TestPro
The Testing Professionals

TAF PRO

Scripting Guide

Version 4.4

DISCLAIMER

The authors have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No Liability is assumed for incidental or consequential damages in connection with or arising out of the use the information or instructions contained herein.

TestPro, the TestPro Logo, TestPro Automation Framework (TAF Pro), and the TAF Pro logo are registered patents in Australia. IBM, the IBM Logo, Rational Eclipse are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Microsoft and Internet Explorer are either registered trademarks or trademarks of Microsoft Corporation in Australia, other countries or both. Java and all Java-based trademarks and logos, MySQL and all MySQL trademarks and logos are trademarks of Sun Microsystems, Inc, in Australia, other countries or both. Other company, product, or services names may be trademarks or service marks of others.

Any alterations made without express permission of TestPro Pty Ltd is deemed unauthorised and unverified and will not in any way be the responsibility of TestPro Pty Ltd.

Licenced Software Patent Numbers: 8195982 and 2010202573 - Property of TestPro Pty Ltd. Software is subject to Patent Laws. Use of this software is subject to a valid and current licence agreement with TestPro Pty Ltd. This application integrates with Open Source and IBM Rational Suite Products whose use is subject to independent licence agreements with those companies.

Contents

1 Introduction

Welcome to the ***TestPro Automated Framework*** (TAF Pro) Scripter Guide! This Guide has been written to assist those who write Test Scripts for Automated Test Tools, primarily IBM Rational Functional Tester, to write or adapt their scripts to work within the TAF Pro environment.

TAF Pro embodies TestPro's approach to automated testing processes and methodologies:

- TAF Pro is a hybrid automated testing framework that combines elements of data-driven and keyword-driven frameworks and adopting a role-ring of automation scripts to improve efficiency and scalability;
- To reduce the test maintenance overhead;
- To provide a non-technical user interface for Business Users and Testers;
- To capture and leverage the domain knowledge of business users into a set of regression tests that can be executed in a repetition-based approach to test development and execution.

The key objective of an automation framework is to enable reuse and process variation without needing the business users to be present.

The key functions within TAF Pro are:

- Execution of tests built with an automated testing tool (such as IBM Rational Functional Tester);
- Easing data handling challenges for data being used during testing;
- The linking of tests to form scenarios and suites of associated tests.

TAF Pro has been developed with a focus on:

- Minimising the impact of the framework on script development;
- Approaching test execution from a business perspective (as opposed to purely testing);
- Providing flexibility in data association and selection.

2 Overview

The key aspects of the TestPro Automation Framework architecture and implementation approach are:

- Review
    - applications to be tested and test objectives
    - type of test user interface required e.g. Spreadsheet, or GUI/database driven
    - automation tools to be used
- Define an automation framework architecture that is *efficient, scalable and maintainable*
- Implement a library of ‘adapter’ test scripts that test specific functions in the application(s) under test. These scripts are implemented using the automation tool selected by the client.
    - These adapter scripts reduce test maintenance because when the application under tests changes, the changes are typically accommodated within the library of adapter scripts, so that most of the functional tests will not require changes.
- Provide a simple, non-technical user interface to the function library so that business users can develop tests. Typically this interface will be Spreadsheet, or GUI/database driven

This guide has been developed to assist an experienced technical IBM Rational Functional Tester script developer to successfully set up and deploy scripts through the TAF Pro framework. Existing scripts can be converted to run and new scripts are effortlessly created with TAF Pro settings automatically applied to them.

2.1 Product Configuration

After installation of the TAF Pro product, before it is ready to use the first time, configuration needs to be completed such that TAF Pro is connected with the underlying Automation and database tools.

For full instructions on how to configure TAF Pro please refer to the TAF Pro Installation Guide as follows:

- **Section 4** – Create TAF Pro Central Repository;
- **Section 5** – Configuring Rational Functional Tester and;
- **Section 6** – Configuring TestPro Automation Framework.

3 Terms

3.1 TAF Pro Enablement

This section details the additional configuration steps required to be carried out on an existing or new Rational Functional Tester Project to enable TAF Pro capability to be utilised.

With TAF Pro Enablement, a Rational Functional Tester project can utilise TAF Pro features to plan tests with TAF Pro GUI and execute tests either with TAF Pro GUI or through the ANT Command Line interface.

Below is the summary of the configurations:

1. Copy TAF Pro Enablement folders to the Functional Tester project path (for detailed steps refer to the TAF Pro Installation Guide Section 5.1 and 5.2);
2. Configure script super class, template and classpath for the Ration Functional Tester project through Rational Functional Tester GUI (for detailed steps refer to the TAF Pro Installation Guide Section 5.3).

3.2 TAF Pro Enablement Folders

Refers to the TAF Pro Folders that need to be present in any Rational Functional Tester Project where the project is not located in the TAF Pro Installation Path. Refer to TAF Pro installation Guide section 5.2 for detailed information. If the Rational Functional Tester Project is located in the TAF Pro Installation Path these folders will already be present.

3.3 Rational Functional Tester TAF Pro Project

Refers to:

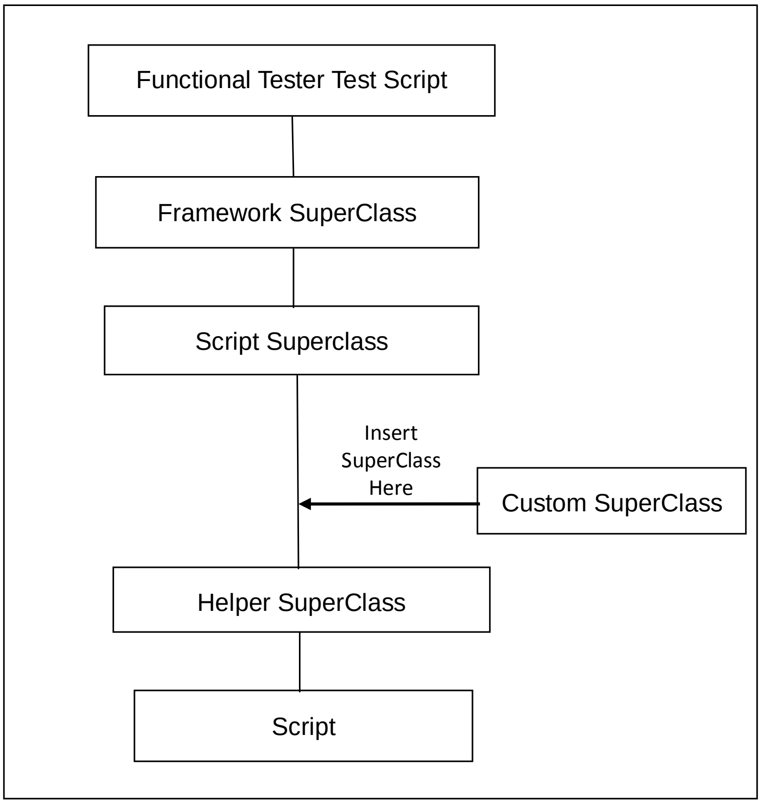
- A Functional Tester project with TAF Pro Enablement. OR
- A Functional Tester project which is located in TAF Pro installation path

4 Project Configuration

Scripts interface with TAF Pro via super class routines. RFT Scripts must be saved in a user defined package rather than the root folder of the RFT Project.

To extend the Functional Tester Superclass as well as keep the TAF Pro functionality, any custom Superclass should be inserted into the Superclass hierarchy as follows:

4.1 Superclass Hierarchy



5 Creating New Scripts with IBM Rational Functional Tester

After the Rational Functional Tester project is TAF Pro Enabled, any new scripts need to be created within these Projects as in Figure**Error! Reference source not found.** 1-1 and Figure 1-2 below. Individual folders an then be created to further segregate the test scripts of the project.

Note: Scripts that will be executed by TAF Pro MUST be in a package within the TAF Pro Enabled Functional Tester Project!

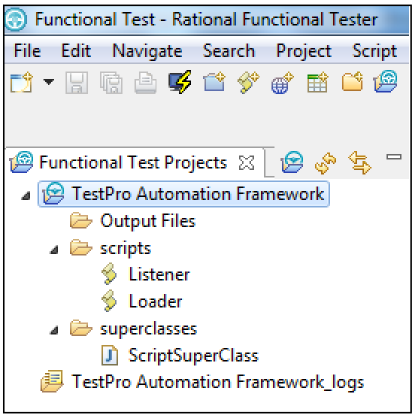


Figure 1-1 shows a basic TestPro Automation Framework project after installation.

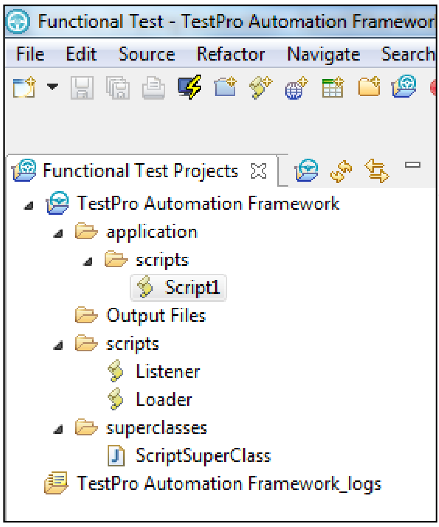


Figure 1-2 shows the TAF enabled project with an application\scripts folder containing a test script.

6 Converting Legacy Scripts to work with TAF Pro

In order to make legacy scripts available to TAF Pro, the following actions need to be taken:

6.1 To Export Legacy Script from Current Project

1. Select the scripts to be exported
2. Choose Export
3. Select Functional Test -> Functional Test Project Items
4. Click ***Next***
5. Expand the project folder, select the script files of your choice
6. ***Browse*** to required location for the export file and ***Enter export File Name***
7. Click Finish

6.2 To Import Legacy Scripts to TAF Pro Scripts

1. Select the required Functional Tester TAF Pro project created & configured as described in Installation Guide chapter 3 & 4. E.g. **TestPro Automation Framework in figure 1-1 or Classics_Demo in figure 1-2.**
2. Right Click and select ***Import***
3. Select **Functional Test -> Functional Test Project Items**
4. Click ***Next***.
5. Click ***Browse*** to select the exported projectrftjdtrfile,
6. Click ***Finish*** to complete the import.

6.3 To Convert the Legacy scripts to TAF Pro

1. Login to TAF Pro Automation Framework
2. Follow TestPro steps in the **TAF Pro User Guide** Chapter 8 Menu Bar – Script Functions, Converting Scripts

6.4 To Complete Script Conversion to TAF Pro

The conversion will change the header and put in a try catch block for the original script. Example original and converted scripts are shown below. The converter generates a new script (the converted script) to the directory nominated by the user in **Scripts > Convert > Script** step.

The Original legacy script header is as follows:

```
/**
  * Script Name   : <b>xxxxxxxx</b>
  * Generated     : <b>dd/MM/yyyy hh:mm:ss AA</b>
       * Purpose        : When a scripter creates a new script in TAF, this
       *                  template automatically applies the functions
       *                  requiredto manage the execution by TAF.
  * Original Host : WinNT Version 6.1  Build 7601 (S)
  *
  * TAF actions. Repeat the following line for multiple actions
  *
  * @testpro app="" action="" description=""
  *
  * @since  yyyy/mm/dd
  * @author aaaaaa
  */
```

Script name, date, time and username will be resolved when a script is created.

The Converted script header MUST include the following lines highlighted as follows:

```
/**
* Script Name   : <b>OrderTotal</b>
* Generated     : <b>30/08/2011 10:29:47 AM</b>
* Description   : Functional Test Script
* Original Host : WinNT Version 6.1  Build 7600 ()
*
* @testpro app="" action="" description=""
*
* @since  2014/04/30
* @author Tester1
*/
```

The scripter needs to update the empty fields in line @testpro app="", action=""description=""

The converted script testMain method is changed from returning void to returning TestResult object, as follow:

```
Public TestResult testMain(Object[] args)
```

The converted script has a try/catch block around the original script’s testMain method body and a return statement as shown below:

```
try {
……. .
} catch (RationalTestException ex) {
logStoreException(ex);
} catch (Exception ex) {
logStoreException(ex);
} finally {
unregisterAll();
}
return getResult();
}
}
```

After this modification, please copy the converted script to overwrite the original script.

The TAF Pro Script Converter doesn't preserve the original script's indentation, nor does it import the necessary TestResult class. The helper class of the overwritten script needs to extend TAF Pro superclass instead of RationalTestScript. Open the overwritten script in Functional Tester editor:

- The indentation can be reinstated by clicking Ctrl + Shift + F to reformat the converted script;
- Right-click **Source --> Organize Imports** or press Ctrl - Shift – O;
- The RFT editor will add the import statement:

```
import au.com.testpro.framework.core.beans.portal.TestResult;
```

- Click on the helper class, press F3, the helper class will be displayed in the editor, change the superclass from RationalTestScript into ScriptSuperClass. Then right-click **Source > Organize Imports**, the editor will add the import statement:

```
import superclasses.ScriptSuperClass
```

Comment [BL1]:

The Script conversion is now complete. User can launch TAF to set up planned scenarios, test cases, action and datagroups to execute the script from TAF.

6.5 Extending the Functionality of Scripts

The Scripter can use TAF Pro API's to accommodate more than one action existing in a single script as the following example shows:

```
     String action = getActionName();

  if (action.equals("Order")) {
    // do order
  } else if (action.equals("Cancel Order")) {
   // do cancel
  } else if (action.equals(" Delete Order")) {
   // do delete
  }
```

The Scripter can add/use TAF API getOutput and setOutput to manage precondition data and output data. These API's are described below.

The Scripter can add/use any method available in TAF Pro APIs to enhance the converted script. The available APIs are listed and explained in Section 8 of this Guide.

7 Variable Scope

TAF Pro variables may be passed from one script to another via setOutput and getOutput methods.

Variable values are retained for ***each row of data*** executed and are current for every action executed in every test case in every scenario the current row.

When TAF Pro begins execution of the next row, all variable values are erased and new values created.

Variables are **NOT** passed from one row of data to another.

7.1 Command Line Interface

In the Command Line Interface (**CLI**), all variable values are retained within each row within each test suite. Variables are **NOT** passed from one test suite to another.

Use the desktop icon to run tests via the command line:

- Select the build file you generated in TAF Pro prior to run!

Refer to the **TAF Pro User Guide, Test Execution with Command Line Interface** for details on how to set up and use CLI processing.

8 TAF Pro user APIs

8.1 tafStore();

Objective: Display row of data sent from TAF Pro.

Reason: Display the current row of data being used in the script execution.

Data are displayed at three levels, the current script, the current test suite row and output (precondition).

Syntax: tafStore();

Example Usage:

```
tafStore();
```

There are no parameters. The data names and values currently used by TAF Pro will be written to the Rational Functional Tester log.

Example Output for TAF Pro store:

```
[ScriptData={CommonKeyword:Common},{Risk:Medium},{var1:value1}]
[SuiteData={CommonKeyword:Common},{Risk:Low},{var2:value2},{var3:value3}]
[Precondition Data={var1:V12},{var3:V312},{var2:V222}]
```

8.2 setMessage("result message");

Objective: Set an information message to the TAF Pro result set.

Reason: Setting message in TAF Pro result set to give more information about execution result.

Syntax: setMessage(“result message”);

Example Usage:

```
setMessage("Client David successfully created ");
```

Note - setMessage can be invoked multiple times in a script. The message(s) can be viewed via the Results window at the DataGroup level when execution is complete.

8.3 setWarning("warning message");

Objective: Set a warning message to the TAF Pro result set.

Reason: Sets a warning message to give more information about the execution result; the script has not passed but did not fail either.

Syntax: setWarning(“warning message”);

Example Usage:

```
setWarning("Expected amount $9.98, actual is $10.00");
```

Note - setWarning can be invoked multiple times in a script. The warning message(s) can be viewed via the Results window at the DataGroup level when execution is complete.

8.4 setOutput("OutPutName",outPutValue);

Objective: Used for setting precondition data.

Reason: Keep execution results in TAF Pro, so other scripts can use those results as input data satisfy some processing flow.

- The maximum length of the message is 767 characters.

Syntax: setOutput("OutPutName",outPutValue);

Example Usage:

```
setOutput("Created Client",”David”);
```

8.5 getOutput("OutPutName");

Objective: Used for getting precondition data.

Reason: Getting precondition data as an input data for other scripts.

Syntax: getOutput("OutPutName");

Example Usage:

```
MatterManagement().inputKeys(getOutput("CreatedClient"));
```

8.6 dpString(“ColumnNameInSpreadsheet”);

Objective: Used for get value from TAF ProSpreadsheet to set text field on the application.

Reason: For data-driven script purposes, the dpString method is used for getting column value from TAF Pro Spreadsheet and as an input value for text field on the application.

Syntax: dpString(“ColumnNameInSpreadsheet”);

Example Usage:

```
clientText().setText(dpString("ClientName"));
```

Note: The name ***must*** be a quoted string; it cannot be an expression or a variable.

8.7 dpValue(“ColumnNameInSpreadsheet”);

Objective: To set the state value in check box or radio button objectsin the application under test.

Reason: For data-driven script purposes, thedpValue method is used for getting the column value from the TAF ProSpreadsheet and set theState value for check box or radio button objectsin the application.

Syntax: dpValue(“ColumnNameInSpreadsheet”);

Example Usage:

```
 rememberThePassword().clickToState((State)dpValue("RememberOrNot"));
```

In the TAF Pro Spreadsheet under the “RememberOrNot” column, use either “SELECTED” or “NOT_SELECTED” as the state’s values.getActionName();

8.8 getActionName()

Objective: Get the action value from Spreadsheet

Reason: This method returns a string value from the supplied Spreadsheet that can control the processing flow of the active script

Syntax: getActionName();

Example Usage:

```
if (getActionName().equalsIgnoreCase(“Create”)) {
                      //  Do CREATE
                  } else {
                     //  Do MODIFY
                  }
```

8.9 isFirstLine()

Objective: To be able to identify if the current row of data is the first row.

Reason: This method provides the ability to do conditional processing based on the first row and to avoid processing if testing is on additional data rows. In using this method you should also refer Scenario Properties in the TAF Pro User Guide for the Iterate capability within TAF Pro.

Syntax: isFirstLine();

Example Usage:

```
  if (isFirstLine()) {
                   // Do first line processing
  } else {
  // Do alternative processing
  }
```

8.10 isLastLine()

Objective: To be able to identify if the current row of data is the last row.

Reason: This method provides the ability to do conditional processing based on the last row and to avoid processing if testing has not reached the last line of data. In using this method you should also refer Scenario Properties in the TAF Pro User Guide for the Iterate capability within TAF Pro.

Syntax: isLastLine();

Example Usage:

```
 if (isLastLine()) {
  // Do last line processing
  } else {
  // Do alternative processing
  }
```

8.11 Finally{}

Objective: Defaulted as part of the TAF Pro template for all scripts created in a TAF Pro Enable project. Finally provides for any processing as the last step in the script regardless of the script state.

Reason: Finally should contain methods that will be executed regardless of whether the script has thrown and exception or not. One of these for best practice purposed should be UnregisterAll() which will release all mapped objects created while the script was running and thus release memory for the next iteration of the test.

Syntax: finally { UnregisterAll (); }

Example Usage:

```
                try {
                   // script code
                   } catch (RationalTestException ex) {
                       logStoreException(ex);
                   } finally {
                              unregisterAll();
                   }
```

9 Frequently Asked Questions

Q: What Functional Tester commands are Reserved Words?

A: We do not recommend Scripter using “**Action**”, “**Common Keyword**” and “**Risk**” as parameter names for dpString() method., because TAF Pro already uses them in its data Spreadsheet. Using these three names in dpString() method will cause confusion in TAF ProSpreadsheet. We suggest scripter using these three words with prefix or suffix to differentiate with TAF Pro parameters.

Q: What do I do when I have the same action in many scripts?

A: Coding best practice recommends that naming conventions be applied to scripting within Functional Tester. This will provide the ability to clearly identify any actions that will be used in TAF Pro such that the action name is prefixed to identify which script or specific action it relates too. For example using the action CREATE can be clarified and identify script ownership by added a prefix or suffix, CUST_CREATE or CREATE_CUST to denote the action refers to the customer creation whereas MAT_CREATE or CREATE_MAT would refer to material creation. Applying the naming convention will also eliminate confusion in TAF Pro in correctly identifying the correct action to be used in a scenario.

Comment [BL2]:

Q: Should I change the script template?

A: The following API is in TAF Pro script template, scripter is not required to modify these two methods, and they are for internal TAF Pro use.

```
getResult();         // Returns script execution result to TAF Pro
```

Q: What is the standard way of declaring and saving global variables of values that are passed between scripts?

A: Using getOutput and setOutput methods to pass variable values between scripts.

Q: Where is the best place to put global function code?

A: Sub-class the ScriptSuperClass class provided with the TAF Pro distribution, and extend helper superclass from it.

Q:How does the script save a test result to TAF Pro report files?

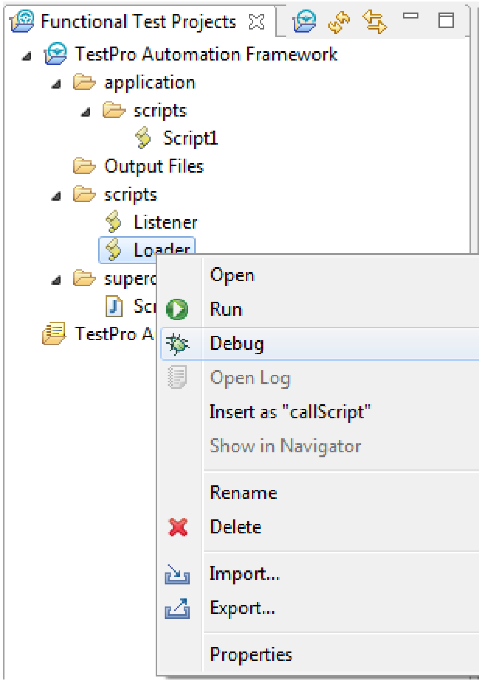
A: Using “setMessage(“Successful message”)” to save successful message in TAF Pro. And using statement “**throw new** Exception(“Failed message)” to save failed message in TAF Pro

Q: Can I use upper and lower case to differentiate dpStrings with the same name?

A: While dpString names in Functional Tester are case sensitive and Functional Tester will recognise them as different by the different text case, databases are not generally case sensitive. This means that an example of CLIENTNAME, ClientName and clientname while all different in Functional Tester, are seen as the same value in a database and will reject as duplicate keys. If similar names are needed the scripter could instead use ClientName versus Client_Name which will be seen as different but may still be confusing to the TAF user. It is recommended to clearly differentiate variable names so the TAF Pro user can understand the data requirement clearly.

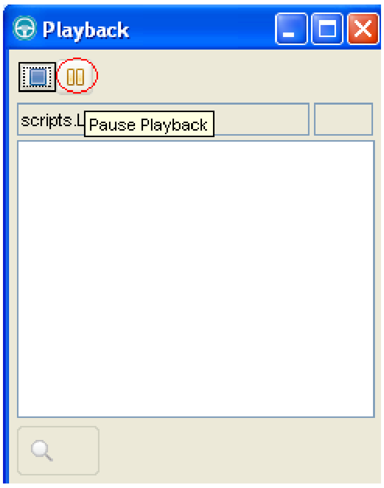
Q: Is it possible for me to step through scripts line by line during running of TAF Pro. Can TAF Pro be run in debug mode?

A: Yes, TAF Pro can be run in debug mode. To run TAF Pro in debug mode, the user needs to start TAF Pro from Functional Tester not from the desktop TAF Pro icon. When starting TAF Pro from Functional Tester, the user needs to run the loader script in debug mode.



Q: Is it possible to “pause” a TAF Pro scenario once it is running? Can I restart it again from the paused state?

A: Yes, user can pause TAF Pro running by clicking pause button in Functional Tester playback window.



Q: If a TAF Pro scenario stops at a particular point, how can I restart it from the correct datapool line rather than simply starting it from scratch?

A: Using filter to create a data set containing all the data after the stopping point, or nominate the starting row when you run the execution suite.

Q: If a script changes in anyway then do I need to re-register it?

A: It depends on the change. If dpString(), setOutput(), getOutput() methods are changed (add, modify, delete), then user needs to re-register the script and re-import data to pick up the change. Otherwise, for any other changes in the script, user only needs to restart TAF Pro to pick up the change.

Q: How can I extract results for a specific date and time period?

A: Simply select the result level for which you want to extract results, right click and select Export Results. Enter the date and time period required and save to a file.

Additional assistance can be obtained by contacting our friendly **TestPro Support** team should you have any questions, issues or feedback. Email support@testpro.com.au.