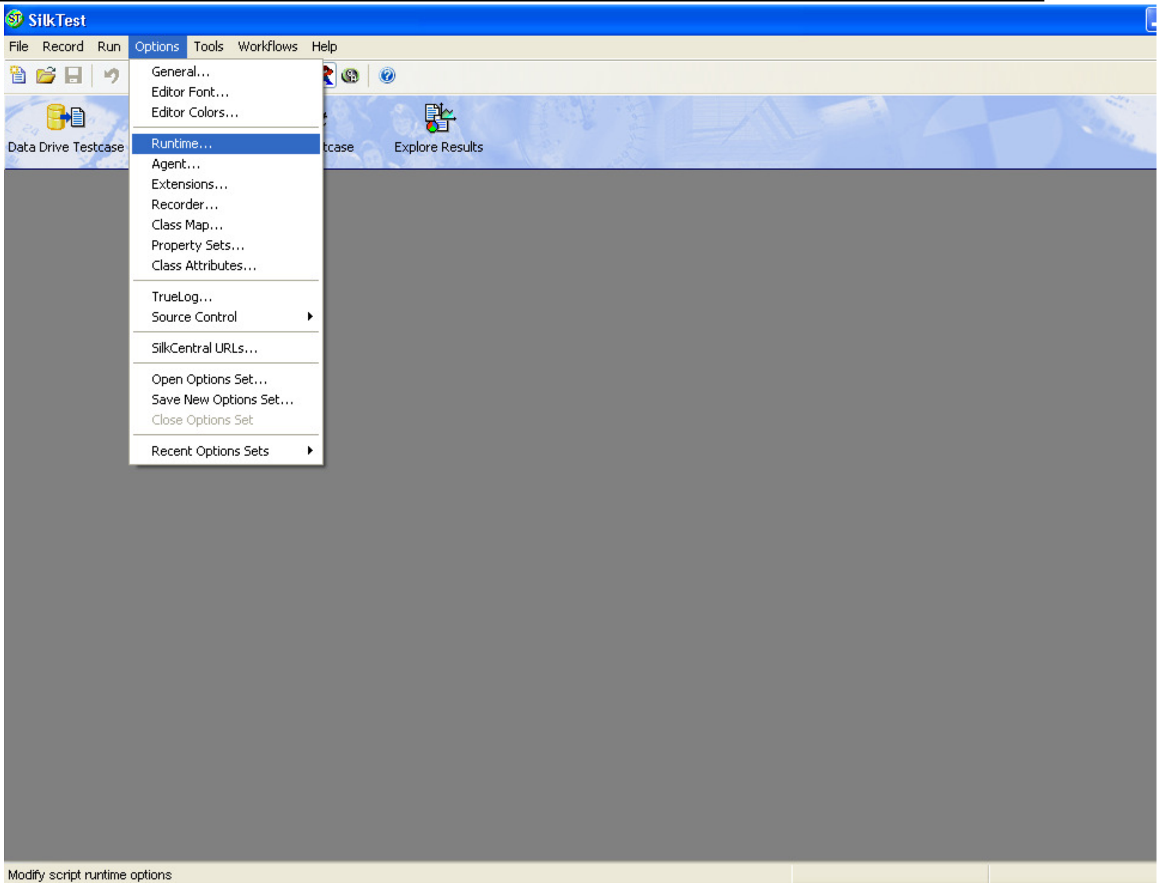
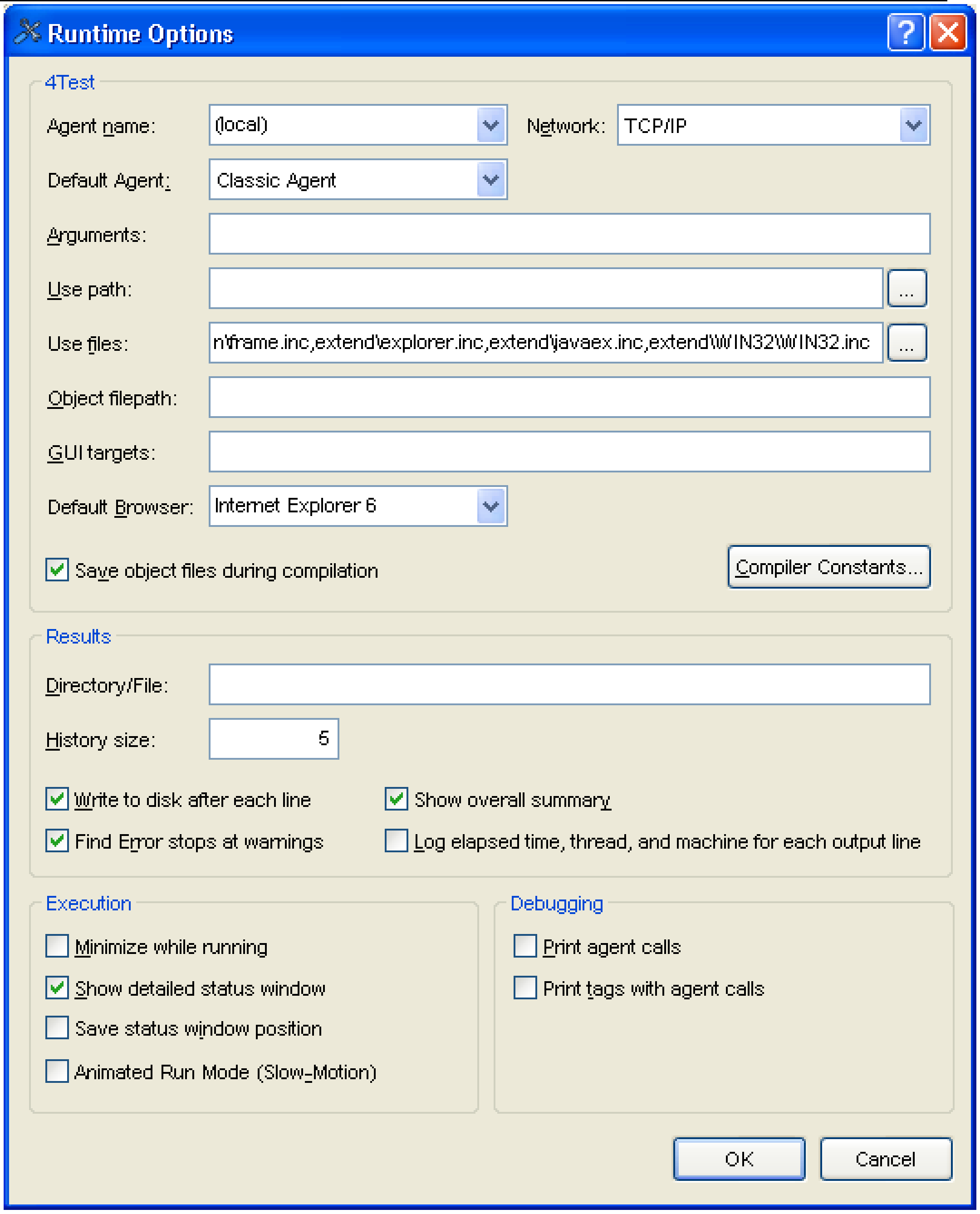


Figure 19 **Associate repository files Dialog**



Runtime Options

4Test

Agent name: (local)
Network: TCP/IP
Default Agent: Classic Agent
Arguments:
Use path:
Use files: n\frame.inc,extend\explorer.inc,extend\javaex.inc,extend\WIN32\WIN32.inc
Object filepath:
GUI targets:
Default Browser: Internet Explorer 6

[✓] Save object files during compilation

Results

Directory/File:
History size: 5

[✓] Write to disk after each line
[✓] Show overall summary
[✓] Find Error stops at warnings
[ ] Log elapsed time, thread, and machine for each output line

Execution

[ ] Minimize while running
[✓] Show detailed status window
[ ] Save status window position
[ ] Animated Run Mode (Slow-Motion)

Debugging

[ ] Print agent calls
[ ] Print tags with agent calls

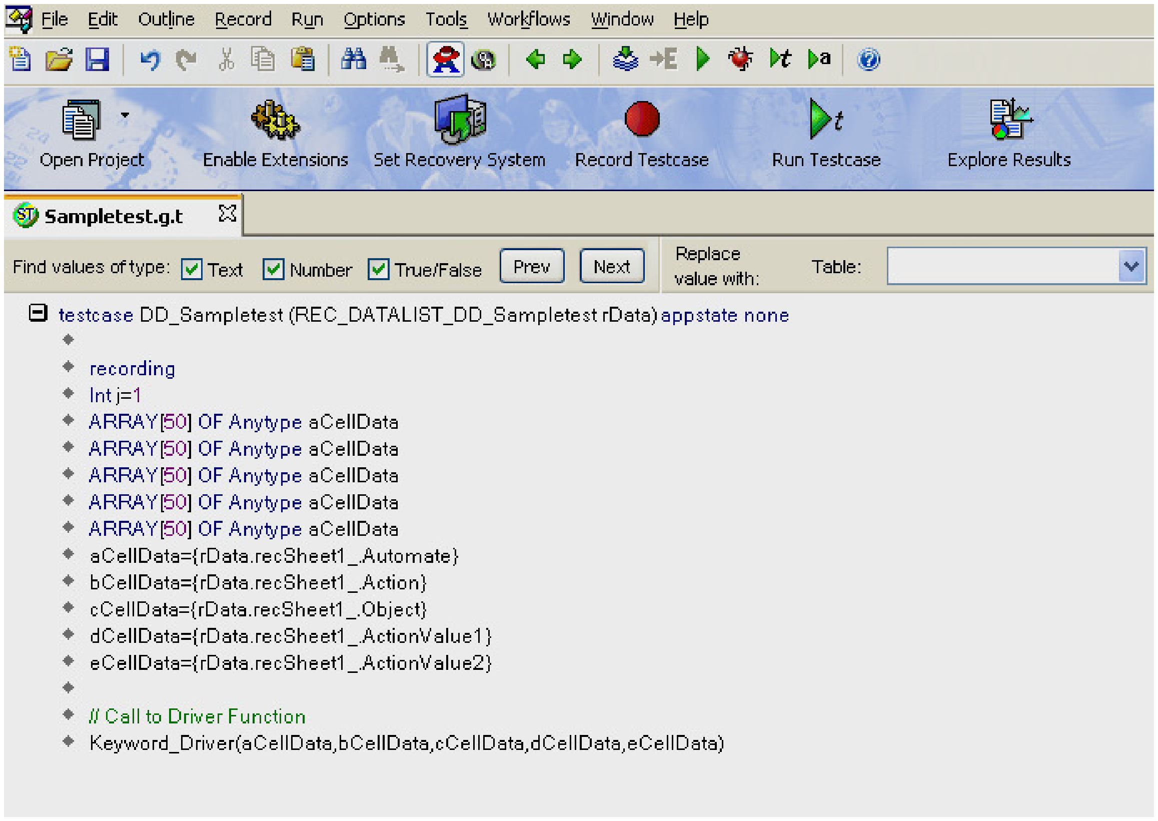
Figure 20 **Runtime Options Dialog**

2. Browse for the object repository files by clicking on the ... icon next to "Use path" filed in Runtime Options dialog, then click on the "OK" button.

Note: Can also associate Framework file, Common functions file, etc. by using the above method, instead of defining in the .g.t file.

6 Call to Framework

The call to Keyword_Driver () needs to be specified in the test script as shown below. This will call the framework file associated with the test and perform the actions by interpreting the keywords specified in the keyword script file. The parameters that are required to be passed to Keyword_Driver () are the names of the active data columns where the script is present.



```
testcase DD_Sampletest (REC_DATALIST_DD_Sampletest rData) appstate none

    recording
    Int j=1
    ARRAY[50] OF Anytype aCellData
    ARRAY[50] OF Anytype aCellData
    ARRAY[50] OF Anytype aCellData
    ARRAY[50] OF Anytype aCellData
    ARRAY[50] OF Anytype aCellData
    aCellData={rData.recSheet1_Automate}
    bCellData={rData.recSheet1_Action}
    cCellData={rData.recSheet1_Object}
    dCellData={rData.recSheet1_ActionValue1}
    eCellData={rData.recSheet1_ActionValue2}

    // Call to Driver Function
    Keyword_Driver(aCellData,bCellData,cCellData,dCellData,eCellData)
```

Figure 21 Call to Framework

The script can be run by clicking on the ▶ icon on the toolbar or by pressing the F9 button.

7 Usage of Keywords

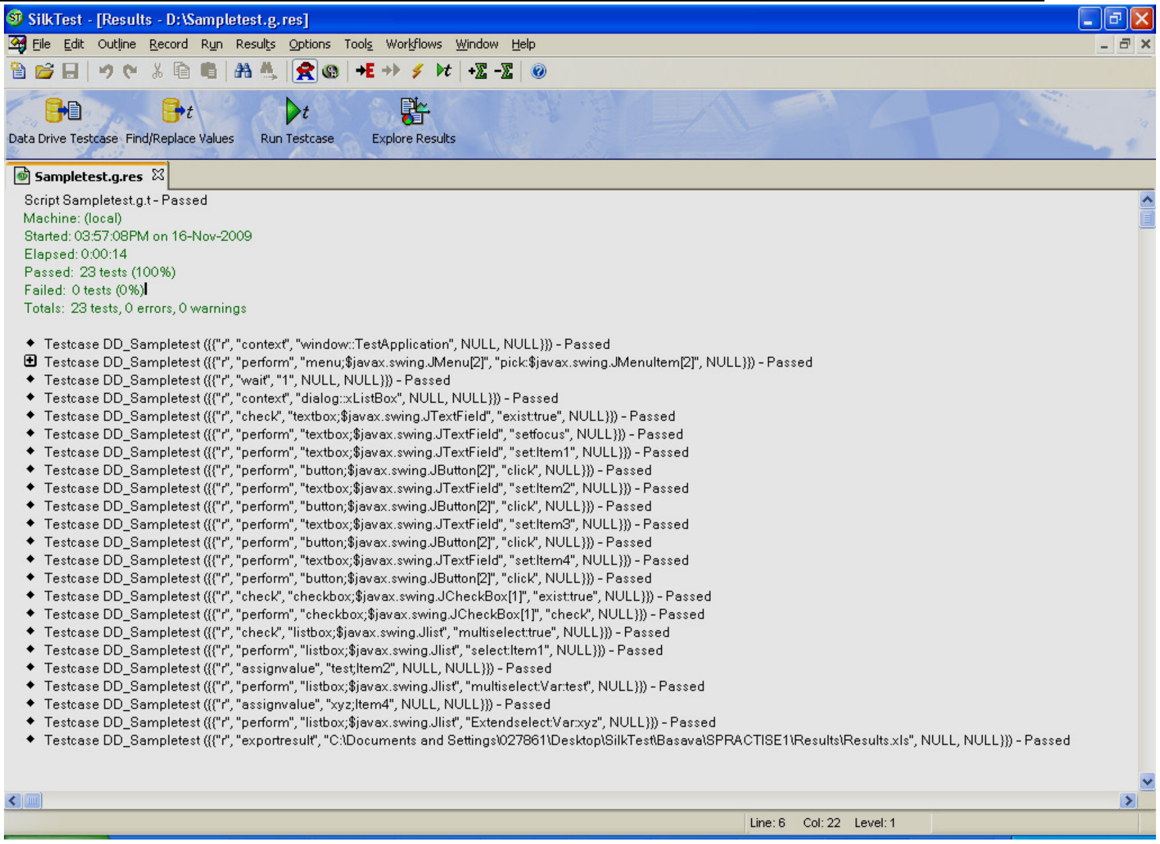
The keywords should be entered in the Excel sheet, and this Excel sheet will be associated to the test in the DSN section of the data-driven test as explained in section 4.2

Note: Refer to the keyword naming convention document for more details about keyword syntax.

8 Test Results for a Keyword-Driven Script

Test execution results can be viewed and analyzed as soon as the run session ends. Click on the "Explore Results" icon on the Data-Driven Workflowbar and choose the file.

The results window will be displayed.


SilkTest - [Results - D:\Sampletest.g.res]
File Edit Outline Record Run Results Options Tools Workflows Window Help
Data Drive Testcase
Find/Replace Values
Run Testcase
Explore Results
Sampletest.g.res
Line: 6 Col: 22 Level: 1

```
Script Sampletest.g.t - Passed
Machine: (local)
Started: 03:57:08PM on 16-Nov-2009
Elapsed: 0:00:14
Passed:  23 tests (100%)
Failed:  0 tests (0%)
Totals:  23 tests, 0 errors, 0 warnings

◆ Testcase DD_Sampletest ({{"r", "context", "window::TestApplication", NULL, NULL}}) - Passed
⊞ Testcase DD_Sampletest ({{"r", "perform", "menu;$javax.swing.JMenu[2]", "pick:$javax.swing.JMenuItem[2]", NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "wait", "1", NULL, NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "context", "dialog::xListBox", NULL, NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "check", "textbox;$javax.swing.JTextField", "exist:true", NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "perform", "textbox;$javax.swing.JTextField", "setfocus", NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "perform", "textbox;$javax.swing.JTextField", "set:Item1", NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "perform", "button;$javax.swing.JButton[2]", "click", NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "perform", "textbox;$javax.swing.JTextField", "set:Item2", NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "perform", "button;$javax.swing.JButton[2]", "click", NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "perform", "textbox;$javax.swing.JTextField", "set:Item3", NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "perform", "button;$javax.swing.JButton[2]", "click", NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "perform", "textbox;$javax.swing.JTextField", "set:Item4", NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "perform", "button;$javax.swing.JButton[2]", "click", NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "check", "checkbox;$javax.swing.JCheckBox[1]", "exist:true", NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "perform", "checkbox;$javax.swing.JCheckBox[1]", "check", NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "check", "listbox;$javax.swing.Jlist", "multiselect:true", NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "perform", "listbox;$javax.swing.Jlist", "select:Item1", NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "assignvalue", "test;Item2", NULL, NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "perform", "listbox;$javax.swing.Jlist", "multiselect:Var:test", NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "assignvalue", "xyz;Item4", NULL, NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "perform", "listbox;$javax.swing.Jlist", "Extendselect:Var:xyz", NULL}}) - Passed
◆ Testcase DD_Sampletest ({{"r", "exportresult", "C:\Documents and Settings\027861\Desktop\SilkTest\Basava\SPRACTISE1\Results\Results.xls", NULL, NULL}}) - Passed
```

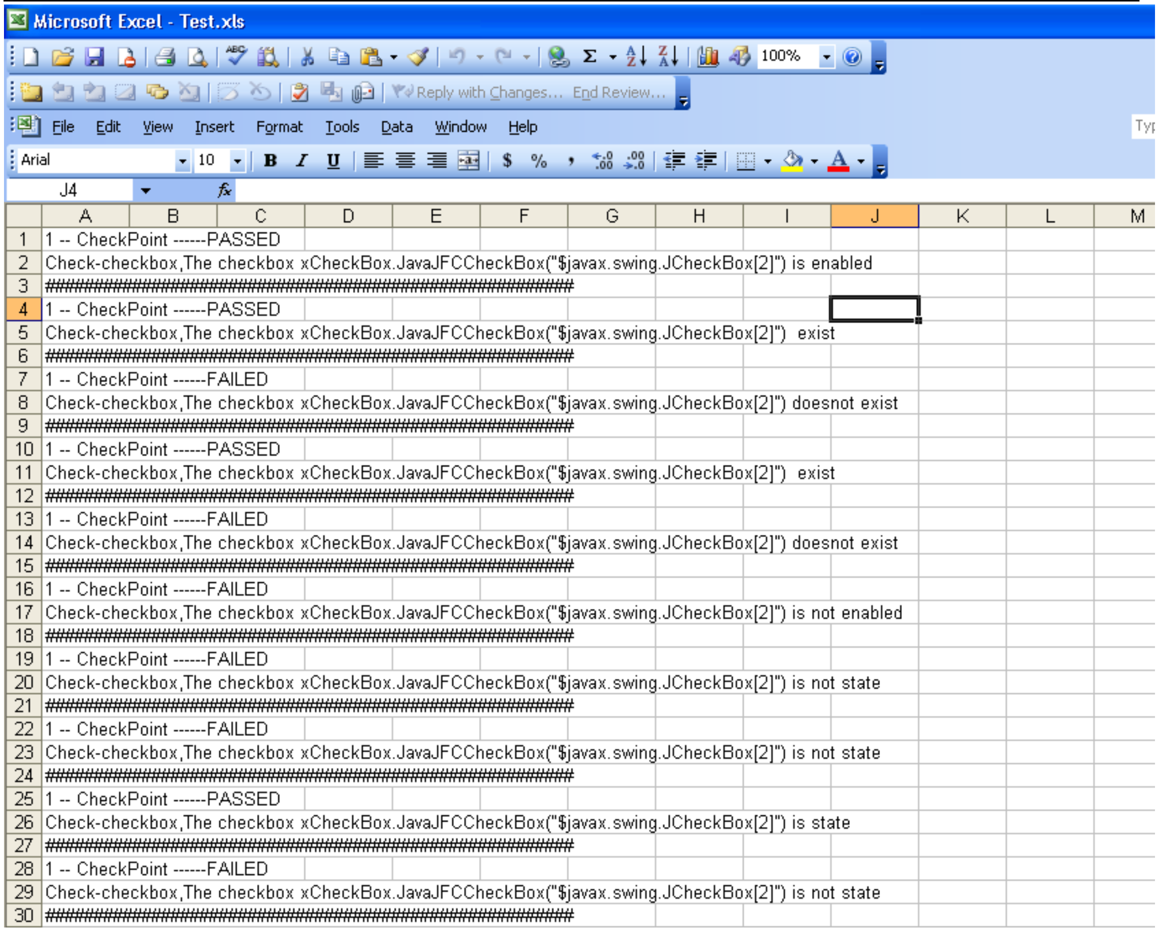
Figure 22 **Test Results in SilkTest**

8.1 External Test Results for Keyword-Driven Test Scripts

The check point test execution results can be viewed and analyzed as soon as the run session ends.

Open the Results.xls file present in the results folder. The result file will be displayed.

Note: The checkpoint results file would be stored in the results folder in the script file path.



Microsoft Excel - Test.xls

	A
1	1 -- CheckPoint ------PASSED
2	Check-checkbox,The checkbox xCheckBox.JavaJFCCheckBox("$javax.swing.JCheckBox[2]") is enabled
3	##
4	1 -- CheckPoint ------PASSED
5	Check-checkbox,The checkbox xCheckBox.JavaJFCCheckBox("$javax.swing.JCheckBox[2]") exist
6	##
7	1 -- CheckPoint ------FAILED
8	Check-checkbox,The checkbox xCheckBox.JavaJFCCheckBox("$javax.swing.JCheckBox[2]") doesnot exist
9	##
10	1 -- CheckPoint ------PASSED
11	Check-checkbox,The checkbox xCheckBox.JavaJFCCheckBox("$javax.swing.JCheckBox[2]") exist
12	##
13	1 -- CheckPoint ------FAILED
14	Check-checkbox,The checkbox xCheckBox.JavaJFCCheckBox("$javax.swing.JCheckBox[2]") doesnot exist
15	##
16	1 -- CheckPoint ------FAILED
17	Check-checkbox,The checkbox xCheckBox.JavaJFCCheckBox("$javax.swing.JCheckBox[2]") is not enabled
18	##
19	1 -- CheckPoint ------FAILED
20	Check-checkbox,The checkbox xCheckBox.JavaJFCCheckBox("$javax.swing.JCheckBox[2]") is not state
21	##
22	1 -- CheckPoint ------FAILED
23	Check-checkbox,The checkbox xCheckBox.JavaJFCCheckBox("$javax.swing.JCheckBox[2]") is not state
24	##
25	1 -- CheckPoint ------PASSED
26	Check-checkbox,The checkbox xCheckBox.JavaJFCCheckBox("$javax.swing.JCheckBox[2]") is state
27	##
28	1 -- CheckPoint ------FAILED
29	Check-checkbox,The checkbox xCheckBox.JavaJFCCheckBox("$javax.swing.JCheckBox[2]") is not state
30	##

Figure 23 External Test Results in SilkTest

9 References

SilkTest Help document